RQF LEVEL 5

CSAHK501

COMPUTER SYSTEM AND ARCHITECTURE

Hobby Kernel Development

TRAINER'S MANUAL

October, 2024

HOBBY KERNEL DEVELOPMENT

KOICA Korea International Cooperation Agency

TQUM TVET Quality Management Project

2024

AUTHOR'S NOTE PAGE (COPYRIGHT)

Copies available from:

- *HQs: Rwanda TVET Board-RTB*
- *Web: www.rtb.gov.rw*
- **KIGALI-RWANDA**

Original published version: October 2024

ACKNOWLEDGEMENTS

The publisher would like to thank the following for their assistance in the elaboration of this training manual:

Rwanda TVET Board (RTB) extends its appreciation to all parties who contributed to the development of the trainer's and trainee's manuals for the TVET Certificate V in Computer System and Architecture, specifically for the module "**CSAHK501: Hobby Kernel Development**".

We extend our gratitude to KOICA Rwanda for its contribution to the development of these training manuals and for its ongoing support of the TVET system in Rwanda.

We extend our gratitude to the TQUM Project for its financial and technical support in the development of these training manuals.

We would also like to acknowledge the valuable contributions of all TVET trainers and industry practitioners in the development of this training manual.

The management of Rwanda TVET Board extends its appreciation to both its staff and the staff of the TQUM Project for their efforts in coordinating these activities.

This training manual was developed:

Under Rwanda TVET Board (RTB) guiding policies and directives

Under Financial and Technical support of

COORDINATION TEAM

RWAMASIRABO Aimable

MARIA Bernadette M. Ramos

MUTIJIMA Asher Emmanuel

Production Team

Authoring and Review

IKIREZI Pacifique

SIBOMANA Emmanuel

NIYITEGEKA Yves

Validation

NIYOMUFASHA Jean Baptiste

YONKURU Blaise

Conception, Adaptation and Editorial works

HATEGEKIMANA Olivier

GANZA Jean Francois Regis

HARELIMANA Wilson

NZABIRINDA Aimable

DUKUZIMANA Therese

NIYONKURU Sylvestre

HABIMANA Emmanuel

Formatting, Graphics, Illustrations, and infographics

YEONWOO Choe

SUA Lim

SAEM Lee

SOYEON Kim

WONYEONG Jeong

HAKIZAYEZU Adrien

Financial and Technical support

KOICA through TQUM Project

TABLE OF CONTENT

ACRONYMS

CPU: Central Processing Unit

GCC: GNU Compiler Collection

GDB: GNU Debugger

GNU: GNU's Not Unix

KOICA: Korea International Cooperation Agency

LFU: Least Frequently Used

LRU: Least Recently Used

MESI: Modified, Exclusive, Shared, Invalid (cache coherence protocol)

MinGW: Minimalist GNU for Windows

MMU: Memory Management Unit

MOESI: Modified, Owned, Exclusive, Shared, Invalid (cache coherence protocol)

Mysys2: Minimal System 2

NASM: Netwide Assembler

PTE: Page Table Entry

QEMU: Quick Emulator

RAM: Random Access Memory

RTB: Rwanda TVET Board

TQUM Project: TVET Quality Management Project

TVET: Technical and Vocational Education and Training

VS Code: Visual Studio Code

INTRODUCTION

This trainer's manual includes all the methodologies required to effectively deliver the module titled **"Hobby Kernel Development."** Trainees enrolled in this module will engage in practical activities designed to develop and enhance their competencies.

The development of this training manual followed the Competency-Based Training and Assessment (CBT/A) approach, offering ample practical opportunities that mirror real-life situations.

The trainer's manual is organized into Learning Outcomes, which is broken down into indicative content that includes both theoretical and practical activities. It provides detailed information on the key competencies required for each learning outcome, along with the objectives to be achieved.

As a trainer, you will begin by asking questions related to the activities to encourage critical thinking and guide trainees toward real-world applications in the labor market. The manual also outlines essential information such as learning hours, didactic materials, and suggested methodologies.

This manual outlines the procedures and methodologies for guiding trainees through various activities as detailed in their respective trainee manuals. The activities included in this training manual are designed to offer students opportunities for both individual and group work. Upon completing all activities, you will assist trainees in conducting a formative assessment known as the end learning outcome assessment. Ensure that trainees review the key reading and the points to remember section.

MODULE CODE AND TITLE: CSAHK501 HOBBY KERNEL DEVELOPMENT

Learning Outcome 1: Prepare working environment
Learning Outcome 2: Implement Memory management
Learning Outcome 3: Implement Process management
Learning Outcome 4: Implement Persistence management
Learning Outcome 5: Implement core kernel

Indicative contents
1.1 Selection of tools and equipment
1.2 Installation of Software tools
1.3 Configuring virtual environment

Key Competencies for Learning Outcome 1: Prepare Working Environment

Knowledge	Skills	Attitudes
• Description of hobby kernel. • Identification of Hobby Kernel requirements • Identification of the required tools for hobby kernel development. • Identification hardware compatibility.	• Selecting of tools and equipment • Installing required software tools • Configuring required software tools	• Being attentive • Being flexible • Having team work • Having confidence • Being creative

Duration: 10 hrs

Learning outcome 1 objectives:

By the end of the learning outcome, the trainees will be able to:

1. Describe properly hobby kernel based on operating system development process.
2. Identify correctly hobby kernel requirements required in working environment.
3. Identify correctly the required tools and equipment for hobby kernel development.
4. Select correctly tools and equipment used in preparation of working environment.
5. Identify properly hardware and software requirements for hobby kernel development.
6. Install properly software tools used in hobby kernel development environment.
7. Configure correctly installed virtualization tools based on hobby kernel development.

Resources

Equipment	Tools	Materials
• Computer • Projector	• Mysys2 • VS Code • NASM • QEMU • Compiler (GCC, MinGW)	• Internet • Electricity

Advance Preparation:

Before delivering this learning outcome, you are recommended to:

- Avail Msys2, QEMU, NASM, Compiler (GCC, MinGW), VS Code setup.
- Avail compatible computers with required software.

Indicative content 1.1: Selection of Tools and Equipment

Duration: 4hrs

Theoretical Activity 1.1.1: Description of Hobby kernel

Notes to the trainer:

- While delivering this content, a small group can be used for describing Hobby kernel.

Key steps:

While delivering this activity, pass through the following steps:

Step1 Introduce the activity and ask Trainees to answer the following questions:

I. What do you understand about Operating system and kernel?
II. What are the roles of kernel in operating system?
III. What do you understand by hobby kernel?
IV. What are the benefits of hobby kernel development?

Step2 Ask trainees to write answers provided on flip-chart/paper or blackboard.

Step3 Asks Trainees to present their findings

Step4 Provides expert view and clarifies ideas.

Step5 Address any questions or concerns.

Step6 Ask trainees to read the key reading 1.1.1 in the trainee manual.

Points to Remember

- A **hobby kernel** is an operating system created primarily for educational purposes, experimentation, or personal interest, rather than for commercial or mainstream use. It is usually developed by individual enthusiasts or small groups who want to explore the fundamental concepts of operating system (OS) design and implementation.
- **Role of the Kernel in an Operating System:**
  - ✓ Process Management
  - ✓ Memory Management
  - ✓ Input and Output Management
  - ✓ File System Management
- **Benefits of Hobby Kernel Development**
  - ✓ Deep Learning Experience
  - ✓ Hands-On Practice
  - ✓ Experimentation and Innovation

- ✓ Problem-Solving Skills
- ✓ Community Engagement
- ✓ Personal Satisfaction

Theoretical Activity 1.1.2: Identification of hobby kernel requirements

Notes to the trainer:

- While delivering this content, a small group can be used for describing Hobby kernel requirements.

Key steps:

While delivering this activity, pass through the following steps:

step1 Introduce the activity and ask Trainees to answer the following questions:

I. What is operating system architecture?
II. List any 3-operating system architecture
III. Explain virtualization tool in hobby kernel development?
IV. List and explain the development tool use hobby kernel development?

step2 Ask trainees to write answers provided on flip-chart/paper or blackboard.

step3 Provides expert view and clarifies ideas.

step4 Address any questions or concerns.

step5 Ask trainees to read the key reading 1.1.2 in the trainee manual.

Points to Remember

- **The target architecture** refers to the specific computer architecture that a hobby kernel is designed to run on. This choice determines the hardware specifications, the instruction set used, and the overall approach to developing the kernel.
- **Common Target Architectures in Hobby Kernel Development:**
  1. **x86 (32-bit):** Widely used and well-documented, making it a popular choice for beginners.
  2. **64-bit:** Offers more advanced features and is suitable for more complex projects.
  3. **ARM:** Common in mobile and embedded systems, providing a different set of challenges and learning opportunities.

Theoretical Activity 1.1.3: Identification of the right tools and equipment used in development

Notes to the trainer:

- While delivering this content, a small group can be used for identifying the right tools and equipment used in development of Hobby Kernel.

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the activity and ask Trainees to answer the following question:
Outline the examples of tools and equipment used in development of Hobby kernel?

Step 2: Ask Trainees to write answers provided on flip-chart/paper or blackboard.

Step 3: Asks Trainees to discuss the provided answer and choose correct answers.

Step 4: Provides expert view and clarifies ideas.

Step 5: Address any questions or concerns.

Step 6: Ask trainees to read the key reading 1.1.3 in the trainee manual.

Practical Activity 1.1.4: Selection of the right tools and equipment to use in development

Notes to the trainer

- Trainer may use individual based for selecting the right tools and equipment used to develop Hobby kernel.

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the activity and ask trainees to read the following task:
As a trainee in Level 5 computer system and architecture, you are requested to Select right tools and equipment used to develop Hobby Kernel.

Step 2: Explaining the task and provide clear work instructions

Step 3: Demonstrate how to select the right tools and equipment, while demonstrating explain the steps to follow

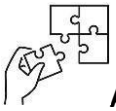
Step 4: Ask trainees to install to select the right tools and equipment and monitoring the procedures

Step 5: Verify whether tools and equipment are selected in correctly manner.

Step 6: Asks trainees to read key readings 1.1.4 on trainee's manual

Points to Remember

- Each tool in hobby kernel development plays a specific role, from emulation and debugging to coding and compiling.
  - ✓ **MSYS2**
  - ✓ **QEMU (Quick Emulator)**
  - ✓ **Visual Studio Code (VSCode)**
  - ✓ **NASM (Netwide Assembler)**
  - ✓ **GCC**



Application of learning 1.1.

As CSA Trainee, near your school there is a person want to develop Hobby Kernel that is supposed to manipulate .txt file. And you are asked you to Select the right tools and equipment to be used while developing Hobby Kernel.

Checklist

Criteria	Indicators /Elements	Observation	
		Yes	No
Selection of tools and equipment for developing Hobby Kernel are well selected	Text editor/IDE is selected		
	GNU Compiler Collection is selected		
	Emulators and Virtual machines are selected		
	Development machine with required requirement is selected		
Decision			

Indicative content 1.2: Installation of Software Tools

Duration: 3 hrs

Practical Activity 1.2.1: Installing software tools

Notes to the trainer

- Avail compatible computer with internet access
- Avail workplace to be used

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the activity and ask trainees to read the task below:
As a Trainee in Level 5 Computer System and Architecture you are requested to install **Msys2, QEMU, NASM, GCC and Visual Studio Code** software tools in computer:

Step 2: Explaining the task and provide clear work instructions

Step 3: Demonstrate how to install the software, while demonstrating explain the steps to follow

Step 4: Ask trainees to install software and monitoring the procedures

Step 5: Verify whether software is installed correctly

Step 6: Asks trainees to read key readings 1.2.1 on trainee's manual

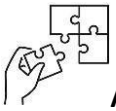
Points to Remember

- **MSYS2 Installation steps are:**
  1. **Download**: Get msys2-x86_64.exe.
  2. **Run Installer**: Requires 64-bit Windows 10 or newer.
  3. **Choose Installation Folder**: Use a short, ASCII-only path on NTFS (no spaces or accents).
  4. **Finish Installation**: Click Finish; UCRT64 terminal launches.
  5. **Update Packages**: Run pacman -Syu.
  6. **Install GCC**: pacman -S mingw-w64-ucrt-x86_64-gcc and confirm with **Y**.
  7. **Verify GCC**: Run gcc --version.
  8. **Re-Update Packages**: Run pacman -Syu.
  9. **Install QEMU**: pacman -S mingw-w64-x86_64-qemu and confirm with **Y**.
  10. **Create Project Folder**: mkdir folder_name (e.g., mkdir hobby_pro).

11. **Navigate to Folder**: cd hobby_pro.
12. **Install NASM**: Run pacman -S nasm.

Visual Studio Code (VS Code) Installation steps are:

1. **Download**: Go to the Visual Studio Code website and click **Download for Windows**.
2. **Run Installer**: Open the downloaded .exe file.
3. **Follow Installation Wizard**:
   - Click **Next** and accept the license.
   - Choose installation location and select tasks (e.g., adding to PATH).
   - Click **Install**.
4. **Launch VS Code**: Check **Launch Visual Studio Code** and click **Finish** or find it in the Start menu.



Application of learning 1.2.

As OS developer you are required to prepare hobby kernel development environment by installing all the basic software tools in the computers remember that these software tools assist in creating virtual machine, providing working space for writing the source codes and another one for compiling c source codes.

Check list

S/N	Assessment criteria (Based on performance criteria)	Indicator/element to check	Observation	
			Yes	No
1.	**MSYS2, QEMU, GCC, NASM and Visual code editor are properly installed.**	MSYS2 is installed		
		Visual code editor is installed		
		NASM is installed		
		GCC is installed		
		NASM is installed		
2.	**The default folder is well created**	Project folder is created		
		NASM packages are installed in project		

Indicative content 1.3: Configuring Virtual Environment

Duration: 3 hrs

Practical Activity 1.3.1: Configuring software tools

Notes to the trainer:

- This activity should take place in a computer lab where trainees should configure the installed software tools.
- Avail computers
- Avail the installed software tools.

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the activity and ask trainees to read the task below:

As a Trainee of Level 5 Computer System and Architecture you are requested to configure **Visual Studio Code** in your computer **by specifying the environment variable Path in MSYS tool**

Step 2: Explaining the task and provide clear work instructions

Step 3: Demonstrate how to configure the software, while demonstrating explain the steps to follow

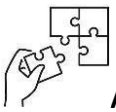
Step 4: Ask trainees to configure the installed software and monitoring the procedures

Step 5: Verify whether the software was configured correctly

Step 6: Asks trainees to read key readings 1.3.1 on trainee's manual

Points to Remember

- **Configuring the path of Visual Studio Code in MSYS using your bashrc file:**
  1. **Open. bashrc**
  2. **Add VS Code to PATH**
  3. **Save and Exit**
  4. **Reload. bashrc**
  5. **Verify configuration**



Application of learning 1.3.

XY company is one of the companies located at Kigali City, it deals with development and configuration of software, want to hire an OS Developer to configure the software in their computers. As the one who skilled the configuration of different software you are requested to configure the Path of VS Code and check if software is working properly in the computer

Checklist:

S/N	Assessment criteria (Based on performance criteria)	Indicator/element to check	Observation	
			Yes	No
1.	**Visual studio code path is well configured**	Bashrc is opened		
		Visual code path is added		
		Path is saved		
		Visual Studio Code is opened using code**.**		

Learning outcome 1 end assessment

Written assessment

1. Choose the correct answer by writing the letter corresponding to the right Answer

i. The primary function of an operating system (OS):

a) Manage hardware components
b) Run only application software
c) Manage all application programs in a computer
d) Provide security features

Answer: C

ii. Which component of an OS acts as a bridge between software applications and hardware?

a) User Interface
b) Shell
c) Kernel
d) Driver

Answer: C

iii. What is a hobby kernel primarily developed for?

a) Commercial use
b) Educational purposes and experimentation
c) High-performance applications
d) Standard operating systems

Answer: B

iv. Which of the following is NOT a benefit of hobby kernel development?

a) Enhanced problem-solving skills
b) Immediate financial gain
c) Community engagement
d) Hands-on practice with low-level programming

Answer: B

v. Which architecture is commonly used in hobby kernel development for beginners?

a) ARM
b) x86 (32-bit)
c) PowerPC
d) SPARC

Answer: B

2. Answer by True or False for the following Questions

a) The kernel is responsible for managing user accounts and preferences.

Answer: False

b) QEMU is a virtualization tool used for testing hobby kernels.

Answer: True

c) A cross-compiler can generate code for the same platform it runs on.
Answer: False

d) VirtualBox can only run Windows operating systems.
Answer: False

e) Completing a functional hobby kernel can provide a sense of personal satisfaction.
Answer: True

4. **Match the following tools with their descriptions and writing number corresponding to the letter in the specified answers column, eg: A-4**

Answers	**Tools**	**Description**
A:-----	A) Cross-Compiler	1. Converts assembly code into machine code
B:…..	B) Assembler	2. Provides a comprehensive environment for coding and debugging
C:….	C) Debugger	3. A tool that allows code compilation for different architectures
D:….	D) IDE	4. Helps identify and fix issues in code during execution
		5. Provide command line interface to write commands to execute nasm

Answers: A-3, B-1, C-4, D-2

Practical assessment

You are a Hobby Kernel developer who has just purchased a new laptop running 64-bit Windows 10. You want to set up a development environment using MSYS2 and Visual Studio Code to compile and run your projects. What specific steps did you follow to successfully install MSYS2 and Visual Studio Code, and how did you configure the environment to ensure that you could run Visual Studio Code from your MSYS2 terminal?

Check list

S/N	Assessment criteria (Based on performance criteria)	Indicator/element to check	Observation	
			Yes	No
1.	MSYS2, QEMU, GCC, NASM and Visual code editor are properly installed.	MSYS2 is installed		
		Visual code editor is installed		
		NASM is installed		
		GCC is installed		
		NASM is installed		
2.	The default folder is well created	Project folder is created		
		NASM packages are installed in project		
3.	Visual studio code path is well configured	Bashrc is opened		
		Visual code path is added		
		Path is saved		
		Visual Studio Code is opened by typing this "**code .**"command in terminal .		

Further information to the trainer

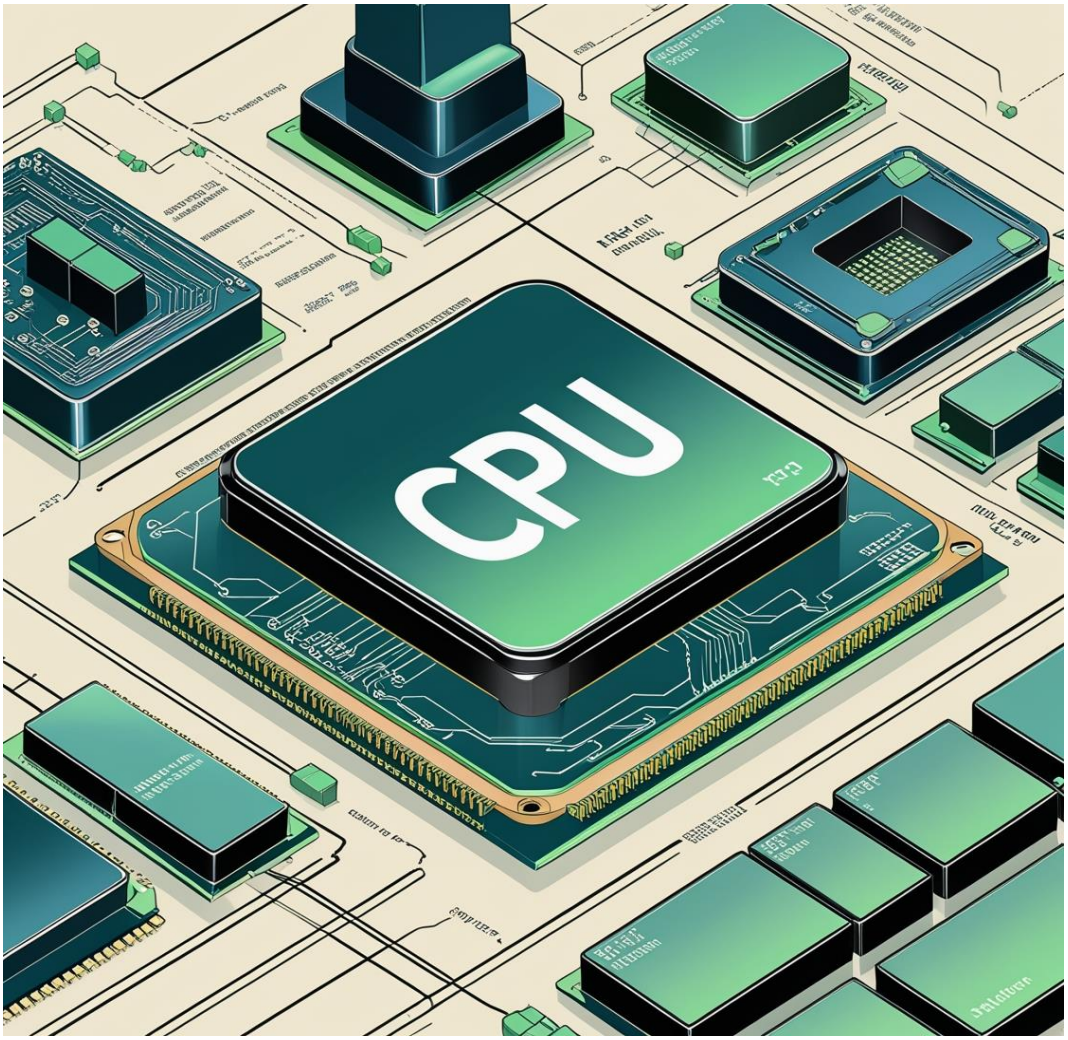
Darnell, L. S. (2017). Create your own operating system. Independently published. https://www.amazon.com/Create-Your-Own-Operating-System/dp/1981624058

Love, R. (2005). Linux kernel development (2nd ed.). Sams Publishing. https://www.amazon.com/Linux-Kernel-Development-Robert-Love/dp/0672327201

Operating Systems: Three Easy Pieces. (n.d.). Pages.cs.wisc.edu. https://pages.cs.wisc.edu/~remzi/OSTEP/

Tanenbaum, A. S., & Woodhull, A. S. (2006). Operating systems: Design and implementation (3rd ed.). Pearson. https://csc-knu.github.io/sys-prog/books/Andrew%20S.%20Tanenbaum%20-%20Operating%20Systems.%20Design%20and%20Implementation.pdf

The Little OS Book. (n.d.). The little book *about OS development*. https://littleosbook.github.io/


CPU

Indicative contents
2.1 Implementation of memory hierarchy and addressing
2.2 Implementation of memory allocation
2.3 Managing Virtual memory and swapping techniques
2.4 Applying defragmentation techniques

Key Competencies for Learning Outcome 2: Implement Memory Management

Knowledge	Skills	Attitudes
• Description of memory addressing. • Description memory allocation and management unit. • Description layout of process address space. • Identification of defragmentation techniques	• Applying memory addressing. • Implementing Cache. • Implementing memory allocation. • Implementation virtual memory. • Applying defragmentation techniques	• Being attention • Being flexible • Having time management • Being organized • Being patience • Having confidence

Duration: 25hrs

Learning outcome 2 objectives:

By the end of the learning outcome, the trainees will be able to:

1. Describe Correctly memory addressing and its model as used in memory management
2. Describe correctly memory allocation management unit as used in memory management
3. Describe correctly the layout of process address space as used in memory management
4. Identify properly defragmentation techniques as used in memory management
5. Apply properly memory addressing in the context of memory management
6. Implement correctly Cashe in the context of memory management
7. Implement correctly memory allocation in the context of memory management
8. Implement correctly virtual memory as used in memory management
9. Apply properly defragmentation techniques used in memory management

Resources

Equipment	Tools	Materials
• Computer • Projector	• VS Code • MSYS2 • GCC • QEMU	• Internet • Electricity

Advance Preparation:

Before delivering this learning outcome, you are recommended to:

- Avail Computer with installed the following software tools: Msys2, QEMU, GCC and VS Code.

Indicative content 2.1: Implementation of Memory Hierarchy and Addressing

Duration: 5 hrs

Theoretical Activity 2.1.1: Description of memory hierarchy and addressing

Notes to the trainer:

- Trainer may use a small group to describe memory hierarchy and addressing

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the activity and ask Trainee to answer the following questions:

i. What is Memory Hierarchy
ii. What is cash memory
iii. What is Memory addressing
iv. List the replacement polices used in cash memory
v. What is Snooping

Step 2: Ask learner to write answers provided on flipchart/paper

Step 3: Asks Trainees to discuss the provided answer and choose correct answers.

Step 4: Provides expert view and clarifies ideas.

Step 5: Address any questions or concerns

Step 6: Ask trainees to read the key reading 2.1.1 in the trainee manual.

Points to Remember

1. Definition

- **Memory Hierarchy**: Structured organization of memory types (cache, RAM, disk) for optimized speed and efficiency.
- **Kernel Management**: Ensures quick data access for processes while maintaining system performance.

2. Memory Hierarchy Components

- **Registers**: Fast storage for immediate computations by the CPU.
- **Cache**: Small, fast memory for frequently accessed data (e.g., L1 cache).
- **Main Memory (RAM)**: Active processes and kernel data structures.
- **Secondary Storage**: Persistent data storage (e.g., files on disk).

3. Addressing Modes

- **Immediate Addressing**: Direct values for quick access.
- **Direct Addressing**: Memory locations specified directly in instructions.
- **Indirect Addressing**: Using pointers/references for data access.
- **Indexed Addressing**: Base address combined with an offset for arrays.

4. Cache Management Mechanisms

- **Cache Initialization**: Set cache size and associativity at startup.
- **Cache Hit/Miss Logic**: Check for data in cache; load from RAM on a miss.
- **Cache Coherency**: Maintain consistency for shared data among processes.

5. Cache Replacement Policies

- **Least Recently Used (LRU)**: Evict the least recently used entry.
- **First In, First Out (FIFO):** Evict the oldest cache entry.
- **Random Replacement**: Randomly evict an entry to make space.

6. Snooping in Shared Memory

- **Shared Memory Management:** Basic lock/flag mechanism for consistency among processes.
- **Invalidate on Write**: Update or invalidate cache entries when shared data is modified by a process.

Practical Activity 2.1.2: Applying memory addressing

Notes to the trainer

- The trainer may use a small group to apply memory addressing

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the activity and ask Trainees to answer the following questions:
you are requested to go in computer lab and write a c program that will perform the following tasks related to the Hobby Kernel:

I. Use address translation scheme?
II. Use direct addressing
III. Use indirect addressing
IV. Use indexing addressing memory?
V. Use register addressing

Step 2: Explain the task and provide clear work instruction.

Step 3: Ask trainees to read key reading 2.1.2 in the Trainee's manual.

Step 4: Demonstrate how to apply memory addressing. While demonstrating, explain the steps to follow

Step 5: Ask Trainee to address memory

Step 6: Verify whether memory is addressed

Points to Remember

Address Translation Scheme

- **Logical to Physical Mapping**: Use a page table for address translation in a flat memory model.

Practical Activity 2.1.3: Implementation of Cache

Notes to the trainer

- Trainer may use a small group to Develop the Algorithm, implement cache coherency using MESI or MOESI protocols and testing Cashe

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the activity and ask Trainees to answer the following questions:

i. Write a c program that will use the following Algorithms:
   a. LRU
   b. LFU
ii. Write a program that will implement Cashe Coherency using MESI and MOESI Protocols
iii. Write a c program that will Test the Cashe in your computer

Step 2: Explain the task and provide clear work instruction.

Step 3: Ask trainees to read key reading 2.1.3 in the Trainee's manual.

Step 3: Demonstrate how to implement Cashe Coherency using MESI and MOESI Protocols

Step 4: Ask learner to Test Cashe

Step 5: Verify whether Cashe is implemented

Step 6: Ask Learner to read

Points to Remember

Cache Implementations

- **LRU Cache**: Track usage order and evict as needed.
- **LFU Cache**: Track access frequency for eviction decisions.
- **MESI Protocol**: Maintain cache coherency with states (Modified, Exclusive, Shared, Invalid).

Practical Activity 2.1.4: Applying Memory Mapping

Notes to the trainer

- This activity should take place in a computer lab where trainees should apply Memory Mapping by using Paging and Segmentation
- While delivering this content, you are required to:
    1. Avail computer connected to the internet.
    2. Avail computer installed with Development Tools, Debugging and Profiling Tools and Visualization Tools

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the topic and ask Trainees do the task described below:

As a Hobby Kernel Developer, you are asked to go to the computer lab for Applying Memory Mapping using paging and segmentation

Step 2: Explain the task and provide clear work instruction.

Step 2: Explain the task and provide clear work instruction.

Step 3: Ask trainees to read key reading 2.1.4.

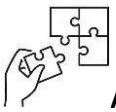
Step 4: Demonstrate how to apply Memory Mapping by using Paging and segmentation.

Step 4: Verify whether the Memory Mapping is applied

Points to Remember

Memory Mapping

- **Paging**: Fixed-size blocks (pages) mapped to physical memory; uses a page table.
- **Segmentation**: Variable-sized segments based on logical organization; uses a segment table.



Application of learning 2.1.

You are designing a simple operating system for an embedded system with limited memory resources. Your system needs to manage multiple applications that have varying memory requirements and access patterns. Provide a design outline that includes data structures for both paging and segmentation.

Include a brief discussion of how you would handle a scenario where a segment needs to grow beyond its initially allocated size.

Checklist

S/N	Assessment criteria (Based on performance criteria)	Indicator/element to check	Observation	
			yes	No
1.	**Memory addressing is well applied**	Direct address is applied		
		Indirect address is applied		
		Indexed address is applied		
		Register address is applied		
2.	**Cash memory is well implemented**	LRU and LFU algorithm are used		
		Cash coherency are implemented		
		Cash is tested		
3.	**Memory mapping is well applied**	Paging is applied		
		Segmentation is applied		

Indicative content 2.2: Implementation of memory allocation

Duration: 5hrs

Theoretical Activity 2.2.1: Description of Memory Management Unit

Notes to the trainer:

- While delivering this content, a small group can be used to describe Memory Management unit.

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the activity and request Trainees to answer to the following questions:

I. What is memory management unit?
II. What is the role of memory management unit?
III. Explain the structure of page table

Step 2: Explain the task and provide clear work instruction.

Step 3: Provides expert view and clarifies ideas by using didactic materials.

Step 4: Ask trainees to read the Key readings 2.2.1 in their manuals.

Points to Remember

- **The Memory Management Unit (MMU)** is a crucial component in computer architecture that handles all memory-related operations for a processor. Its primary responsibilities include managing the translation of virtual addresses to physical addresses, controlling access to memory, and ensuring efficient memory utilization.
- **Key Functions of the MMU:**
    1. Address Translation:
    2. Paging and Segmentation
    3. Access Control
    4. Memory Protection
    5. Cache Management
- **The page table** is a crucial data structure used in virtual memory systems to manage the mapping between virtual addresses and physical memory addresses. Its structure can vary based on system architecture, memory requirements, and design choices, influencing performance and resource utilization

Practical Activity 2.2.2: Implementing Page Table Management

Notes to the trainer

- The trainer may use small group to implement page table

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the topic and ask Trainees to read the task described below:

As a Trainee of level 5 Computer System and Architecture, you are asked to go to the computer lab and implement Data structures and use Algorithm for managing page tables by using C language.

Step 2: Explain the task and provide clear work instruction.

Step 3: Ask trainees to to implement data structure and how to use Algorithm in management of page tables

Step 4: Demonstrate how to implement data structure and how to use Algorithm in management of page tables

Step 5: Verify whether the data structure is implemented and if the Algorithm is used in management of page tables.

Step 6: Ask trainees to read key reading 2.2.2

Points to Remember

- **The effective management of page tables** involves careful allocation, translation, fault handling, and maintenance processes tailored to the needs of the operating system and the workload.
- **Managing page tables** involves several algorithms for various tasks, such as page replacement, page allocation, and address translation

Theoretical Activity 2.2.3: Description of Memory Allocation

Notes to the trainer:

- While delivering this content, a small group can be used to describe Memory Allocation

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Trainer ask Learner to answer the following questions:

1.what is memory allocation

2.what are the strategies used in memory allocation

Step 2: Ask trainees to present their findings to the whole class

Step 3: Provides expert view and clarifies ideas by using didactic materials

Step 4: Ask trainees to read key readings in activity 2.2.3

Points to Remember

- **Memory allocation** is the process of reserving a portion of computer memory for a program to use while it is running.
- There are two main types of memory allocation which are Static Allocation and Dynamic Allocation
- **Memory allocation strategies** refer to the techniques used to allocate memory efficiently and manage memory usage in computing systems.
- Here are some common strategies:
    1. Static Allocation
    2. Dynamic Allocation
    3. First-Fit
    4. Best-Fit
    5. Worst-Fit
    6. Buddy System
    7. Paging
    8. Segmentation
    9. Garbage Collection
    10. Memory Pooling

Practical Activity 2.2.4: Apply Memory Allocation

Notes to the trainer

- While delivering this content, a small group can be used to Apply Memory Allocation

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the topic and ask Trainees to read the task described below:
As a Hobby Kernel Developer, you are asked to go to the computer lab and allocate the process to the memory by using memory allocation algorithms

Step 2: Explain the task and provide clear work instruction.

Step 3: Ask trainees to use memory allocation Algorithm such as first-fit, best-fit or worst-fit

Step 4: Ask trainees to read key reading 2.2.4

Step 5: Demonstrate how to allocate memory by using first-fit, best-fit and worst-fit algorithms

Step 6: Verify whether the memory allocation algorithms are applied

Points to Remember

- **Memory allocation algorithms are techniques** used to allocate memory in a system efficiently some common algorithms:
    1. First-Fit Algorithm
    2. Best-Fit Algorithm
    3. Worst-Fit Algorithm
    4. Next-Fit Algorithm

- **Test cases in memory allocation** are structured scenarios designed to verify the functionality, reliability, and performance of memory allocation mechanisms within a program or system. These test cases help ensure that memory management behaves as expected under various conditions and edge cases

Theoretical Activity 2.2.5: Description of Memory Protection

Notes to the trainer:

- While delivering this content, a small group can be used to describe Memory Protection

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Trainer Involves trainees to form groups.

Step 2: Trainer ask Learner to answer the following questions:

1. what is memory protection
2. what is Memory regions
2. what are the memory protection Mechanisms

Step 3: Ask trainees to present their findings to the whole class

Step 4: Provides expert view and clarifies ideas by using didactic materials

Step 5: Ask trainees to read key readings in activity 2.2.5 in Trainee' manual

Points to Remember

- **Memory protection** is a mechanism that restricts access to certain areas of memory, ensuring that processes operate within their allocated space and do not interfere with each other or with the operating system.
- **Memory regions** refer to distinct areas within a computer's memory that are designated for specific purposes or types of data. Understanding memory regions is essential for managing memory effectively in programming and operating systems
- **List of common memory protection mechanisms used in operating systems to safeguard memory regions and manage access rights:**
  1. Paging
  2. Segmentation
  3. Access Control Lists (ACLs)
  4. Memory Mapping
  5. User and Kernel Mode
  6. Hardware Memory Protection
  7. Stack Guard/Stack Canaries
  8. Address Space Layout Randomization (ASLR)
  9. Code Signing and Execution Prevention
  10. Memory Isolation

Practical Activity 2.2.6: Applying Memory Protection

Notes to the trainer

- While delivering this content, a small group can be used to Apply Memory Protection

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the topic and ask Trainees do the task described below:
As a Hobby Kernel Developer, you are asked to go to the computer lab to set the access permission and use hardware-based memory protection

Step 2: Explain the task and provide clear work instruction.

Step 3: Ask trainees to read key reading 2.2.6 in the Trainee's Manual

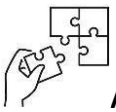
Step 4: Ask trainees to to set permission on memory and use hardware –based on memory protection

Step 5: Demonstrate how to set the permission on memory and use hardware –based on memory protected

Step 6: Verify whether the memory is protected

Points to Remember

- Setting access permissions on memory is essential for protecting sensitive data and preventing unauthorized access
- Using hardware-based memory protection involves leveraging features provided by the CPU and the operating system to safeguard memory access



Application of learning 2.2.

You are developing a simple memory management system for a hobby operating system kernel. The system needs to efficiently allocate and deallocate memory for various processes, ensuring minimal fragmentation and optimal performance.

1. Write a C function that implements the basic allocation logic (e.g., first-fit, best-fit, or buddy system), including handling requests for different block sizes.
2. Write a C function that takes a pointer to a previously allocated block and updates the internal data structures to mark the block as free. Include logic for coalescing adjacent free blocks to reduce fragmentation.
3. Discuss how you would monitor and manage fragmentation in your memory allocator. Implement a function that scans the memory and reports the total used, free, and fragmented space. What strategies might you use to mitigate fragmentation over time?

Additional Instructions

1. Include appropriate data structures (e.g., linked lists, bitmaps) to track allocated and free memory blocks.
2. Provide detailed comments in your C code to explain your logic and implementation choices.
3. Discuss how you would test your memory allocation system to ensure its correctness and performance under different scenarios.

Checklist

S/N	Assessment criteria (Based on performance criteria)	Indicator/element to check	Observation	
			yes	No
1.	**Memory allocation is well applied**	First-fit is applied		
		best-fit is applied		
		buddy system is applied		
		different block sizes requests are handled		
2.	**Deallocation of Memory is well applied**	Block is updated		
		Fragmentation is reduced		
		free blocks are created		

Indicative content 2.3: Managing Layout of Process Address Space and Protection

Duration: 5hrs

Theoretical Activity 2.3.1: Description of Processing address space

Notes to the trainer:

- While delivering this content, a small group can be used to manage the process address space and protection

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the activity and ask trainees to answer the following questions:

1. what is process
2. what is Memory-based attacks
3. Manage Address Space Layout Randomization (ASLR) Settings
4. how to manage Process

Step 2: Ask trainees to present their findings to the whole class

Step 3: Provides expert view and clarifies ideas by using didactic materials

Step 4: Ask trainees to read key readings in activity 2.3.1 in Trainee 'manual

Points to Remember

- **Managing the layout of a process address space and ensuring protection** involves a combination of understanding memory organization, using system features, and implementing best practices in software design.
- Memory-based attacks exploit vulnerabilities in a program's memory management to gain unauthorized access, manipulate data, or execute arbitrary code
- **Effectively managing ASLR settings** enhances the security posture of your systems by making it harder for attackers to predict memory layouts. Regularly check and configure ASLR settings as part of your security practices, ensuring that applications are developed with these protections in mind
- Managing processes effectively is crucial for system performance, resource utilization, and application stability

Practical Activity 2.3.2: Applying Stack and Heap algorithms

Notes to the trainer

- While delivering this content, a small group can be used to Apply Memory Protection

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the topic and ask Trainees do the task described below:

As a Hobby Kernel Developer, you are asked to go to the computer lab to Write a C program that apply Stack and Heap algorithms.

Step 2: Explain the task and provide clear work instruction.

Step 3: Ask trainees to read key reading 2.3.2 in the Trainee's Manual

Step 4: Ask trainees to to Write a C program that apply Stack and Heap algorithms.

Step 5: Verify whether Stack and Heap algorithm is used in C program.

Points to Remember

- **The stack** is a region of memory used for dynamic memory allocation that follows the Last In, First Out (LIFO) principle. It is primarily utilized for managing function calls, local variables, and control flow
- **The heap** is an area of memory used for dynamic allocation of memory blocks. Unlike the stack, which is managed automatically, the heap allows for flexible memory allocation and deallocation at runtime.
- Common stack algorithms:
  Push, Pop, Peek/Top, isEmpty, **Size**:

Theoretical Activity 2.3.3: Description of Memory usage optimization

Notes to the trainer:

- While delivering this content, a small group can be used to describe Memory usage optimization

Key steps:

While delivering this activity, pass through the following steps:

Step 1. Introduce the activity and ask trainees to answer the following question:

Describe the memory usage optimization

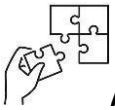
Step 2. Ask trainees to present their findings to the whole class

Step 3. Provides expert view and clarifies ideas by using didactic materials

Step 4. Ask trainees to read key readings in activity 2.3.3 in Trainee 'manual

Points to Remember

- **Memory usage optimization** is crucial for improving the performance and efficiency of applications
- Several strategies and techniques to optimize memory:
  1. Efficient Memory Allocation
  2. Minimizing Memory Footprint
  3. Lazy Loading
  4. Use of Memory Pools
  4. Memory Compaction
  6. Smart Caching
  7. Monitoring and Profiling
  8. Optimize Data Structures
  9. Reclaiming Memory
  10. Kernel Configuration



Application of learning 2.3.

A software development company face a challenge of priotorization of processes on the memory. As Kernel developer you are requested to write a C program to apply Stack and heap algorithm.

Checklist

S/N	Assessment criteria (Based on performance criteria)	Indicator/element to check	Observation	
			yes	No
1.	**Stack and Heap algorithms are well applied**	Stack algorithms is well applied		
		Heap algorithms is well applied		

Indicative content 2.4: Applying Virtual memory and swapping techniques

Duration: 5 hrs

Theoretical Activity 2.4.1: Description of Virtual memory and Swapping techniques

Notes to the trainer:

- While delivering this content, a small group can be used to describe virtual memory and Swapping techniques

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the activity and ask trainees to answer the following questions:

1. What is virtual memory
2. What is Swapping
3. What are the techniques used in swapping?

Step 2: Ask trainees to present their findings to the whole class

Step 3: Provides expert view and clarifies ideas by using didactic materials

Step 4: Ask trainees to read key readings in activity 2.4.1 in Trainee 's manual

Points to Remember

- **Virtual memory** is a memory management technique used by operating systems to create the illusion of a larger amount of memory than is physically available on a computer. It does this by using a combination of hardware and software to allow programs to use memory addresses that may not correspond directly to actual physical RAM.
- **Swapping** is a memory management technique used by operating systems to optimize the use of RAM when the physical memory is full. It involves moving inactive or less frequently used pages of memory from RAM to a designated area on the disk, known as swap space or a swap file

Practical Activity 2.4.2: Applying Page Replacement Algorithms

Notes to the trainer

- This activity should take place in a computer lab where trainees should apply page Replacement Algorithms

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the topic and ask trainees to read the following tasks:
As software developer, you are requested to go to the computer lab to apply page Replacement such as FIFO, LRU AND Clock algorithm

Step 2: Provide instruction that will be followed

Step 3: Demonstrate how to apply page Replacement algorithm, while demonstrating, explain the steps to follow.

Step 4: Asks Trainees to apply page Replacement algorithm and monitor the procedures.

Step 5: Verify whether FIFO, LRU and Clock algorithm are clearly applied

Step 6: Ask trainees to read key reading 2.4.2. in the Trainee's Manual

Points to Remember

- **FIFO, or First-In-First-Out,** is a page replacement algorithm used in operating systems to manage how pages are swapped in and out of physical memory (RAM)
- **The Least Recently Used (LRU) algorithm** is a page replacement strategy that removes the page that has not been used for the longest period of time.
- **The Clock algorithm** is an approximation of the LRU algorithm that is more efficient in terms of memory and speed. It uses a circular queue and a "use bit" to keep track of page usage.

Practical Activity 2.4.3: Apply Demand Paging

Notes to the trainer:

- While delivering this content, a small group can be used for introduce Demand Paging
- The use of videos as didactic materials is required.

Key steps:

While delivering this activity, pass through the following steps:

Step 1. You are requested to answer the following questions related the Applying Demand Paging:
As a Hobby Kernel Developer, you are asked to go to the computer lab to apply demand Paging.

Step 2. Participate in group formulation

Step 3. Present your findings to your classmates and trainer

Step 4. For more clarification, read the key readings 2.4.3. In addition, ask questions where necessary.

Points to Remember

- **Demand paging** is a memory management technique used in operating systems to load pages into memory only when they are needed, rather than preloading all pages at the start. This approach helps optimize the use of physical memory by only loading the parts of a program that are currently required, which can improve performance and reduce memory usage.
- **A page fault** is an event that occurs when a program tries to access a page of memory that is not currently loaded in physical RAM. When this happens, the operating system must intervene to handle the fault**.**
- Setting the required pages into memory involves several steps within the context of demand paging.
- Steps to Handle a Page Fault:
    1. Detect the Page Fault
    2. Save the Context
    3. Check the Page Table
    4. Locate the Page on Disk
    5. Allocate a Frame in RAM
    6. Page Replacement (if necessary)
    7. Load the Required Page
    8. Update the Page Table
    9. Restore Context

10. Resume Execution

Practical Activity 2.4.4: Applying Swap Space Management

Notes to the trainer

- This activity should take place in a computer lab where trainees should select the Techniques to manage swap space and reduce disk i/o overhead

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the topic and ask Trainees read the task described below:

As a Hobby Kernel Developer, you are asked to go to the computer lab to manage the swap space and reduce disk overhead

Step 2: Explain the task and provide clear work instruction.

Step 3: Demonstrate how to manage the swap space and reduce disk overhead

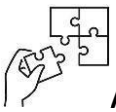
Step 4: Ask Trainees to manage swap space on memory and reduce disk overhead.

Step 5: Verify whether are swap space on memory and disk overhead are correctly managed.

Step 6: Ask trainees to read key reading 2.4.4.in the Trainee's manual

Points to Remember

- **Managing swap space** and reducing disk I/O overhead are crucial for optimizing system performance in environments that utilize virtual memory.
- **Several techniques to achieve these goals:**
  1. Efficient Page Replacement Algorithms
  2. Swapping Strategies
  3. Pre-Paging
  4. Use of a Larger RAM
  5. Memory Compression
  6. Separate Swap Space
  7. Reduce Fragmentation
  8. Optimize Disk I/O
  9. File System Tuning
  10. Memory Mapping
  11. Application-Level Optimization



Application of learning 2.4.

You are developing a hobby operating system kernel for an embedded system that requires efficient memory management to run multiple user applications simultaneously. To optimize memory usage, you decide to implement virtual memory and swapping techniques. Your kernel will allow processes to use more memory than is physically available by leveraging disk space.

1. Write a C function to handle page table entries and simulate the mapping of virtual addresses to physical addresses.
2. Write a C function that simulates the actions taken when a page fault occurs, including checking the page table, loading the required page from disk, and updating the page table.
3. Write a C function that simulates swapping a process out to disk when memory is full, as well as bringing it back into memory when needed.

Additional Instructions

1. For each part, consider aspects such as memory allocation, maintaining page tables, and ensuring that the kernel remains responsive during these operations.
2. Provide comments in your C code to explain the logic and decisions made during the implementation.
3. Discuss how you would test your virtual memory and swapping mechanisms to ensure they work correctly and efficiently.

Checklist

S/N	Assessment criteria (Based on performance criteria)	Indicator/element to check	Observation	
			yes	No
1.	**Memory allocation is well applied**	First-fit is applied		
		best-fit is applied		
		buddy system is applied		
		different block sizes requests are handled		
2.	**Deallocation of Memory is well applied**	Block is updated		
		Fragmentation is reduced		
		free blocks are created		

Indicative content 2.5: Applying defragmentation techniques

Duration: 5 hrs

Theoretical Activity 2.5.1: Description of Defragmentation

Notes to the trainer:

- While delivering this content, a small group can be used for introduce Demand Paging
- The use of videos as didactic materials is required.

Key steps:

While delivering this activity, pass through the following steps:

Step1: Involve trainees in group formulation

Step 2: Introduce the activity and ask trainees to answer to the following questions:

1. What do you understand about defragmentation?
2. What are the types of Defragmentation?
3. What are the techniques used to Defragment memory?

Step 3: ask trainees to present their findings

Step 4: Provides expert view and clarifies ideas by using didactic materials

Step 5: Ask trainees to read the Key readings 2.5.1 in their manuals

Points to Remember

- **Defragmentation** is a process used to reorganize the data on a storage device (like a hard drive) to improve access speed and overall performance. Over time, as files are created, modified, and deleted, they can become fragmented. This means that parts of a single file may be scattered across different areas of the disk rather than being stored in a contiguous block
- **The main types of defragmentation:**
  1. File Defragmentation
  2. Free Space Defragmentation
  3. System Defragmentation
  4. Disk Defragmentation
  5. Online Defragmentation
  6. Offline Defragmentation
  7. Solid State Drive (SSD) Optimization

- **Techniques used to Defragment memory:**
    1. Memory Compaction
    2. Paging
    3. Segmentation
    4. Garbage Collection
    5. Allocation Strategies
    6. Defragmentation Tools
    7. Hybrid Memory Management

Practical Activity 2.5.2: Performing defragmentation

Notes to the trainer

- This activity should take place in a computer lab where trainees should perform Defragmentation of memory.

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the topic and ask trainees read the task described below:
As a Hobby Kernel Developer, you are asked to go to the computer lab to perform Memory defragmentation

Step 2: Explain the task and provide clear work instruction

Step 3: Demonstrate how to perform memory defragmentation and monitor the procedure

Step 4: Ask Trainees to perform memory compaction, Buddy system allocation and memory pooling

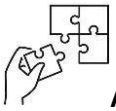
Step 5: Verify whether memory compaction, Buddy system allocation and memory pooling are correctly performed

Step 6: **Step 6:** Ask trainees to read key reading 2.5.2.

Points to Remember

- **Memory compaction** is a process used in computer memory management to eliminate fragmentation and improve the efficiency of memory usage
- **How Memory Compaction Works:**
    1. Identify Fragmentation
    2. Relocate Active Data
    3. Up Free Up Space
    4. date References

- **The Buddy System** is a memory allocation scheme that helps manage dynamic memory in a way that minimizes fragmentation and allows efficient allocation and deallocation of memory blocks.
- **Use Cases:** The Buddy System is often used in operating systems and memory allocators where dynamic memory allocation is crucial, such as in embedded systems or real-time applications.
- **Memory pooling** is a memory management technique designed to improve performance and reduce fragmentation by allocating memory in large blocks, or pools, instead of allocating and deallocating small chunks of memory individually.



Application of learning 2.5.

You are working on a hobby operating system kernel designed for embedded systems with limited resources. The kernel needs to efficiently manage memory for various tasks, including running user applications, managing device drivers, and supporting real-time processing. You notice that as the system runs, memory fragmentation increases, leading to performance issues and inefficient memory usage.

1. Write a C program to implement defragmentation and memory compaction in your kernel to address fragmentation?
2. Write c language to implement the buddy system allocation to efficiently manage memory for processes of varying sizes in your kernel.
3. How would you implement a memory pool for fixed-size allocations using C language?

Checklist

S/N	Assessment criteria (Based on performance criteria)	Indicator/element to check	Observation	
			yes	No
1.	**Memory management is well applied**	Free space is managed		
		Memory is allocated		
		Memory compaction is applied		
2.	**Defragmentation is well applied**	Fragment is removed		
		Fragmentation is reduced		
		buddy system is applied		

Learning outcome 2 end assessment

Written assessment

I) **Multiple Choice Questions, select the best answer by circling the correct letters**

1.What is Memory Hierarchy?

A) A method of organizing data in a database

B) A system of storing data at different speeds and costs

C) A technique for memory encryption

D) A programming language feature

2.What is cache memory?

A) A permanent storage device

B) A type of fast, temporary storage for frequently accessed data

C) An error-checking mechanism

D) A method for data compression

3.What is Memory Addressing?

A) The method used to encrypt data

B) The technique for specifying locations in memory

C) The process of compressing files

D) A type of network configuration

4.Which of the following is NOT a cache replacement policy?

A) LRU (Least Recently Used)

B) FIFO (First in, First Out)

C) Random Replacement

D) Sequential Replacement

5.What is Snooping?

A) Monitoring cache states in a multiprocessor environment

B) A method of data encryption

C) A technique for file compression

D) A programming error

6.What is a Memory Management Unit (MMU)?

A) A device that handles graphics processing

B) A hardware component that manages memory addressing and protection

C) A storage device

D) A network interface

7.Which of the following is a role of the Memory Management Unit?

A) Manage user interfaces

B) Translate virtual addresses to physical addresses

C) Encrypt data

D) Process input/output operations

II)Answer by True or False for the following Questions

1.Cache memory is slower than main memory. **False**

2.Memory protection ensures that processes cannot interfere with each other's memory space. **True**

3.Virtual memory allows programs to access more memory than physically available. **True**

4.Demand paging loads all pages of a process into memory at once. **False**

5.Defragmentation is only applicable to hard drives, not RAM. **True**

III)Matching Questions

Match the following terms with their definitions:

Answers	Terms	Definition
1: C	1. Stack	A) A technique to manage memory dynamically
2:A	2. Heap	B) The process of reorganizing fragmented data to improve performance
3:D	3. Swapping	C) A type of memory that grows downwards for local variables
4:E	4. Page Fault	D) Moving processes between main memory and disk
5:B	5. Defragmentation	E) An error that occurs when a program accesses an invalid page

Practical assessment

Building a Simple Kernel Memory Management System

You are tasked with developing a simple memory management system for a hobby operating system kernel targeting an embedded device. Your goal is to implement a comprehensive memory management framework that includes memory addressing, caching, memory allocation, virtual memory, and fragmentation management.

Tasks:

1.Memory Addressing:

Design a system that supports a flat memory model. Implement a function that maps virtual addresses to physical addresses. Consider how you will handle address spaces for multiple processes.

Deliverable: Write a C function that translates a virtual address to a physical address using a simple page table approach.

2.Cache Implementation:

Implement a basic cache mechanism to speed up memory access for frequently used data. Use a direct-mapped cache structure for simplicity.

Deliverable: Write a C function to handle cache reads and writes, including cache miss handling and updating the main memory.

3.Memory Allocation:

Develop a simple memory allocation strategy for managing heap memory. Implement a first-fit allocator that can handle allocation and deallocation requests.
Deliverable: Write C functions for allocating and deallocating memory, ensuring to manage the free list and prevent fragmentation.

4.Virtual Memory:

Integrate a virtual memory system that allows processes to utilize more memory than is physically available. Implement page fault handling and a simple swapping mechanism.
Deliverable: Write C functions that handle page faults, including checking the page table, loading pages from disk, and updating the page table.

5.Defragmentation Techniques:

Implement techniques to manage memory fragmentation. This could include coalescing free blocks upon deallocation and periodic compaction of memory.
Deliverable: Write C functions for coalescing free memory blocks and an optional function for compacting memory.

Checklist

S/N	Assessment criteria (Based on performance criteria)	Indicator/element to check	Observation	
			yes	No
1.	**Memory Addressing is well applied**	Address spaces is created		
		Virtual address is translated to physical		
2.	**Cache Implementation is well applied**	Cache read is applied		
		Cache write is applied		
		Cache is updating the main memory		
3.	**Memory Allocation is well applied**	first-fit is applied		
		Process is allocating free space		
		Fragmentation is prevented		
4.	**Virtual Memory is well applied**	Virtual memory is created		
		page fault is handled		
		Pages are loaded		
5	**Defragmentation Techniques is well applied**	Free space is removed		
		Fragment is overcoming		

Further information to the trainer

Bovet, D. P., & Cesati, M. (2005). *Understanding the Linux kernel* (3rd ed.). O'Reilly Media. https://www.cs.utexas.edu/~rossbach/cs380p/papers/ulk3.pdf

IEEE Xplore, https://ieeexplore.ieee.org/abstract/document/7378320

Lighthousand. (2014). *books/Modern Operating Systems 4th Edition--Andrew Tanenbaum.pdf at master · lighthousand/books*. GitHub. https://github.com/lighthousand/books/blob/master/Modern%20Operating%20Systems%204th%20Edition--Andrew%20Tanenbaum.pdf

Niemann, T. (1998). *Memory management: Algorithms and implementation in C/C++*. Creative Commons. https://training.brussels/wp-content/uploads/2014/02/memory-management-algorithms-and-implementation-in-c-c-2002-en.pdf

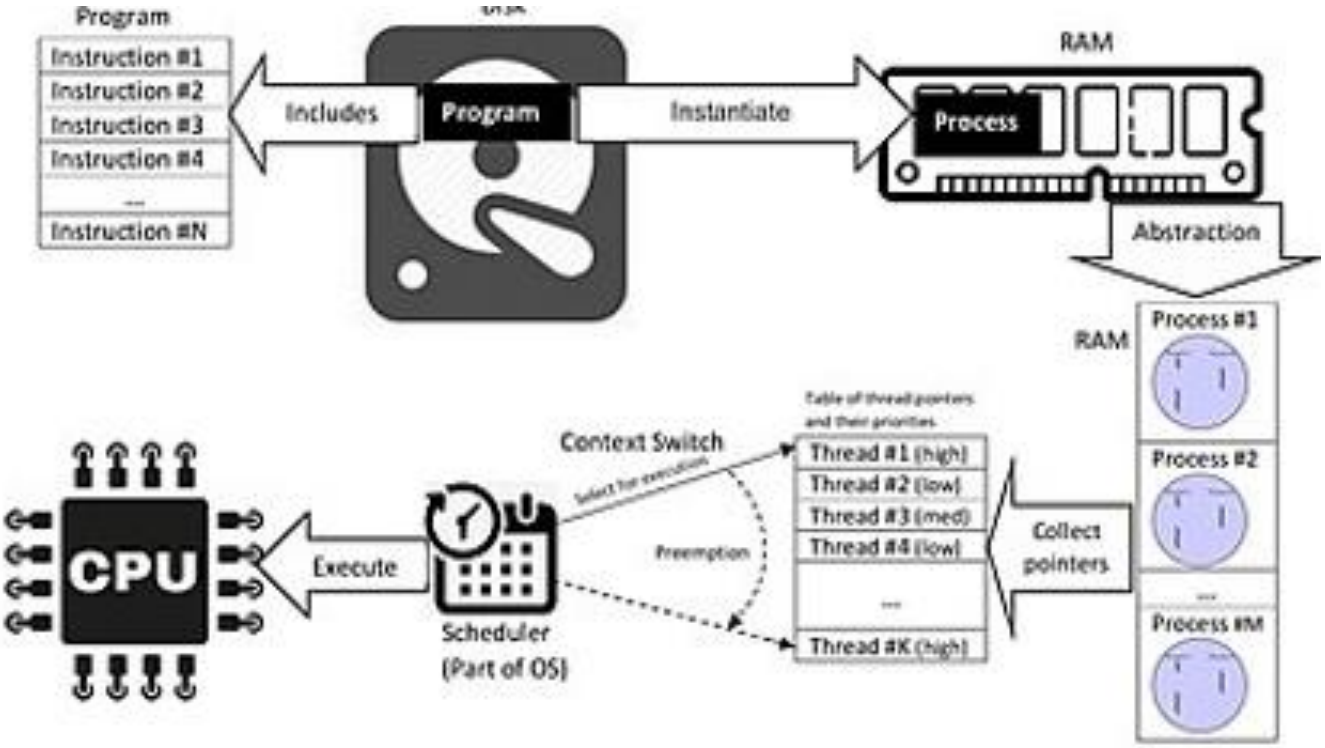
Operating Systems: Three Easy Pieces. (n.d.). Pages.cs.wisc.edu. https://pages.cs.wisc.edu/~remzi/OSTEP/

Process Management." GeeksforGeeks, December 6th, 2023, https://www.geeksforgeeks.org/introduction-of-process-management/

Sabela, Ramos, et al. "Cache Line Aware Algorithm Design for Cache-Coherent Architectures."

Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating system concepts* (10th ed.). Wiley. https://www.wiley.com/en-us/Operating+System+Concepts%2C+10th+Edition-p-9781119320913

Stephen, Kellett. "Memory Fragmentation, your worst nightmare." Softwareverify, May 11th, 2021, https://www.softwareverify.com/blog/memory-fragmentation-your-worst-nightmare/


Program
Instruction #1
Instruction #2
Instruction #3
Instruction #4
...
Instruction #N
Includes
Program
Instantiate
RAM
Process
Abstraction
RAM
Process #1
Process #2
...
Process #M
Table of thread pointers and their priorities
Thread #1 (high)
Thread #2 (low)
Thread #3 (med)
Thread #4 (low)
...
Thread #K (high)
Context Switch
Preemption
Collect pointers
CPU
Execute
Scheduler
(Part of OS)

Indicative contents
3.1 Implementation of process and threads
3.2 Implementation of process scheduling algorithms
3.3 Applying Parallelism
3.4 Applying Lock-Based Concurrency Control

Key Competencies for Learning Outcome 3: Implementing the Process Management

Knowledge	Skills	Attitudes
• Description of layout of process address space • Description of process state and scheduling algorithm • Description of scheduling criteria • Description of Lock-Based Concurrent Control	• Applying process states • Applying thread operations • Implementing process control blocks • Implementing multi-threading and synchronization mechanics • Developing of scheduling algorithms • Applying Parallelism • Applying Lock-Based Concurrency Control	• Being attentive • Having Adaptability • Being a Team work • Having Creativity

Duration: 25hrs

Learning outcome 3 objectives:

By the end of the learning outcome, the trainees will be able to:

1. Describe correctly layout of process address space in process management
2. Describe process state and scheduling algorithm in process management
3. Describe scheduling criteria in process management
4. Describe Lock-Based Concurrent Control in process management
5. Implement correctly the process and threads in process management
6. Implement correctly the process scheduling algorithm in process management
7. Apply properly a parallelism based on the intended performance and efficiency in process management
8. Apply properly Lock-Based Concurrency Control in process management

Resources

Equipment	Tools	Materials
• Computer • Projector	• VS Code • MSYS2 • GCC • QEMU	• Internet • Electricity

Advance Preparation:

Before delivering this learning outcome, you are recommended to:

- Avail QEMU, VMware workstation, text editor setup.
- Avail files or project related to operating system development

Indicative content 3.1: Implementation of Process and Threads.

Duration: 10hrs

Theoretical Activity 3.1.1: Description of key terms.

Notes to the trainer:

- While delivering this content, a small group can be used for describing Hobby kernel.

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the activity and ask trainees to describe the following hobby kernel terms:

I. Process
II. Thread
III. Multi-thread
IV. Process control block
V. State process model
VI. No preemptive
VII. Preemptive
VIII. Cooperating process(inter-process)
IX. User level threads
X. Kernel level thread
XI. Deadlock
XII. Livelock

Step 2: Ask learner to write answers provided on flip-chart/paper or blackboard.

Step 3: Asks Trainees to discuss the provided answer and choose correct answers.

Step 4: Provides expert view and clarifies ideas.

Step 5: Address any questions or concerns.

Step 6: Ask trainees to read the key reading 3.1.1 in the trainee manual.

Points to Remember

1. Process & Thread:

- **Process**: A program in execution with its own memory/resources.
- **Thread**: A lightweight execution unit within a process.
- **Multi-threading**: Running multiple threads for efficiency.

2. Process Control Block (PCB): Stores all process-related data (PID, state, scheduling info).

3. Process States:

- New, Ready, Running, Waiting, Terminated.
- State transitions based on scheduling and events like I/O.

4. **Scheduling**:

- **Non-preemptive:** Process runs to completion without interruption.
- **Preemptive**: OS can interrupt/switch processes for responsiveness.

5. Deadlock Processes are stuck waiting on each other while **Livelock** Processes change states but make no progress.

6. Process Synchronization: Locks, Semaphores, Monitors ensure safe access to shared resources.

7. IPC (Inter-Process Communication): Methods include pipes, shared memory, message queues, and signals for process interaction.

8. Thread Operations: Creation, termination, synchronization, scheduling, and context switching.

9. Deadlock Handling: Prevention, avoidance, detection, and recovery strategies.

10. Livelock Handling: Backoff algorithms and randomization to prevent constant retries.

Theoretical Activity 3.1.2: Explanation of processes interpretation

Notes to the trainer:

- While delivering this content, a small group can be used for explain processes in hobby kernel.

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the activity and ask trainees to answer the following questions:

I. Explain process life cycle in operating system
II. Design the process three in operating system

Step 2: Ask learner to write answers provided on flip-chart/paper or blackboard.

Step 3: Asks Trainees to discuss the provided answer and choose correct answers.

Step 4: Provides expert view and clarifies ideas.

Step 5: Address any questions or concerns.

Step 6: Ask trainees to read the key reading 3.1.2 in the trainee manual.

Points to Remember

1. Process States and State Diagram:

- **New**: Process is being created.
- **Ready**: Process is loaded into memory, waiting for CPU allocation.
- **Running**: Process is actively executing on the CPU.

- **Blocked (Waiting)**: Process is paused, waiting for an event (e.g., I/O completion).
- **Terminated**: Process has finished execution.

State Transitions:

- **New → Ready**: Process creation is complete, and it's ready for scheduling.
- **Ready → Running**: Scheduler assigns CPU, and the process starts executing.
- **Running → Blocked**: Process is waiting for an event (e.g., I/O).
- **Blocked → Ready**: Event is complete; the process is ready to run again.
- **Running → Terminated**: Process execution is completed.

2. Process Trees:

- **Process Hierarchy**: Processes are organized in a tree structure where a parent creates child processes.
- **Parent & Child Processes**: Each process can have child processes, and all processes originate from an initial parent

Key Components in Kernel Development:

- **Process Control Block (PCB)**: Stores process metadata (PID, state, scheduling info, etc.).
- **Scheduling**: Kernel scheduler determines which process runs next based on algorithms (e.g., priority).
- **Context Switching**: When switching between processes, the kernel saves the current process state and restores the next scheduled process's state.

Theoretical Activity 3.1.3: Description of process hierarchy important.

Notes to the trainer:

- While delivering this content, a small group can be used to identifying importance of process hierarchy.

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the activity and ask Trainees to describe the following process hierarchy

I. Foreground process
II. Visible process
III. Service process
IV. Background process
V. Empty process

Step 2: Ask learner to write answers provided on flip-chart/paper or blackboard.

Step 3: Asks Trainees to discuss the provided answer and choose correct answers.

Step 4: Provides expert view and clarifies ideas.

Step 5: Address any questions or concerns.

Step 6: Ask trainees to read the key reading 3.1.3 in the trainee manual.

Points to Remember

1. **Foreground Process**: Interacts directly with the user, running in an active session (e.g., terminal commands).
2. **Visible Process**: Appears in the GUI, interacting with users via visual elements (e.g., web browser, file manager).
3. **Service Process**: Background system task with no user interaction, often starting automatically (e.g., sshd, database services).
4. **Background Process**: Runs without user input, handling tasks like downloads or updates while other activities continue.
5. **Empty Process**: Refers to a zombie process (finished but not yet collected by its parent) or an idle process (runs when CPU is idle).

Practical Activity 3.1.4: Applying process States

Notes to the trainer

- The trainer is required to explain operating system process states
- Avail of computer

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the topic and ask trainees to read the following tasks:

As CSA trainee, you are requested to go to the computer lab to write a C program code that will:

- **I.** Fork New process
- **II.** Simulate Running process
- **III.** Simulate Waiting process
- **IV.** Simulate Ready process

Step 2: Provide instruction that will be followed

Step 3: Demonstrate how to apply process states, while demonstrating, explain the steps to follow.

Step 4: Verify whether process states are clearly applied

Step 5: Ask trainees to read key reading 3.1.4. in the Trainee's Manual

Points to Remember

1. Fork New Process: Forking a new process involves creating a child process from a parent process.
2. Simulate Running Process: To simulate a running process, you need a scheduler that selects a process from the ready queue and changes its state to running.
3. Simulate Waiting Process: A process can enter the waiting state when it needs to wait for an event (e.g., I/O completion).
4. Simulate Ready Process: A process in the waiting state can be moved to the ready state once the event it was waiting for occurs.
5. Simulate Terminated Process: When a process finishes execution, it enters the terminated state.
6. Creation of a Thread into a Process: Creating a thread within a process involves allocating a new thread context and associating it with the parent process.

Practical Activity 3.1.5: Applying Threads operations

Notes to the trainer

- While delivering this content, a small group can be used for introducing Thread and its operations.
- Avail of computer.

Key steps:

While delivering this activity, pass through the following steps:

Step 1:	Introduce the activity and ask Trainees to perform the following task: As software developer, you are requested to go to the computer lab and write C program to applying Thread into process?
Step 2:	Provide instruction that will be followed
Step 3:	Demonstrate how to apply threads operations, while demonstrating explain steps to follow.
Step 4:	Asks Trainees to discuss the provided answer and monitor the procedures.
Step 5:	Provides expert view and clarifies ideas.
Step 6:	Address any questions or concerns.
Step 7:	Ask trainees to read the key reading 3.1.5 in the trainee manual.

Points to Remember

1. Thread Operations

- **Thread Creation:** Creating a new thread within a process.
- **Thread Termination:** Terminating a thread when it has completed its task.
- **Thread Synchronization:** Coordinating the execution of multiple threads to ensure correct operation.
- **Thread Scheduling:** Determining which thread runs at any given time.
- **Thread Context Switching:** Saving the state of a currently running thread and restoring the state of another thread.
- **Thread Communication**: Enabling threads to communicate with each other, often through shared memory or message passing.

Practical Activity 3.1.6 Applying process of control block (PCB) to manage process

Notes to the trainer

- While delivering this content, a small group can be used for explain Process of Control block (PCB) and its components.
- Avail of computer.

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the activity and ask Trainees to perform the following task:
As software developer, you are requested to go to the computer and write C program to applying Process Control Block?

Step 2: Provide instruction that will be followed

Step 3: Demonstrate how to apply process control block, while demonstrating explain steps to follow.

Step 4: Asks Trainees to discuss the provided answer and monitor the procedures.

Step 5: Provides expert view and clarifies ideas.

Step 6: Address any questions or concerns.

Points to Remember

1. Definition:

Process Control Block (PCB) is a data structure used by the operating system to store all the information about a process.

2. Key Components of a PCB

- **Process State:** Indicates the current state of the process (e.g., running, waiting, ready, terminated).
- **Process ID (PID):** A unique identifier assigned to each process.
- **Program Counter**: Stores the address of the next instruction to be executed.
- **CPU Registers:** Holds the values of the CPU registers for the process.
- **Memory Management Information:** Includes details like base and limit registers.
- **Accounting Information:** Tracks CPU usage, memory usage, etc.
- **I/O Status Information**: Information about I/O devices allocated to the process.
- **Process Scheduling Information**: Includes process priority and scheduling algorithms.

Practical Activity 3.1.7: Applying state management mechanism

Notes to the trainer

- While delivering this content, a small group can be used for introducing Thread and its operations.
- Avail of computer.

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the activity and ask Trainees to perform the following task:
As software developer, you are requested to go to the computer lab to write C program that perform context switching and process synchronization?

Step 2: Demonstrate how to apply context switching and process synchronization, while demonstrating explain steps to follow.

Step 3: Asks Trainees to discuss the provided answer and monitor the procedures.

Step 4: Provides expert view and clarifies ideas.

Step 5: Address any questions or concerns.

Step 6: Ask trainees tor read the key reading 3.1.7 in the trainee manual

Points to Remember

1. Context Switching: is a fundamental aspect of state management in operating systems. It involves saving the state of a currently running process and loading the state of the next process to be executed.

2. Process Synchronization: ensures that multiple processes can operate concurrently without interfering with each other. Includes:

I. **Locks:** prevents multiple processes from accessing a shared resource at the same time.
II. **Semaphores:** is a counter that controls access to shared resources, allowing a limited number of processes to access the resource concurrently.
III. **Monitors:** is a high-level synchronization construct that combines mutual exclusion (like a lock) and condition variables to allow safe access to shared resources.

Practical Activity 3.1.8: Applying Inter-process mechanism

Notes to the trainer

- While delivering this content, a small group can be used for explain mechanism of inter-process.
- Avail of computer.

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the activity and ask Trainees to perform the following task:
As software developer, you are requested to go to the computer lab to write C program that perform the following inter-process mechanism?
I. Pipes
II. Shared memory
III. Message queue
IV. Signals

Step 2: Demonstrate how to apply inter-process mechanism, while demonstrating explain steps to follow.

Step 3: Asks Trainees to discuss the provided answer and monitor the procedures.

Step 4: Provides expert view and clarifies ideas.

Step 5: Address any questions or concerns.

Step 6: Ask trainees tor read the key reading 3.1.8 in the trainee manual

Points to Remember

1. **Pipes**: Unidirectional communication mechanism where one process writes and another reads.

Example: A child writes to a pipe, and a parent reads the data.

2. **Shared Memory:** A faster communication method where processes share a common memory space.

Example: A child writes to shared memory, and the parent reads from it.

3. **Message Queue**: Allows processes to exchange messages through a queue, supporting structured message communication.

Example: A child sends a message to the queue, and the parent receives it.

4. **Signals**: Software interrupts used to notify processes of events, allowing them to handle those events asynchronously.

Example: A child process sends a signal to the parent, and the parent handles the signal using a handler function.

Theoretical Activity 3.1.9: Explanation of Algorithm to handle deadlocks and lovelocks

Notes to the trainer:

- While delivering this content, a small group can be used for explain deadlocks and livelocks.
- Avail of computer.

Key steps:

Step 1: Introduce the activity and ask trainees to answer the following questions:

I. Write an algorithm to handle deadlocks

II. Write an algorithm to handle livelocks

Step 2: Ask trainees to present their findings to the whole class

Step 3: Provides expert view and clarifies ideas by using didactic materials

Step 4: Ask trainees to read key readings in activity 3.1.9 in Trainee 's manual

Points to Remember

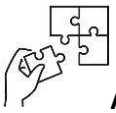
1. **Deadlocks** occur when processes block each other indefinitely while waiting for resources. To handle deadlocks:

- **Prevention**: Ensure one of the following conditions doesn't hold: mutual exclusion, hold and wait, no preemption, circular wait.
- **Avoidance**: Use algorithms like Banker's Algorithm, which grants resources only if the system remains in a safe state.
- **Detection and Recovery**: Detect deadlocks by building a wait-for graph and checking for cycles. Recover by terminating processes or rolling them back to a safe state.

2. Livelocks happen when processes keep changing states without making progress. To handle livelocks:

- Backoff Algorithms: Processes wait for random or increasing periods before retrying, preventing constant retries.

- Randomization: Introduce random behavior in process scheduling to avoid repeated conflicts.



Application of learning 3.1.

Scenario

As CSA Trainee you requested to write C program that incorporates concepts related to process states, thread operations, process control blocks (PCBs), state management mechanisms, inter-process communication.

Checklist

S/N	Assessment criteria (Based on performance criteria)	Indicator/element to check	Observation	
			yes	No
1	**Implementation of processes and threads**	Process state is applied		
		Threads operation is used		
		Process control block (PCB) is managing process		
		State management mechanism is applied		
		Inter-process mechanism is applied		

Indicative content 3.2: Implementation of Process Scheduling Algorithms

Duration: 8hrs

Theoretical Activity 3.2.1: Description of key terms.

Notes to the trainer:

- While delivering this content, a small group can be used for describing Hobby kernel.

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the activity and ask Trainees to describe the following hobby kernel terms:

1. Process scheduling
2. Preemptive
3. Non preemptive
4. CPU Burst
5. I/O Burst
6. CPU Scheduler
7. Dispatcher modules
8. Dispatcher latency

Step 2: Ask learner to write answers provided on flip-chart/paper or blackboard.

Step 3: Asks Trainees to discuss the provided answer and choose correct answers.

Step 4: Provides expert view and clarifies ideas.

Step 5: Address any questions or concerns.

Step 6: Ask trainees to read the key reading 3.2.1 in the trainee manual.

Points to Remember

1. **Process Scheduling:** Method for selecting which process runs to ensure effective CPU sharing.
2. **Preemptive Scheduling** OS can interrupt processes (e.g., Round Robin, SRTF, Priority Scheduling) While **Non-Preemptive Scheduling** Process runs to completion (e.g., FCFS, SJF)
3. **CPU Burst** Period where the process uses the CPU while **I/O Burst** Process waits for I/O, allowing the CPU to switch to another process.
4. **CPU Scheduler**: Selects the next process from the ready queue.
5. **Dispatcher**: Handles process switching (context switching, user mode switching).

Theoretical Activity 3.2.2: Description of Scheduling criteria

Notes to the trainer:

- While delivering this content, a small group can be used for describing Scheduling criteria.

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the activity and ask Trainees to describe the following hobby kernel terms:

1. CPU utilization
2. Throughput
3. Turnaround time
4. Waiting time
5. Response time

Step 2: Ask learner to write answers provided on flip-chart/paper or blackboard.

Step 3: Asks Trainees to discuss the provided answer and choose correct answers.

Step 4: Provides expert view and clarifies ideas.

Step 5: Address any questions or concerns.

Step 6: Ask trainees to read the key reading 3.2.2 in the trainee manual.

Points to Remember

1. **CPU Utilization**: The percentage of time the CPU is actively executing processes, with the aim to maximize it, ideally nearing 100%, for optimal system performance.
2. **Throughput**: The number of processes completed in a given time period. Higher throughput signifies better efficiency, especially important in high-demand environments.
3. **Turnaround Time:** The total time from process submission to its completion, including waiting and execution time. Minimizing this ensures quicker completion of processes.
4. **Waiting Time**: The time a process spends in the ready queue waiting for CPU access. Reducing this improves system responsiveness and avoids unnecessary delays.
5. **Response Time**: The time between submitting a process and its first execution. Lower response times are essential for interactive and real-time systems to ensure quick user feedback and a better overall experience.

Theoretical Activity 3.2.3: Description of Scheduling algorithm optimization criteria

Notes to the trainer:

- While delivering this content, a small group can be used for describing Scheduling algorithm optimization criteria.

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the activity and ask Trainees to describe scheduling optimization algorithm

1. Max CPU utilization
2. Max throughput
3. Min turnaround time
4. Min waiting time

Step 2: Ask learner to write answers provided on flip-chart/paper or blackboard.

Step 3: Asks Trainees to discuss the provided answer and choose correct answers.

Step 4: Provides expert view and clarifies ideas.

Step 5: Address any questions or concerns.

Step 6: Ask trainees to read the key reading 3.2.3 in the trainee manual.

Points to Remember

1. **Max CPU Utilization** ensures that the CPU is kept as busy as possible, reducing idle time and maximizing system efficiency by maintaining near-continuous operation of the CPU.
2. **Max Throughput** focuses on completing the maximum number of processes in a given time period, enhancing system productivity by increasing the rate at which tasks are finished, especially in high-load environments.
3. **Min Turnaround Time** aims to reduce the total time a process takes from submission to completion, ensuring that processes finish quickly, which boosts overall system performance and user satisfaction.
4. **Min Waiting Time** minimizes the time a process spends waiting in the ready queue before being executed, improving system responsiveness and ensuring faster CPU access for processes, leading to better resource utilization.

Practical Activity 3.2.4: Applying scheduling algorithms

Notes to the trainer

- Avail of computer.

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the topic and ask trainees to read the following tasks:

As software developer, you are requested to go to the computer lab and write C program to apply scheduling algorithm.

Step 2: Provide instruction that will be followed

Step 3: Demonstrate how to apply page Replacement algorithm, while demonstrating, explain the steps to follow.

Step 4: Asks Trainees to apply scheduling algorithm and monitor the procedures.

Step 5: Verify whether scheduling algorithm are clearly applied

Step 6: Ask trainees to read key reading 3.2.4. in the Trainee's Manual

Points to Remember

1. Pre-emptive Scheduling

- **Round Robin (RR):** Each process is assigned a fixed time slice (quantum). If a process doesn't finish within its time slice, it's moved to the end of the queue.
- **Shortest Remaining Time First (SRTF):** The process with the shortest remaining execution time is selected next.
- **Priority Scheduling:** Each process is assigned a priority. The process with the highest priority is selected next.

2. Non-Pre-emptive Scheduling

- **First Come First Serve (FCFS)**: Processes are executed in the order they arrive.
- **Shortest Job First (SJF)**: The process with the shortest execution time is selected next.

Practical Activity 3.2.5: Applying scheduling algorithm evaluation

Notes to the trainer

- Avail of computer

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the topic and ask trainees to go to the computer lab and Write C program to calculate **average waiting time in the queue (Wq) and number of processes in the queue (Lq) if** λ (arrival rate) = 2 processes/second and μ (service rate) = 5 processes/second

Step 2: Provide instruction that will be followed

Step 3: Demonstrate how to apply page scheduling algorithm evolution, while demonstrating, explain the steps to follow.

Step 4: Asks Trainees to apply scheduling algorithm evolution and monitor the procedures.

Step 5: Verify whether scheduling algorithm evolution are clearly applied

Step 6: Ask trainees to read key reading 3.2.5. in the Trainee's Manual

Points to Remember

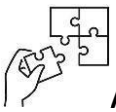
1. **Deterministic Evaluation**: Evaluates algorithm performance using predefined workloads.
2. **Queuing Model**: Models processes in terms of arrival and service rates to determine queue length and waiting time.
3. **Little's Formula**: Relates system load with average time in the system to evaluate performance stability.

- **Turnaround Time (TAT)**: The total time taken from submission of a process to its completion.
- **Waiting Time (WT)**: The total time a process waits in the ready queue.
- **Throughput**: The number of processes completed per unit of time.
  - We calculate the **Turnaround Time** and **Waiting Time** for each process.
  - We use the First-Come-First-Serve (FCFS) scheduling algorithm.
  - **Average waiting time in the queue (Wq)**:

$$\mathrm{Wq} = \frac{\lambda}{\mu(\mu - \lambda)}$$

  - **Average number of processes in the queue (Lq)**:

$$Lq = \frac{\lambda}{}$$



Application of learning 3.2.

As a trainee in CSA go to the computer lab and Write C program to calculate **average waiting time in the queue (Wq) and number of processes in the queue (Lq) if** λ (arrival rate) = 2 processes/second and μ (service rate) = 5 processes/second

S/N	Assessment criteria (Based on performance criteria)	Indicator/element to check	Observation	
			yes	No
1	C program to calculate average waiting time in the queue (Wq) and number of processes in the queue is written	Correctness of Logic is done		
		Input Handling is applied		
		Efficiency and Performance is checked		
		Code Readability and Maintainability is applied		
		Robustness and Testing is done		

Indicative content 3.3: Applying Parallelism

Duration: 3 hrs

Theoretical Activity 3.3.1: Description of parallelism in Kernel development

Notes to the trainer:

- While delivering this content, a small group can be used for describing parallelism, it's types and parallelism roles.

Key steps:

While delivering this activity, pass through the following steps:

Step1: Introduce the topic and ask trainees to answer the following tasks related to parallelism in operating system:

1. Explain parallelism
2. Outline parallelism types
3. What are roles of parallelism

Step2: Ask trainees to present their findings to the whole class.

Step3: Provides expert view and clarifies ideas by using didactic materials.

Step4: Address any questions or concerns

Step5: Ask trainees to read key reading 3.3.1 in trainee manual

Points to Remember

- **Types of Parallelism** are Task Parallelism**,** Data Parallelism, Instruction-Level Parallelism (ILP), Thread-Level Parallelism (TLP)
- **Roles of Parallelism** are Improves Performance, Maximizes Resource Utilization and Enhances Scalability

Practical Activity 3.3.2: Implementing parallelism techniques

Notes to the trainer

- Avail of computer

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the activity and ask Trainees to explain
n the following terms:

As software developer, you are requested to go to the computer lab to write C program to implementing multi-threading, multi-processing and SIMD instructions?

Step 2 Provide instruction that will be followed

Step 3 Demonstrate how to apply parallelism techniques, while demonstrating, explain the steps to follow.

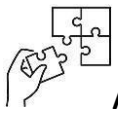
Step 4 Asks trainees to apply parallelism techniques and monitor the procedures.

Step 5 Verify whether parallelism techniques are clearly applied

Step 6 Ask trainees to read key reading 3.3.2 in trainee manual

Points to Remember

- **Multithreading**: Executes multiple threads within the same process concurrently using shared memory.
- **Multiprocessing**: Runs multiple processes in parallel, with each having its own memory space.
- **SIMD Instructions**: Enables parallelism by applying the same operation to multiple data points simultaneously.



Application of learning 3.3.

Scenario

You are developing a high-performance computing application that processes large datasets. To enhance performance, your team decides to implement various parallelism techniques, including multithreading, multiprocessing, and SIMD (Single Instruction, Multiple Data) instructions.

Task:

Develop a C program that demonstrates the use of parallelism techniques through the following:

1. Implement multithreading to perform matrix multiplication.
2. Use multiprocessing to calculate the sum of elements in large arrays.
3. Utilize SIMD instructions for vector addition to enhance performance.

Checklist

S/N	Assessment criteria (Based on performance criteria)	Indicator/element to check	Observation	
			yes	No
1	**Parallelism techniques is well used**	Multithreading is implemented		
		Multiprocessing is used		
		SIMD instruction is utilized		

Indicative content 3.4: Applying Lock-Based Concurrency Control

Duration: 4 hrs

Theoretical Activity 3.4.1: Description of Lock-Based Concurrency Control

Notes to the trainer:

- Avail of computer

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the activity and ask Trainees to answer the following questions related to Lock-based concurrency:

I. What is Lock-Based Concurrency Control?
II. Explain the difference between a shared lock and an exclusive lock. Provide an example of when each might be used.
III. List and briefly describe two alternatives to Lock-BCC, including one potential advantage of each.

Step 2 Explain the task and provide clear instruction.

Step 3 Ask trainees to present their findings to the whole class.

Step 4 Provides expert view and clarifies ideas by using didactic materials.

Step 5 Address any questions or concerns.

Step 6 Ask trainees to read key reading 3.4.1 in trainee manual.

Points to Remember

- **Lock-BCC** manages how multiple processes or transactions access shared data concurrently, ensuring orderly access to avoid conflicts.
- **Data Items**: Includes shared resources like variables or database records.
- **Transactions**: Series of atomic operations requiring access control.
- **Locks** are Shared Lock (S)**,** Exclusive Lock (X)
- **Benefits**: Ensures data integrity, isolation, and controlled access.
- **Drawbacks**: Can cause deadlocks, blocking, and performance overhead.
- **Alternatives**: Optimistic Concurrency Control (OCC), Timestamp Ordering, and Multiversion Concurrency Control (MVCC) offer different approaches to concurrency management

Practical Activity 3.4.2: Implementing Lock_Based Concurrency Control Protocol

Notes to the trainer

- Avail of computer

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the activity and ask Trainees to answer the following questions related to the Applying Lock-Based Concurrency Control:

As software developer, you are requested to go to the computer lab and write C Program that will applying read lock, write lock, awaiting conflict lock release and release locks.

Step 2 Provide instruction that will be followed

Step 3 Demonstrate how to apply Lock-Base Concurrency Control, while demonstrating, explain the steps to be followed

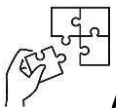
Step 4 Asks trainees to apply Lock-Base Concurrency Control and monitor the procedures

Step 5 Verify whether Lock-Base Concurrency Control are clearly applied.

Step 6 Ask trainees to read key reading 3.4.2 in trainee manual.

Points to Remember

- **Lock Variables:**
  - ✓ **pthread_mutex_t lock:** Controls access to shared resources.
  - ✓ **pthread_cond_t cond:** Condition variable allowing threads to wait for a lock if there's a conflict.
  - ✓ **read_count:** Keeps track of how many threads hold the read lock.
  - ✓ **write_lock**: A flag indicating if a thread holds the write lock.
- **Applying a Read Lock**: Threads check for a write lock. If one exists, they wait using **pthread_cond_wait()**. If not, they proceed.
- **Applying a Write Lock**: Threads wait until no other readers or writers hold locks before proceeding.
- **Awaiting Lock Release:** Threads use **pthread_cond_wait()** to wait for locks (read or write) to be released to avoid conflicts.
- **Releasing Locks:** When a thread completes its operation, it releases the lock and signals waiting threads with **pthread_cond_signal()** or **pthread_cond_broadcast()** to allow others to acquire the lock.



Application of learning 3.4

You are developing a simple banking application that demonstrates lock-based concurrency control by using a C program. The application allows multiple threads to read and write to a shared bank account balance safely using locks.

Checklist

S/N	Assessment criteria (Based on performance criteria)	Indicator/element to check	Observation	
			yes	No
1	**Implementing Lock-based Concurrency Control Protocol**	Lock to read and write is applied		
		Lock conflict is released		
		Release lock transaction is completed		

Learning outcome 3 end assessment

Written assessment

I. Multiple choice question, choose the right answer and circle the corresponding letter

1.Which of the following is NOT a state in the process lifecycle?

a. New
b. Running
c. Scheduled
d. Blocked

Answer: c) Scheduled

2.What happens to a process when it enters the "Blocked" state?

a. It waits for CPU time.
b. It waits for an event or I/O operation to complete.
c. It finishes its execution.
d. It terminates.

Answer: b) It waits for an event or I/O operation to complete.

3.In the context of process management, what does the Process Control Block (PCB) NOT store?

a. Process ID (PID)
b. Process state
c. Process owner's name
d. Program counter

Answer: c) Process owner's name

4.Which process synchronization technique is used to prevent race conditions in shared resources?

a. Thread Creation
b. Context Switching
c. Mutex Locks
d. Thread Scheduling

Answer: c) Mutex Locks

5. What is the primary role of a kernel scheduler?

a. Managing the GUI
b. Selecting which process gets CPU time
c. Monitoring file system usage
d. Creating new processes

Answer: b) Selecting which process gets CPU time

II. Answer True or False on the following questions

a) A process moves from the "Running" state to the "Ready" state when it completes its execution? **False**

b) The Process Control Block (PCB) stores information about the process's memory, CPU registers, and scheduling? **True**
c) Foreground processes are always service processes? **False**
d) The system scheduler decides which process in the "Blocked" state gets CPU time next? **False**
e) Context switching is required when a process moves from "Ready" to "Running"? **True**

III. Choice the correct answer by ticking

1. Which of the following best describes thread-level parallelism (TLP)?

a. Executing different operations concurrently across different threads
b. Executing the same operation on different data simultaneously
c. Executing multiple instructions from a single stream concurrently
d. Using SIMD instructions for vector processing

Answer: a) Executing different operations concurrently across different threads

2. Which parallelism technique is ideal for CPU-bound tasks that can fully utilize multiple cores?

a. Multithreading
b. SIMD
c. Multiprocessing
d. Instruction-level parallelism

Answer: c) Multiprocessing

IV. Match each type of parallelism with its appropriate description:

Answer	Parallelism type	Description
1-d	1. Multithreading	a) Executes different operations on multiple data points
2-c	2. Multiprocessing	b) Executes multiple instructions from the same stream
3-a	3. SIMD	c) Runs multiple processes with independent memory spaces
4-b	4. Instruction-Level	d) Runs multiple threads in a shared memory space
		e) Applies the same operation to different pieces of data simultaneously

V. Answer True/False on the following Questions

a. In multithreading, each thread has its own independent memory space, separate from other threads. **False**
b. Task parallelism applies the same operation to different pieces of data simultaneously. **False**
c. The fork() function is used to create new threads within the same process. False

d. SIMD is effective for tasks like vector processing and matrix multiplication. **True**

VI. Choose the correct answer

1. Which of the following problems is commonly associated with Lock-Based Concurrency Control?

a. Non-atomic transactions
b. Deadlock
c. High concurrency
d. Data loss

Answer: b) Deadlock

2. In Lock-BCC, what type of lock would allow multiple readers but prevent any writers from accessing a data item concurrently?

a. Exclusive Lock
b. Mutex Lock
c. Shared Lock
d. Read-Write Lock

Answer: c) Shared Lock

3. What is the role of a condition variable in lock-based concurrency control, as demonstrated in the provided C code example?

a. To increment the number of readers
b. To signal waiting threads when a lock is released
c. To manage deadlock scenarios
d. To track the state of the write lock

Answer: b) To signal waiting threads when a lock is released

VIII. Match each term related to lock-based concurrency control with its correct description:

Answer	Term	Description
1-d	1. Deadlock	a) Orders transactions based on their starting time
2-c	2. Exclusive Lock	b) Permits multiple readers, but prevents writers during concurrent access
3-b	3. Shared Lock	c) Allows only one transaction to read/write at a time
4-a	4. Timestamp Ordering	d) System standstill where transactions wait for each other to release locks
		e) Allows reads without locking by maintaining multiple versions of data)

IX.Answer True/False on the following Questions

a) An exclusive lock allows multiple transactions to write to a data item concurrently. **False**
b) Multiversion Concurrency Control (MVCC) requires locking mechanisms to maintain high concurrency. **False**
c) Using condition variables, threads can wait until a conflicting lock is released. **True**
d) Optimistic Concurrency Control always avoids deadlock by not using locks during transaction processing. **True**

Practical assessment

Scenario

As trainee in Computer System and Architecture, you are a part of a software development team at your TVET School tasked with creating a simplified process management system. This system is intended to help students understand how operating systems manage processes and threads, as well as how to implement scheduling algorithms and concurrency control.
Develop a C program simulating a simplified process management system to understand operating system concepts.

Requirements:

1. **Process States:**
   - Implement states: **New**, **Ready**, **Running**, **Waiting**, and **Terminated**.
   - Enable state transitions based on simulated events.
2. **Thread Operations:** Use threads to represent processes, performing operations like incrementing a counter.
3. **Process Control Block (PCB):** Create a structure for PCB with attributes like process ID, state, priority, and execution time.
4. **Process Scheduling:** Implement:
   - **Preemptive Scheduling:** Round Robin.
   - **Non-Preemptive Scheduling:** First-Come, First-Served (FCFS).
5. **Concurrency Control:** Use mutexes for lock-based concurrency to manage shared resources between threads.

Checklist

S/N	Assessment criteria (Based on performance criteria)	Indicator/element to check	Observation	
			yes	No
1	**Implementation of process management is well done**	Process state is used		
		Thread operation is applied		
		Process control block managed process		
		Scheduling algorithm is used		
		Parallelism techniques is used		
		Lock-based Concurrency Control Protocol is used		

Further information to the trainer

Bovet, D. P., & Cesati, M. (2005). *Understanding the Linux kernel* (3rd ed.). O'Reilly Media. https://www.cs.utexas.edu/~rossbach/cs380p/papers/ulk3.pdf.

Herlihy, M., & Shavit, N. (2012). *The art of multiprocessor programming* (Revised ed.). Morgan Kaufmann. https://www.elsevier.com/books/the-art-of-multiprocessor-programming/herlihy/978-0-12-415950-1.

lighthousand. (2014). *books/Modern Operating Systems 4th Edition--Andrew Tanenbaum.pdf at master · lighthousand/books*. GitHub. https://github.com/lighthousand/books/blob/master/Modern%20Operating%20Systems%204th%20Edition--Andrew%20Tanenbaum.pdf

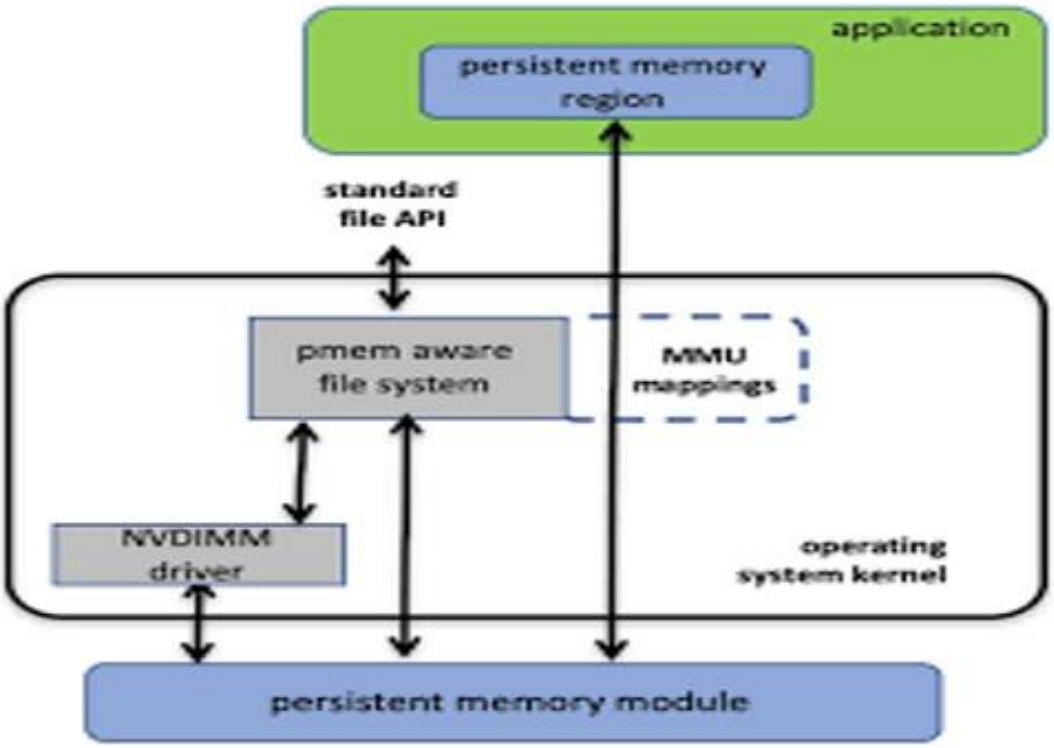
Operating Systems: Three Easy Pieces. (n.d.). Pages.cs.wisc.edu. https://pages.cs.wisc.edu/~remzi/OSTEP/

Process Management." GeeksforGeeks, December 6th, 2023, https://www.geeksforgeeks.org/introduction-of-process-management/

Sabela, Ramos, et al. "Cache Line Aware Algorithm Design for Cache-Coherent Architectures." IEEE Xplore, https://ieeexplore.ieee.org/abstract/document/7378320

Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating system concepts* (10th ed.). Wiley. https://www.wiley.com/en-us/Operating+System+Concepts%2C+10th+Edition-p-9781119320913

Stephen, Kellett. "Memory Fragmentation, your worst nightmare." Softwareverify, May 11th, 2021, https://www.softwareverify.com/blog/memory-fragmentation-your-worst-nightmare/


application
persistent memory region
standard file API
pmem aware file system
MMU mappings
NVDIMM driver
operating system kernel
persistent memory module

Indicative contents
4.1 Introduction to persistence management
4.2 Management of Mass storage structure
4.3 Applying I/O systems
4.4 Implementing file systems

Key Competencies for Learning Outcome 4: Implement Persistence Management

Knowledge	Skills	Attitudes
• Description of key terms • Identification mass storage structure. • Identification of data persistence Mechanism • Description of I/O systems • Identification of I/O devices, Device Drivers and I/O models • Description of file system • Identification of file system selection criteria	• Developing of scheduling algorithms. • Implementing file system • Implementing I/O systems. • Implementing file allocation methods and file system management operations. • Implementing file and directory operations	• Respecting time management and being organized • Having curiosity • Having Passion • Being creative • Being patience • Having integrity

Duration: 20hrs

Learning outcome 4 objectives:

By the end of the learning outcome, the trainees will be able to:

1. Describe correctly key terms as they are used in hobby kernel persistence management
2. Identify properly Data persistence Mechanism used in hobby kernel persistence management
3. Describe properly mass storage physical structure as they are required in hobby kernel development
4. Apply correctly free space management by writing appropriate program in C
5. Apply correctly disk scheduling algorithm by selecting the best fit for your project
6. Describe properly I/O system as they required in hobby kernel development
7. Identify correctly I/O devices, Device Drivers and I/O models as used in I/O system for hobby kernel development
8. Apply properly Interrupts handling, I/O scheduling algorithm and device management for I/O systems used through persistence management
9. Describe correctly file system as used in hobby kernel development
10. Identify correctly file system selection criteria as used in hobby kernel development
11. Apply correctly file and directory operations as used in hobby kernel development
12. Use correctly file allocation methods and file system management operations as they are used in hobby kernel development

Resources

Equipment	Tools	Materials
• Computer • Projector	• QEMU • MSYS2 • VSCode • NASM • SASM • MinGw • GCC	• Internet • Electricity

Advance Preparation:

Before delivering this learning outcome, you are recommended to:

- Avail computer lab
- Avail software tools
- Avail prewritten codes or algorithms (Materials)
- Avail video tutorials

Indicative content 4.1: Introduction to Persistence Management

Duration: 5 hrs

Theoretical Activity 4.1.1: Description of key terms and identification of data Mechanism

Notes to the trainer:

- While delivering this content, a small group can be used for describing key term and identification of data persistence management.

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the activity and ask trainees to answer the following questions:

i) What do you understand about the following key terms used in data persistence management?

- **a)** System calls
- **b)** Data Buffers
- **c)** Caching
- **d)** Disk flushing
- **e)** Journaling

ii) Identify data persistence mechanism

Step 2: Asks trainees to write answers provided on flip-chart/paper.

Step 3: Asks trainees to discuss and agree from their provided answers and choose correct answers.

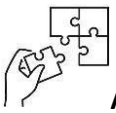
Step 4: Provides expert view and addresses any questions if any.

Step 5: For more clarification ask trainees to read the key reading 4.1.1 in the trainee's manual.

Points to Remember

- **System calls** are mechanisms that allow user-level processes to communicate with the kernel of the operating system.
- **Data buffers** are temporary storage areas used to hold data while it is being transferred between different parts of a system or while it is being processed
- **Caching** in persistence management is a technique used to improve performance and efficiency by storing frequently accessed data in a temporary storage area known as a cache.

- **Disk flushing**" typically refers to the process of ensuring that all data held in a computer's memory cache is written to disk storage.
- **Journaling** involves maintaining a "journal" or "log" of changes that are about to be made to the data. This log records the intended modifications before they are actually applied to the main data storage.
- **Direct writes** involve writing data directly to a file or device, often bypassing the file system cache to ensure that data is immediately and reliably written to storage.
- **Journaling file systems** in kernel-level persistence management ensure data integrity and consistency by keeping a log of changes before they are applied to the file system.



Application of learning 4.1.1

Kernel is the core component of the operating system for example System calls are mechanisms allow user-level processes to communicate with the kernel of the operating system, they are other key terms and data persistence mechanisms that manages data storage and retrieval in a system from here you are requested to write short notes on key terms and data persistence mechanisms.

Checklist

S/N	Assessment criteria (Based on performance criteria)	Indicator/element to check	Observation	
			yes	No
1	**Persistence management is well introduced**	Data buffers is described		
		Caching described		
		Disk flushing described		
		Journaling described		
		Direct writes described		
		Journaling file systems described		

Indicative content 4.2: Management of Mass Storage Structure

Duration: 5 hrs

Theoretical Activity 4.2.1: Description of mass storage physical structure

Notes to the trainer:

- While delivering this content, a small group can be used for describing mass storage physical structure.

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the activity and ask trainees to answer the following questions:
Describe mass storage physical structure by the following ways,

i. Types
ii. Characteristics
iii. Disk attachment
iv. Effect of a device's structure

Step 2: Asks trainee to write answers provided on flip-chart/paper.

Step 3: Asks trainees to share the findings

Step 4: Provides expert view and addresses any questions or concerns if any.

Step 5: Directs the trainees to read the key reading 4.2.1 from trainee's manual.

Points to Remember

- **Mass storage management** includes handling large volumes of data on devices such as hard drives, SSDs, and network-attached storage.
- **Types of mass storage physical structures are:** Hard Disk Drives (HDDs), Solid-State Drives (SSDs), Optical Discs, Flash Drives and Memory Cards.
- **Characteristics of mass storage physical structures include:** HDDs,SSDs, Flash drives. Optical discs
- **Disk attachment with mass storage physical structures** refers to the methods and technologies used to connect storage devices, such as hard disk drives (HDDs), solid-state drives (SSDs), and other storage media, to a computer system.
- **The physical structure of a mass storage device** significantly impacts its performance, reliability, and the way data is managed.

Practical Activity 4.2.2: Applying free space management, disk scheduling algorithm and RAID

Notes to the trainer

- While delivering this content, you are required to write C source codes just applying disk scheduling and source codes for free space management.
- Avail video tutorials

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the topic and ask trainees to read the task below:

As a programmer, you are asked to write c source codes implementing disk scheduling by selecting appropriate algorithm and by writing the codes to do free space management as they are required to build kernel using C.

Step 2: Explain the task and provide clear guiding instructions.

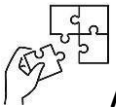
Step 3: Asks trainees to use provided video tutorial demonstrating concrete logical building of disk scheduling and free space management

Step 4: Guide trainees to keep verifying and debugging in case they meet any code defects

Step 5: Directs trainees to copy the code from key reading 4.2.2. From trainee's manual.

Points to Remember

- **Techniques for applying free space management (**Bitmap-Based Free Space Management, Linked List-Based Free Space Management and Free Space Table-Based Management) from the key reading source codes for each is available.
- Select and apply Disk Scheduling Algorithms (**First-Come, First-Served (FCFS)**and Shortest Seek Time First (SSTF**)** from the key reading source codes for each is available.
- **Implementing RAID** (Redundant Array of Independent Disks) techniques in a kernel requires a deep understanding of both disk management and low-level kernel programming.



Application of learning 4.2.

As a trainee from level 5 computer system and architecture you have learnt different programming languages such C and C++ and after acquiring knowledge and skills on persistence management, free space management, disk scheduling algorithm and disk RAID.

SIBOME LTD is a Tech company wanting to hire an expert developer who will write a system program in C or in lower level programming which make free spaces storage on hard disk based on the request from other user's applications, selecting the required algorithm to fit with their need applying disk scheduling based on arrival requests, later those subprograms will be integrated in OS kernel.

NOTE: You need to understand deeply free space management approaches using C, disk scheduling algorithms using C and disk RAID.

As OS developer you are hired to develop this program

CHECK LIST

S/N	Assessment criteria (Based on performance criteria)	Indicator/element to check	Observation	
			Yes	No
1.	**Free space management is well enhanced**	Bit map based, table based or linked list based is selected		
		Source codes for selected free space management are written		
		Programs are working properly and Free spaces are counted		
2.	**Disk scheduling algorithm is well enhanced**	Disk scheduling algorithm is selected		
		Program for each selected algorithm based on kernel project need is written		
		C or low-level programs to do disk scheduling is working		
3.	**Redundant Arrays of Independent Disks is well enhanced**	Different RAID levels and their characteristics mentioned		
		C program written source codes implementing RAID is working		

Indicative content 4.3: Applying I/O Systems

Duration: 5 hrs

Theoretical Activity 4.3.1: Description of I/O systems in OS kernel

Notes to the trainer:

- While delivering this content, a small group can be used for describing I/O systems

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the activity and ask trainees to answer the following questions:

a) With clear ideas explain the different importance of I/O systems in OS kernel
b) What are the Challenges of device heterogeneity?
c) Identify I/O devices based on types and characteristics?
d) Identify I/O device drivers based on function and structure?
e) Identify I/O model based on Blocking I/O, Non-Blocking I/O and I/O Completion Ports respectively

Step 2: Asks trainees to write answers.

Step 3: Asks trainees to share the findings.

Step 4: Provides expert view and addresses any questions or concerns if any.

Step 5: Directs the trainees to read the key reading 4.3.1 from trainee' manual.

Points to Remember

- **The I/O system** encompasses both hardware and software elements, working together to handle input and output operations efficiently and reliably.
- **I/O system is important because of the following:** Facilitates Communication with External Devices, Efficient Resource Management, performance Optimization, Support for Multi-tasking and Concurrency, ….
- **Device heterogeneity refers** to the presence of diverse types of hardware devices in a computing environment, each with its own characteristics, interfaces, and protocols.
- **I/O devices can be categorized into several types**, including storage, network, and human interface devices.
- Device drivers handle specific hardware details, while the I/O system manages request processing, resource allocation, and data transfer.
- **Blocking i/o, non-blocking i/o,** and **i/o completion ports (iocps)** are i/o models.

Practical Activity 4.3.2: Applying I/O systems through Interrupts handling, I/O scheduling and device management

Notes to the trainer

- While delivering this content, you are required to write C source codes to implement I/O system through handling I/O interrupts, I/O scheduling and device management.
- Avail video tutorials

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the topic and ask trainees to read the task below:

As a programmer, you are asked to write c source codes managing interrupts from input and output devices, managing the performance of input and output operations using appropriate algorithm.

Step 2: Explain the task and provide clear guiding instructions.

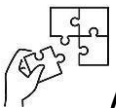
Step 3: Asks trainees to use provided video tutorial demonstrating concrete logical building of handling interrupts, input/output scheduling algorithm and device management.

Step 4: Guide trainees to keep verifying and debugging in case they meet any code defects

Step 5: Directs trainees to copy the code from key reading 4.3.2. From trainee's manual

Points to Remember

- **Interrupt-Driven I/O** is a method where the CPU is notified of I/O operations rather than continuously polling the I/O devices.
- **Applying I/O scheduling** in a kernel involves managing how I/O operations are performed and optimizing the order and timing of these operations. Effective I/O scheduling can improve performance and responsiveness, especially in systems with heavy I/O operations.
- **Device management** involves handling various hardware devices, ensuring that they interact efficiently with the operating system, and managing device-specific operations. When developing a hobby operating system kernel, managing device allocation and deallocation is a fundamental task.



Application of learning 4.3.

As a trainee from level 5 computer system and architecture you have learnt different programming languages such C and C++ and you acquired knowledge and skills on **interrupts handling, i/o scheduling and device management using C programming.**

SIBOME LTD is a Tech company wanting to hire an expert developer who will write a program in C or in lower-level programming which enhance I/O **interrupts, i/o scheduling and manage connected devices**, later those subprograms will be integrated in kernel to handle and manage all the predefined tasks.

As OS developer you are hired to develop this program

CHECKLIST

S/N	Assessment criteria (Based on performance criteria)	Indicator/element to check	Observation	
			Yes	No
1	Interrupt's handling is well enhanced	Basic Setup to handle interrupt is achieved		
		Necessary structures and function prototypes in program to handle interrupt are defined		
		Programmable interrupt Controller is initialized		
2	I/O scheduling is well achieved	Various I/O requests from different sources are achieved		
		Structure to represent an I/O request are defined in a program		
		Selection Scheduling Algorithms implementing multiple I/O requests is enhanced.		
3	Devices are well managed	Structure to represent a device is defined.		
		C program to handle device allocation and deallocation is created.		
		Program created is working and detecting device's errors, allocating them and deallocating.		

Indicative content 4.4: Implementing File Systems

Duration: 5 hrs

Theoretical Activity 4.4.1: Description of file system

Notes to the trainer:

- While delivering this content, a small group can be used for describing file system

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the activity and ask trainees to answer the following questions related to file system**.** Explain clearly file system based on,

i. File system types
ii. Purpose of file systems
iii. File system organization and file system options
iv. Directory structures and file attributes
v. Existing file systems, file system operations and file system selection criteria

Step 2: Asks trainees to write answers.

Step 3: Asks trainees to discuss and agree from their provided answers and choose correct answers.

Step 4: Provides expert view and addresses any questions or concerns if any.

Step 5: Directs the trainees to read the key reading 4.4.1 from trainee' manual.

Points to Remember

- **A file system** is a method used by operating systems to manage and organize files on a storage device, such as a hard drive, SSD, or flash drive. It defines how data are stored, accessed, and organized on the disk, providing a structure for files and directories.
- **In a hobby kernel,** implementing a directory structure involves creating a hierarchical organization of files and directories that enables efficient navigation and management.
- **Tree Structure:** The directory structure resembles a tree, where:
  - The root directory is the topmost node, typically represented as /.
  - Each directory can contain files and subdirectories, creating a branching structure.
- FAT, NTFS, APFS, ZFS, Btrfs By examining these existing file systems, you can draw inspiration for your own design.
- **When developing a hobby kernel**, selecting a file system is a critical decision that impacts performance, security, compatibility, advantages, and limitations.

Practical Activity 4.4.2: Apply basic file, directory, file system management operations respectively and file allocation methods

Notes to the trainer

- While delivering this content, you are required to write C source codes implementing basic file operations, directory operations, file system management operations and file allocation.
- Avail video tutorials

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the topic and ask trainees to do the task below:

As a trainee from level 5 computer system and architecture, you are hired to write a program implementing the following operations on file systems.

i) Basic file operations
ii) Directory operations
iii) File system management operations
iv) File allocation

This program will be produced using C programming, then it will be combined with other subprogram to assist in the complete hobby kernel development project.

Step 2: Explain the task and provide clear guiding instructions.

Step 3: Asks trainees to use provided video tutorial demonstrating file operations, directory operations, file system management operations and file allocation.

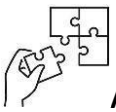
Step 4: Ask trainees to perform the task provided in step 1

Step 5: Guide trainees to keep verifying and debugging in case they meet any code defects

Step 6: Directs trainees to copy the code from key reading 4.4.2. From trainee's manual

Points to Remember

- **Creating a hobby kernel** involves implementing basic file operations such as create, open/close, read/write, and delete, which can be quite an engaging task.
- **Implementing directory operations** in a kernel environment involves managing directories similarly to how files are managed, but with added complexity due to hierarchical structures, basic directory operations are Create, Delete, List contents, Change directory, and Search for a file in a hypothetical kernel.
- **File allocation methods** are crucial for managing how files are stored on disk.

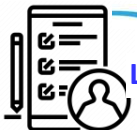


Application of learning 4.4.

As a trainee from level 5 computer system and architecture you have learnt different programming languages such C and C++ you acquired knowledge and skills on file system, file basic operations, directory operations, file system management and file allocation methods.
SIBOME LTD is a Tech company wanting to hire an expert developer who will write a program in C or in lower-level programming which enhance file basic operations, directory operations, file system management and file allocation, later those subprograms will be integrated in kernel to ensure a complete file systems responsibility on hobby kernel project.
As Os developer you are hired to develop this program

CHECKLIST

S/N	Assessment criteria (Based on performance criteria)	Indicator/element to check	Observation	
			Yes	No
1	**Basic file operations are well achieved**	Program implementing file operation is created		
		Basic file operations are achieved correctly with the created program		
2	**Directory operations are well achieved**	Program implementing directory operations is created		
		Basic directory operations are achieved correctly with the created program		
3	**File system Management Operations and file allocation are properly achieved**	File and directory storage, accessibility, movement and permissions are enhanced		
		Structure to allocate files using different method are achieved		
		Program to allocate contiguous blocks on the disk is created and allocation is properly working		



Learning outcome 4 end assessment

Theoretical assessment

Multiple choice questions, choose the correct answer and circle them.

1. **In a linked list-based free space management system, what does each node represent?**
   A) An allocated block
   B) A free block
   C) A file
   D) A user process
   Answer: B
2. **What value is typically used to indicate a free block in a free space table?**
   A) 1
   B) 0
   C) -1
   D) NULL
   Answer: B
3. **Which of the following is NOT a method for managing free space?**
   A) Bitmap
   B) Linked List
   C) Binary Tree
   D) Free Space Table
   Answer: C

True or False Questions

4. In a bitmap-based free space management system, each bit corresponds to a block of memory.

Answer: True

5. Free space tables can only track the allocation status of blocks but not their actual indexes.

Answer: False

6. The linked list method of free space management is more memory-efficient than a bitmap.

Answer: False (It can be less memory-efficient due to the overhead of pointers.)

7. A free space table can allow for faster allocation than a linked list implementation.

Answer: True

8. **Match the free space management methods with their descriptions:**

Method	**Description**
A) Bitmap	1) Uses nodes to represent free blocks
B) Linked List	2) Uses an array where each index represents a block's status
C) Free Space Table	3) Uses bits to indicate allocation status
	4)Uses memory address like IPv4

	5)Uses byte as first memory

Answers:

A) Bitmap - 3

B) Linked List - 1

C) Free Space Table - 2

9. **What is the purpose of a system call?**

   A) To manage memory allocation

   B) To enable user applications to request services from the operating system

   C) To interact with hardware directly

   D) To terminate processes

   Answer: B

10. **What does disk flushing refer to?**

   A) Reading data from the disk

   B) Writing cached data from memory to the disk

   C) Deleting files from a disk

   D) Formatting a disk

   Answer: B

11. **Which of the following is a characteristic of a journaling file system?**

   A) It does not support recovery

   B) It logs changes before they are committed to the disk

   C) It only supports text files

   D) It cannot handle multiple users

   Answer: B

12. **What does device heterogeneity refer to?**

   A) The uniformity of hardware components

   B) The variety of hardware devices in a system

   C) The speed of data transfer between devices

   D) The amount of storage available

   Answer: B

13. **Which RAID level offers data redundancy through mirroring?**

   A) RAID 0

   B) RAID 1

   C) RAID 5

   D) RAID 10

   Answer: B

14. **Answer by True or False:**

   I. Data buffers are permanent storage areas used to hold data for long-term storage.

   Answer: False

   II. In free space management, the buddy system helps to efficiently allocate and deallocate memory blocks.

   Answer: True

III. Disk scheduling algorithms are not important for improving the performance of I/O operations.

Answer: False

IV. The I/O system is responsible for managing communication between the CPU and peripheral devices.

Answer: True

V. Journaling file systems are less reliable than traditional file systems.

Answer: False

15. **Match the terms with their correct descriptions:**

Answer	**Term**	**Description**
A……..6	A) System Call	1) Temporary storage for data during transfer
B…….. 1	B) Data Buffers	2) Signals for asynchronous event handling
C……. 3	C) Disk Attachment	3) Physical connection between storage and the system
D……. 2	D) Interrupts	4) Logical connection between storage and system
E…….. 5	E) File System	5) Structure for organizing and accessing files
		6) Mechanisms for user applications to request OS services

Answers:

16. **Identify two data persistence mechanisms:**

A) File Systems

B) Caches

C) Databases

D) Buffers

Answer: A and C

Practical assessment

As a trainee from level 5 computer system and architecture you have learnt different programming languages such C and C++ you acquired knowledge and skills for setting working environment for hobby kernel, Memory Management, Process Management and persistence management in hobby kernel development using C.
SIBOME LTD is a Tech company wanting to hire an expert developer who will write a program using C and lower-level programming (assembler) which provide the following tasks: Mass storage structure management, I/O systems operations, Handling I/O interrupts and file systems which play a vital role in hobby kernel development.
As OS developer you are hired to develop this program single complete program implementing all the above-mentioned operations.

Checklist

SN	Assessment criteria (Based on performance criteria)	Indicator/element to check	Observation	
			Yes	No
	Mass storage structure management is well achieved	Free space management technics are created based on project need		
		Disk scheduling algorithm and RAID are achieved respectively		
	I/O systems operations is well enhanced	Interrupt service routines (ISRs) is achieved		
		Device allocation and deallocation is enhanced		
	Handling I/O interrupts and file systems is well achieved	Basic file and directory operations are created		
		File system Management Operations are enhanced		
		File Allocation Methods are enhanced		

Further information to the trainer

Bovet, D. P., & Cesati, M. (2005). *Understanding the Linux Kernel, 3rd Edition [Book]*. (n.d.). Www.oreilly.com. https://www.oreilly.com/library/view/understanding-the-linux/0596005652/

Developing a Multithreaded Kernel from Scratch! | Udemy

https://www.linux-magazine.com/Issues/2020/240/Building-a-Hobby-OS

https://www.youtube.com/watch?v=Jl_8XbTEfSQ&list=PLOsF-OO4qVOT6qtNKd4vY3s1ugP_yAw-G

lighthousand. (2014). *books/Modern Operating Systems 4th Edition--Andrew Tanenbaum.pdf at master · lighthousand/books*. GitHub. https://github.com/lighthousand/books/blob/master/Modern%20Operating%20Systems%204th%20Edition--Andrew%20Tanenbaum.pdf

Operating Systems: Three Easy Pieces. (n.d.). Pages.cs.wisc.edu. https://pages.cs.wisc.edu/~remzi/OSTEP/

scheduling-algorithms GitHub Topics GitHub

Sibsankar, H. (2010). *Design of the Unix operating system*. Pearson. https://www.amazon.com/Design-UNIX-Operating-System/dp/0132017997

Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating system concepts* (10th ed.). Wiley. https://www.wiley.com/en-us/Operating+System+Concepts%2C+10th+Edition-p-9781119320913


.ISO
DISK IMAGE

Indicative contents
5.1 Applying Assembly Language concepts
5.2 Initialization of the core kernel.
5.3 Development of a simple Bootloader with assembly language
5.4 Implementation of Interrupt Handling
5.5 Implementing a function to display console output on the screen in a kernel
5.6 Implement publication of our system

Key Competencies for Learning Outcome 5: Implement Core Kernel

Knowledge	Skills	Attitudes
● Description assembly Language and x86 architecture.	● Developing assembly hello, world. ● Applying register commands. ● Writing and build C and Assembly kernel objects. ● Developing a simple boot loader. ● Implementing of OS displays. ● Implementing system calls	● Having Paying Attention ● Being Flexibility ● Having Team work ● Being Innovative ● Having Creativity ● Having Confidence

Duration: 40 hrs

Learning outcome 5 objectives:

By the end of the learning outcome, the trainees will be able to:

1. Describe correctly assembly language used in implementing core kernel.
2. Apply correctly assembly language based on language standards.
3. Applying properly register commands as used in implementing core kernel.
4. Write, build C and Assembly correctly kernel objects as used in implementing core kernel.
5. Develop properly a simple boot loader used in implementing core kernel.
6. Implement effectively OS displays as used in core kernel.
7. Implement clearly system calls as used in implementing core kernel.

Resources

Equipment	Tools	Materials
● Computer ● Projector	● HexEditor ● Code editor ● NASM ● SASM ● QEMU ● QtEMU ● Compiler ● VMware workstation	● Internet ● Electricity

Advance Preparation:

Before delivering this learning outcome, you are recommended to:

- Avail QEMU, VMware workstation, text editor setup.
- Avail files or project related to operating system development

Indicative content 5.1: Applying Assembly Language Concepts

Duration: 5 hrs

Theoretical Activity 5.1.1: Description of Assembly Language concepts

Notes to the trainer:

- While delivering this content, a small group can be used for describing assembly language concepts.
- The use of videos as didactic materials is required.

Key steps:

While delivering this activity, pass through the following steps:

Step 1. Introduce the activity and ask trainees to answer the following questions:

Discuss the foundational concepts of Assembly Language, including its definition, history, and advantages. In your response, outline the primary sections of an Assembly Language program, explain the role of comments, and provide an overview of statement syntax. Illustrate your answer with a simple "Hello, World!" output and specify the typical file extension used for Assembly Language files. Additionally, describe the role of compilers in the context of Assembly Language, and detail the data sizes and endianness relevant to the x86 processor architecture. Finally, classify the various types of registers (General, Control, Segment) and elaborate on the management of variables within Assembly Language programming.

Step 2. Ask trainees to present their findings to the whole class.

Step 3. Provides expert view and clarifies ideas by using didactic materials.

Step 4. Address any questions or concerns.

Step 5. Ask trainee to read the key readings on activity 5.1.1 in trainee manual

Points to Remember

- **Assembly Language** is a low-level programming language that provides a symbolic representation of a computer's machine code. It serves as an interface between high-level programming languages and the hardware, allowing programmers to write instructions that are closely aligned with the architecture of a specific processor.
- While **Assembly Language** is more complex and time-consuming to write compared to high-level languages, it is still used in scenarios where performance and resource management are critical, such as in operating systems, device drivers, and real-time systems.

Practical Activity 5.1.2: Compiling and Linking an Assembly Program in NASM

Notes to the trainer

- This activity should take place in a computer lab where trainees should Compiling and Linking an Assembly Program in NASM
- Avail computers and installed software

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the topic and ask trainees do the task described below:
Create a simple NASM Assembly program that defines a function to add two integers. Then, write a C program that calls this Assembly function and prints the result. Explain the steps to compile and link both the Assembly and C programs using a Windows environment, including the commands or tools needed. Explain the task and provide clear work instruction.

Step 2: Explain the task and give clear instruction

Step 3: Demonstrate how to Compiling and Linking an Assembly Program in NASM

Step 4: Ask trainees to Compiler and Link an Assembly Program in NASM and monitor the procedures.

Step 5: Verify whether the Assembly Program is Compiled and Linked in NASM correctly

Step 6: Ask trainees to read key reading 5.1.2.in trainee manual

Points to Remember

- **Steps to Compile and Link:**
- Install NASM
- Compile the Assembly Program
- Compile the C Program
- Link Both Object Files
- Run the Program

Practical Activity 5.1.3: Applying memory segments

Notes to the trainer

- This activity should take place in a computer lab where trainees should Applying memory segments

- Avail computers and installed software

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the topic and ask trainees do the task described below:

Write an Assembly program using NASM that demonstrates the use of different memory segments: the data segment, code segment, and stack segment. Your program should perform the following tasks:

1.Data Segment: Define a string message and an integer variable in the data segment.

2.Code Segment: Write a procedure that prints the string message.

3.Stack Segment: Use the stack to store a local variable and calculate its square, then print the result.

Step 2: Explain the task and give clear instruction

Step 3: Demonstrate how to Apply memory segments

Step 4: Ask trainees to Applying memory segments

Step 5: Verify whether the memory segments are applied correctly

Step 6: Ask trainees to read key reading 5.1.3.in trainee manual

Points to Remember

- When applying memory segments in Assembly Language, you typically deal with three main segments: the data segment, code segment, and stack segment.

Practical Activity 5.1.4: Applying different categories of register

Notes to the trainer

- This activity should take place in a computer lab where trainees should Applying memory segments
- Avail computers and installed software

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the topic and ask trainees do the task described below:

Write a simple Assembly program for a hobby kernel that demonstrates the use of different categories of registers in the x86 architecture. Your program should:

1.Use General Purpose Registers to perform basic arithmetic (e.g., adding two numbers).

2.Utilize a Control Register to enable or modify a feature (e.g., setting the CR0 register to switch to protected mode).

3.Leverage Segment Registers to manage memory segments (e.g., loading values into the CS or DS register).

Step 2: Explain the task and give clear instruction

Step 3: Demonstrate how to Apply memory segments

Step 4: Ask trainees to Applying memory segments

Step 5: Verify whether the memory segments are applied correctly

Step 6: Ask trainees to read key reading 5.1.4.in trainee manual

Points to Remember

- Applying different categories of registers in a hobby kernel involves understanding their roles and using them effectively in your code

Practical Activity 5.1.5: Applying system calls

Notes to the trainer

- This activity should take place in a computer lab where trainees should Applying Apply system calls
- Avail computers and installed software

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the topic and ask trainees do the task described below:

You are developing a simple file management utility in C that needs to create, read, and delete files.

1.Creating a file: Write a function using the appropriate system call to create a new file named "example.txt" and write "Hello, World!" into it.

2.Reading a file: Implement another function that reads the contents of "example.txt" and prints it to the console.

3.Deleting a file: Finally, write a function that deletes "example.txt" from the file system.

Step 2: Explain the task and give clear instruction

Step 3: Demonstrate how to Apply system calls

Step 4: Ask learners to Apply system calls

Step 5: Verify whether the system calls are applied correctly

Step 6: Ask trainees to read key reading 5.1.5

Points to Remember

- Applying system calls typically involves using specific functions provided by the operating system in a programming language like C.
- System calls are the fundamental interface between an application and the operating system (OS). They allow user-level programs to request services from the OS kernel, enabling them to perform tasks that require higher privileges, such as accessing hardware or managing system resources.

Practical Activity 5.1.6: Applying addressing mode

Notes to the trainer

- This activity should take place in a computer lab where trainees should Applying Apply addressing mode
- Avail computers and installed software

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the topic and ask trainees do the task described below:

As a hobby kernel, implement a simple memory management module that demonstrates the following addressing modes:

1.Immediate Addressing: Create a function that takes a constant value and returns its doubled value without using any memory location.

2.Direct Addressing: Create a function that takes a pointer to an integer in memory and increments its value by 1.

3.Indirect Addressing: Create a function that takes a pointer to a pointer of an integer, dereferences it to get the original integer, and squares its value.

Step 2: Explain the task and give clear instruction

Step 3: Demonstrate how to Apply addressing mode

Step 4: Ask trainees to Apply addressing mode

Step 5: Verify whether the addressing mode are applied correctly

Step 6: Ask trainees to read key reading 5.1.6

Points to Remember

- **In immediate addressing mode**, the operand is a constant value embedded directly within the instruction itself. This means that the value is provided as part of the instruction rather than being fetched from a memory location.
- **In register addressing mode,** the operand is located in a register within the CPU
- **In memory addressing mode**, the operand is located in memory. The instruction specifies a memory address from which the data should be fetched or to which data should be written.

Practical Activity 5.1.7: Applying variables and constants

Notes to the trainer

- This activity should take place in a computer lab where trainees should Applying variables and constants
- Avail computers and installed software

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the topic and ask trainees do the task described below:

Write a C program that calculates the area and circumference of a circle.

i. Define a Constant: Use #define to create a constant for the value of π (pi).
ii. Use Variables: Declare a variable to hold the radius of the circle, and use additional variables to store the calculated area and circumference.
iii. Implement a Function: Create a function that takes the radius as an argument and returns the area and circumference.
iv. Display the Results: In the main function, prompt the user for the radius, call your function, and display the results.

Step 2: Explain the task and give clear instructions

Step 3: Demonstrate how to Apply variables and constants in program

Step 4: Ask learners to Apply variables and constants in program

Step 5: Verify whether variables and constants are applied correctly in c program

Step 6: Ask trainees to read key reading 5.1.7 in trainee manual

Points to Remember

- **A variable** is a named storage location in a program that can hold different values during the program's execution. Variables are used to store data that can change or be modified. Each variable has a specific type that determines what kind of data it can hold (e.g., integers, floats, characters).
- **A constant** is a fixed value that cannot be changed during the program's execution. Constants are useful for defining values that should remain the same throughout the program, such as mathematical constants or configuration settings. In C, constants can be defined using #define or the const keyword.

Practical Activity 5.1.8: Applying datatypes and Operators

Notes to the trainer

- This activity should take place in a computer lab where trainees should Applying variables and constants
- Avail computers and installed software

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the topic and ask trainees do the task described below:
Write an assembly program that counts up a "score" value based on certain conditions, loops to increment the score multiple times, and displays a final message with the score result. The score counter starts at zero, increments by a certain value each iteration, and stops once it reaches or exceeds a target score.

- Use conditional jumps to check if the score has reached a target.
- Implement a loop that increments the score.
- Perform basic arithmetic to update the score.
- Display the final score as a message.

Step 2: Explain the task and give clear instruction

Step 3: Demonstrate how to Apply variables and constants in program

Step 4: Ask learners to Apply variables and constants in program

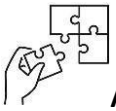
Step 5: Verify whether variables and constants are applied correctly in c program

Step 6: Ask trainees to read key reading 5.1.8 in trainee manual

Points to Remember

Operations and Control Flow

- **Arithmetic and Logical Instructions**: Instructions for mathematical operations (e.g., ADD, SUB) and logic operations (e.g., AND, OR).
- **Conditions**: Implement conditional execution with comparison instructions (e.g., CMP).
- **Loops (Optional)**: Create loops for repeated execution (e.g., using jump instructions).
- **Numbers**: Handling integer and floating-point numbers.
- **Strings**: Operations on strings using specific instructions (e.g., MOVS).
- **Arrays**: Access and manipulate arrays in memory.



Application of learning 5.1.

Create a simple assembly program with the following requirements:

1. **Display a Message**: Output "Hello, World!" to the screen.
2. **Memory Segments**:
    - **Data Segment**: Store the message.
    - **Code Segment**: Contain the executable code.
    - **Stack Segment**: Use for function calls or local variables.
3. **Registers**:
    - Utilize general-purpose registers for data handling.
    - Implement a control register for system calls if needed.
4. **System Calls**: Use appropriate calls to print the message.
5. **Addressing Modes**:
    - Use **register addressing** for operations.
    - Use **immediate addressing** for constants.
    - Use **memory addressing** for accessing the data segment.
6. **Variables and Constants**: Define a variable and use it in the program.
7. **Arithmetic and Logical Instructions**: Perform an arithmetic operation (e.g., addition) and store the result in a variable.
8. **Conditions**: Implement a conditional statement based on the arithmetic result and print a different message if the condition is met.
9. **Loops (Optional)**: Use a loop to print the message multiple times based on a counter.
10. **Arrays**: Define an array in the data segment and calculate the sum of its elements.

Deliverables:

- Assembly source code file (hello.asm).
- Explanation of the implementation of each component.
- Compile and link the program using NASM and demonstrate its execution.

Checklist

SN	Assessment criteria (Based on performance criteria)	Indicator/element to check	Observation	
			Yes	No
1	**Memory Segment is well created**	Data segment is created		
		Stack segment is created		
2.	**Register is well enhanced**	Register is used		
		Register is implemented		
3.	**System call is well achieved**	System call is performed		
		File is retrieved		
4.	**Variables and Constants are well used**	Variables are used		
		Constants are used		
5.	**Condition and loop are well performed**	Condition is used		
		Loop is used		

Indicative content 5.2: Initialization of the Core Kernel

Duration: 5 hrs

Theoretical Activity 5.2.1: Description of core kernel

Notes to the trainer:

- While delivering this content, a small group can be used for core kernel.
- The use of videos as didactic materials is required

Key steps:

While delivering this activity, pass through the following steps:

1. Introduce the activity and ask trainees to answer the following questions:
   i. What is the kernel's role in an OS, and how does it differ from user-level applications?
   ii. How has kernel design evolved from early systems to modern architectures, and what factors influenced this evolution?
   iii. Compare monolithic, micro, hybrid, and exokernels. What are their advantages and disadvantages?
   iv. What are the main components of a kernel layout, and what functions do they serve?
   v. What primary functions does the kernel serve in resource management and application services?
   vi. What is the difference between kernel mode and user mode, and why is this distinction crucial for system stability and security?
   vii. What key tasks does the kernel perform to accomplish its functions, and how do these contribute to the OS's overall functionality?
   viii. What are device drivers, why are they essential, and what types exist?
2. Ask trainees to present their findings to the whole class.
3. Provides expert view and clarifies ideas by using didactic materials.
4. Address any questions or concerns.
5. Ask trainee to read the key readings on activity 5.2.1

Points to Remember

- **Kernel's Role**: Core of the OS, manages hardware and enables system-level operations, unlike user applications with limited access.

- **Evolution of Kernel Design**: Evolved from simple to complex, driven by hardware advances, security needs, and performance requirements.
- **Kernel Types**:
    - **Monolithic**: Fast, but complex.
    - **Microkernel**: Modular, but can be slower.
    - **Hybrid**: Balances speed and modularity.
    - **Exokernel**: High efficiency but complex for developers.
- **Kernel Components**: Manages processes, memory, files, devices, and security.
- **Resource Management**: Allocates CPU, memory, and I/O, supporting application services.
- **Kernel vs. User Mode**: Kernel mode has full system access; user mode is restricted for security.
- **Key Tasks**: Scheduling, memory, I/O, and system calls enable OS functionality.
- **Device Drivers**: Essential for hardware communication, enabling device functionality in the OS.

Practical Activity 5.2.2: write and build object file from assembly Language

Notes to the trainer

- This activity should take place in a computer lab where trainees should Compiling and Linking an Assembly Program in NASM
- Avail computers and installed software

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the topic and ask trainees do the task described below:
As a hope kernel developer, you are asked to write and build object file by using assembly Language

Step 2: Explain the task and give clear instruction

Step 3: Demonstrate how to Compiling and Linking an Assembly Program in NASM

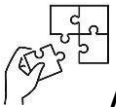
Step 4: Ask learners to Compiler and Link an Assembly Program in NASM and monitor the procedures.

Step 5: Verify whether the Assembly Program is Compiled and Linked in NASM correctly

Step 6: Ask trainees to read key reading 5.2.2.

Points to Remember

- **Assemble boot code:**
  nasm -f elf32 boot.asm -o boot.o
- **Compile kernel code:**
  gcc -m32 -ffreestanding -c kernel.c -o kernel.o
- **Link with linker script:**
  ld -m elf_i386 -T linker.ld -o kernel.bin boot.o kernel.o
- **Test in QEMU:**
  qemu-system-i386 -kernel kernel.bin



Application of learning 5.2.

Develop a simple kernel to display "Hello, Kernel!" on the screen.

1. **Create an Assembly File** (boot.asm):
   - Initialize the stack.
   - Call the C function _kernel_main.
2. **Implement the C Function** (kernel.c):
   - Define _kernel_main to write "Hello, Kernel!" to VGA text mode.
3. **Compile with NASM**:
   - Convert boot.asm to an object file.
4. **Compile with GCC**:
   - Compile kernel.c into an object file with the appropriate flags.
5. **Write a Linker Script** (linker.ld):
   - Specify the entry point and memory layout.
6. **Link Object Files**:
   - Merge the assembly and C object files to create kernel.bin.
7. **Test the Kernel**:
   - Run kernel.bin using an emulator like QEMU to verify the output.

Deliverables:

- Source files (boot.asm, kernel.c, linker.ld)
- Final executable (kernel.bin)
- Brief report on challenges and resolutions.

Checklist

SN	Assessment criteria (Based on performance criteria)	Indicator/element to check	Observation	
			Yes	No
1	**Assembly File is well created**	Stack is created		
		Function is created		
2.	**NASM and GCC are well compiled**	boot.asm file is used		
		GCC file is implemented		
3.	**Linker Script is well achieved**	Linker.Id is performed		
		File is retrieved		

Indicative content 5.3: Development of a Simple Bootloader with Assembly Language

Duration: 5hrs

Theoretical Activity 5.3.1: Introduction of Bootloader with assembly

Notes to the trainer:

- While delivering this content, a small group can be used for describing Hobby kernel.

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the activity and ask trainees to answer the following questions:

i. What is Bootloader?
ii. What are the languages used to implement the bootloader?

Step 2: Ask trainees to write answers provided on flip-chart/paper or blackboard.

Step 3: Asks trainees to discuss the provided answer and choose correct answers.

Step 4: Provides expert view and clarifies ideas.

Step 5: Address any questions or concerns.

Step 6: Ask trainees to read the key reading 5.3.1 in the trainee manual.

Points to Remember

1. **Definition**: A bootloader is a small program that initializes a computer when powered on and loads the operating system (OS) into memory.
2. **Memory Location**: It typically resides in specific memory locations, like the Master Boot Record (MBR) on hard drives.
3. **Programming Languages**: Bootloaders are primarily written in low-level languages:
   - **Assembly Language**: For direct hardware control.
   - **C Language**: For more complex functionalities.
4. **Compilers/Assemblers**: Common tools include:
   - **GCC**: For compiling C and assembly.
   - **NASM** and **FASM**: Popular assemblers for writing bootloader code.
5. **Boot Process**:
   - **POST**: Firmware checks hardware.
   - **Loading the Bootloader**: Firmware loads the bootloader into memory.
   - **Execution**: The bootloader initializes the system and loads the OS kernel.
6. **Architecture Components**:
   - **Entry Point**: First instruction executed (e.g., 0x7C00).
   - **Initialization Code**: Prepares the CPU.

- **Loading Mechanism**: Loads OS/kernel into memory.
- **Execution Transfer**: Jumps to the OS/kernel entry point.
- **Error Handling**: Manages loading errors.

7. **Development Environment**:
   - Requires tools like an assembler (NASM), text editor, emulator (QEMU/Bochs), disk image tools (e.g., dd), and debugging tools (GDB).

Practical Activity 5.3.2: Implement basic BIOS Interrupts and screen clearing

Notes to the trainer

- The trainer is required to explain operating system architecture

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the topic and ask trainees to perform the following tasks:
As software developer, you are requested to go to the computer lab and write C Program that will Snippet for Screen Clearing and Using BIOS Interrupts.

Step 2: Provide instruction that will be followed

Step 3: Demonstrate how to implement BIOS Interrupt, while demonstrating, explain the steps to follow.

Step 4: Asks Trainees to BIOS Interrupt and monitor the procedures.

Step 5: Verify whether scheduling algorithm are clearly applied

Step 6: Ask trainees to read key reading 5.3.2. in the Trainee's Manual

Points to Remember

1. **File Format**: The example code must be saved with a .nasm extension, indicating it's written for the NASM assembler.
2. **Code Structure**:
   - **Entry Point**: The code starts execution at the _start label.
   - **Section Directive**: The code is defined in the .text section, which is standard for executable code.
3. **Clearing the Screen**:
   - **Setting Video Mode**: The instruction mov ax, 0x0003 sets the video mode to 80x25 text mode by invoking BIOS interrupt 0x10.
   - **Clearing the Screen**: The mov ah, 0x0E prepares to write a character (space) repeatedly to clear the screen, with cx set to 2000 (the total number of characters).

4. **Looping**: The clear_loop uses the int 0x10 BIOS interrupt to print the space character, looping until the screen is cleared.
5. **Next Stage**: The comment indicates that loading the next stage (e.g., reading from disk) is not implemented in this snippet, but it's typically part of a bootloader's functionality.
6. **Boot Signature**:
    - The line times 510-($-$$) db 0 fills the remaining space of the boot sector with zeros.
    - dw 0xAA55 adds a boot signature at the end, which BIOS uses to recognize the sector as bootable

Practical Activity 5.3.3: Implement Mixed code

Notes to the trainer

- Avail of computer

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the topic and ask trainees to perform the following tasks:

As software developer, you are requested to go to the computer lab and write C Program that will perform the following tasks:

a. **CString.c**: Contains string manipulation functions.
b. **CDisplay.c**: Contains functions for screen output.
c. **Types.h**: Header file for common type definitions.
d. **BootMain.c**: The main C entry point for the bootloader.
e. **StartPoint.asm**: The assembly entry point and initialization code.

Step 2: Provide instruction that will be followed

Step 3: Demonstrate how to mix code of Cstrings, CDisplay, Types.h, BootMian andStartPoint, while demonstrating, explain the steps to follow.

Step 4: monitor the procedures.

Step 5: Verify whether scheduling algorithm are clearly applied

Step 6: Ask trainees to read key reading 5.3.3. in the Trainee's Manual

Points to Remember

1. **File Structure Overview**:
   - **CString.c**: Implements string manipulation functions like strcpy and memset.
   - **CDisplay.c**: Contains functions for outputting characters to the screen.
   - **Types.h**: Defines common data types (BYTE, WORD, DWORD).
   - **BootMain.c**: The main C entry point for the bootloader.
   - **StartPoint.asm**: Contains the assembly entry point and initialization code.

8. **Code Snippets**:
   - **Types.h**: Uses #define guards to prevent multiple inclusions and defines basic types.
   - **CString.c**:
     - strcpy: Copies a string from source to destination and null-terminates it.
     - memset: Fills a memory block with a specified value.
   - **CDisplay.c**:
     - putchar: Uses BIOS interrupt 0x10 to print a character with specified attributes.
     - puts: Prints a string by calling putchar for each character.
   - **BootMain.c**:
     - Displays a welcome message and a loading message.
     - Uses an infinite loop to halt execution (in a real bootloader, further actions would occur).
   - **StartPoint.asm**:
     - Sets up the stack and calls BootMain.
     - Contains an infinite loop to halt execution.
     - Includes a boot signature to ensure BIOS recognition.

9. **Build and Compile:**
   - **Compilation Tools**: Use NASM for assembly files and GCC for C files.
   - **Makefile**:
     - Defines object files and the final bootloader binary.
     - Specifies commands to compile each source file and link them into a bootloader binary.
     - Ensures the final binary has a boot signature by filling space with zeros and appending 0xAA55.

Practical Activity 5.3.4: Assembling files

Notes to the trainer

- Avail of computer

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the topic and ask trainees to perform the following tasks:
As software developer, you are requested to go to the computer lab and write C Program that will automate assembly using a COM File

Step 2: Provide instruction that will be followed

Step 3: Demonstrate how to create COM file to automate assembly file, while demonstrating, explain the steps to follow.

Step 4: monitor the procedures.

Step 5: Verify whether scheduling algorithm are clearly applied

Step 6: Ask trainees to read key reading 5.3.4. in the Trainee's Manual

Points to Remember

1. **Creating a COM File**:
    - A COM file is a simple executable format that the BIOS can load and run.
    - **Memory Address**: The code for COM files starts at the address 0x0100.
    - **Size Limit**: The total code must fit within the 64 KB limit of a COM file.
2. **Assembly Automation**:
    - You can automate the compilation and linking process using a Makefile.
3. **Complete Makefile Example**:
    - **Output File**: The final output is specified as bootloader.com.
    - **Object Files**: Lists the object files that will be created: StartPoint.o, BootMain.o, CString.o, CDisplay.o.
    - **Compiler and Assembler**:
        - **CC**: Set to gcc for C compilation.
        - **AS**: Set to nasm for assembly.
    - **Flags**:
        - **CFLAGS**: Includes -c (compile only), -ffreestanding (no standard library), and -m16 (16-bit mode).
        - **ASFLAGS**: Set to -f bin for binary output.

- **COM File Rule**: The target rule for creating the COM file:
  - Combines object files using cat.
  - Pads the file with zeros to ensure it's a multiple of 512 bytes.
  - Appends the boot signature 0x55AA at the end.
- **Assembly and Compilation Rules**:
  - Specific rules for creating object files from assembly and C source files.
- **Clean Rule**: Removes generated object files and the COM file.

Practical Activity 5.3.5: Test and Demonstrate bootloader in virtual machine

Notes to the trainer

- Avail of computer

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the topic and ask trainees to perform the following tasks:

As software developer, you are requested to go to the computer lab and perform the following task:

I. Create Virtual machine using Vmware work station

II. Convert the COM file to a floppy image (IMG)

III. Testing Bootloader in virtual machine

Step 2: Provide instruction that will be followed

Step 3: Explain the steps to follow.

Step 4: Monitor the procedures.

Step 5: Verify whether scheduling algorithm are clearly applied

Step 6: Ask trainees to read key reading 5.3.5. in the Trainee's Manual

Points to Remember

1. **General Steps to Test the Bootloader**:
   - **Create the COM File**: Compile your bootloader into a valid COM file.
   - **Use Bootable Media**: Write the COM file to a bootable medium (USB drive or floppy disk).
   - **Monitor Output**: Prepare to observe the output to ensure the bootloader operates as expected.
2. **Testing with Virtual Machine (VMware)**:
   - **Create a Virtual Machine**:

1. Open VMware and create a new virtual machine.
2. Choose Custom (advanced) and select the appropriate hardware compatibility.
3. Select "Other" for the guest OS.

- **Configure Virtual Machine Settings**:
  1. Allocate memory (e.g., 512 MB RAM) and CPU resources.
  2. Add a small virtual disk (1 GB is sufficient for testing).
- **Attach the Bootloader**:
  1. Convert the COM file to a floppy image using the command:
     dd if=bootloader.com of=boot.img bs=512 count=1
  2. In VMware settings, attach the image as floppy disk
- **Boot the Virtual Machine**:

1. Start the VM and ensure it boots from the floppy drive.
2. Observe the console output for expected behaviors or messages.

3. **Testing on Real Hardware**:

- **Create a Bootable USB**:
  1. Write the COM file to a USB drive using a command like:
     sudo dd if=bootloader.com of=/dev/sdX bs=512 conv=notrunc
  2. Ensure the USB drive is marked as bootable.
- **Configure BIOS/UEFI Settings**:
  1. Insert the USB drive and reboot the computer.
  2. Enter BIOS/UEFI setup (using keys like F2, F12, DEL, or ESC).
  3. Adjust the boot order to prioritize the USB drive.
- **Boot the System**:
  1. Save changes and exit BIOS/UEFI.
  2. Monitor the output on the screen for expected behavior from the bootloader.

4. **Common Troubleshooting Tips**:

- **Black Screen or No Output**: Verify the bootloader is correctly written and check the boot order.
- **Unexpected Behavior**: Review the code for logical errors or unhandled cases.
- **Testing Environment Issues**: Ensure hardware settings (like virtualization support) are enabled in BIOS if it works in VMware but not on real hardware.

Theoretical Activity 5.3.6: Description of Debugging our system with tools

Notes to the trainer:

- While delivering this content, show trainee some video related to debugging of software.

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the activity and ask trainees to answer the following questions:

I. What is Debugging in kernel development?
II. Describe debugging tools in debugging Bootloader?

Step 2: Ask trainees to write answers provided on flip-chart/paper or blackboard.

Step 3: Asks trainees to discuss the provided answer and choose correct answers.

Step 4: Provides expert view and clarifies ideas.

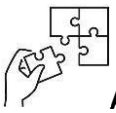
Step 5: Address any questions or concerns.

Step 6: Ask trainees to read the key reading 5.3.6 in the trainee manual.

Points to Remember

1. **Turbo Debugger (TD)**:
    - **Setup**:
        - Install Turbo Debugger in your development environment.
    - **Debugging Session**:
        - Boot the system or emulator (e.g., QEMU or VMware) with the bootloader.
        - Set breakpoints in the bootloader code to enter the debugger.
        - Load the bootloader COM file in Turbo Debugger.
        - Use commands like G (Go), T (Trace), and B (Breakpoint) to control execution.
    - **Tips**:
        - Set breakpoints at critical sections, especially after initialization.
        - Utilize the memory window to monitor registers and memory changes.
2. **CodeView**:
    - **Setup**:
        - Ensure CodeView is installed.
    - **Debugging Steps**:
        - Compile the bootloader with debug symbols.
        - Launch CodeView and load the bootloader.
        - Set breakpoints, step through the code, and inspect memory and registers.
    - **Tips**:
        - Use the MAP file generated during compilation to locate functions and variables.
        - Familiarize yourself with CodeView's command set for effective debugging.
3. **D86**:
    - **Setup**:

        - Download and install D86 from a reliable source.
    - **Debugging Process**:
        - Start D86 and load your bootloader.
        - Use commands to examine memory (D), set breakpoints (B), and run the program (G).
    - **Tips**:
        - Command syntax is concise; refer to the documentation for details.
        - Closely monitor the stack and registers to identify issues.
4. **Bosch**:
    - **Setup**:
        - Ensure Bosch is installed and configured.
    - **Debugging Workflow**:
        - Load your bootloader into Bosch.
        - Set breakpoints and step through the execution flow.
    - **Tips**:
        - Utilize disassembly features to trace through the assembly code.
        - Pay attention to registers and memory to spot incorrect behaviors.
5. **General Debugging Tips**:
    - **Use Emulators**: Tools like QEMU and Bochs often have built-in debugging capabilities and allow better control of the execution environment.
    - **Logging**: Implement simple output functions to log execution paths or variable states for better visibility into issues.
    - **Incremental Testing**: Make small changes to the bootloader and test incrementally to understand the impact of each modification.



Application of learning 5.3.

XYZ Electronics Ltd, located in Gasabo district, is a growing startup that specializes in creating custom computer systems. They have built a unique computer designed for running specific applications, but it currently lacks a boot loader, preventing it from initializing properly and loading the necessary software. As an OS Developer, you are tasked with creating a boot loader that will enable the system to boot up and run the designated applications effectively.

Task:

1. Develop a boot loader that initializes the hardware and loads an application.
2. Generate a COM file for the boot loader.
3. Prepare clear and concise documentation explaining how to install and use the boot loader.

Instructions:

1. Implement basic BIOS interrupts for hardware initialization.
2. Integrate mixed code (Assembly and C) to ensure functionality.

3. Ensure the boot loader can load applications seamlessly into memory.
4. Implement error handling routines to manage hardware initialization failures.
5. Ensure the boot loader is resilient to power cuts, maintaining state upon reboot.
6. Create a simple console interface for user interaction during the boot process.
7. Test the boot loader in both a virtual machine environment and on real hardware.

Checklist

S/N	Assessment criteria (Based on performance criteria)	Indicator/element to check	Observation	
			Yes	No
1.	Working Environment is well prepared	Tools and equipment are selected		
		Software tools are installed		
		Virtual environment is configured		
2.	BIOS Interrupts is well implemented	BIOS interrupts are implemented		
		Screen clearing function is implemented		
		Error handling routines are implemented		
3.	Mixed Code is well implemented	Mixed code integration is implemented		
		Functionality is implemented		
		Linking is implemented		
4.	COM File are well Assembled and Generated	Assembly is completed		
		COM file Is created		
5.	Testing and Documentation are well done	Testing in VM is completed		
		Testing on hardware is completed		
		Documentation is provided		
		Resilience is demonstrate		

Indicative content 5.4: Implementation of Interrupt Handling

Duration: 10 hrs

Theoretical Activity 5.4.1: Description of Interrupt Handling

Notes to the trainer:

- Trainer may show the trainee on the physical machine how handle interrupted

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the activity and ask trainees to answer the following questions:

I. What is Interrupt and Interrupt Handling?
II. What are the primary responsibilities of an interrupt handler?

Step 2: Ask trainnees to write answers provided on flip-chart/paper or blackboard.

Step 3: Asks trainees to discuss the provided answer and choose correct answers.

Step 4: Provides expert view and clarifies ideas.

Step 5: Address any questions or concerns.

Step 6: Ask trainees to read the key reading 5.4.1 in the trainee manual.

Points to Remember

- I**nterrupt Handling**: Essential for responding to urgent events, allowing the CPU to pause, handle the event, and resume tasks.
- **Interrupt Definition**: A signal from hardware or software that indicates an event requiring immediate attention.
- **Interrupt Handler (ISR)**:

✓ Saves the current CPU state.
✓ Processes the interrupt.
✓ Restores the CPU state after handling.

- **Interrupt Descriptor Table (IDT)**:

✓ Defines interrupt vectors.
✓ Contains interrupt vector, handler address, and attributes.

- **Programmable Interrupt Controller (PIC)**:

✓ Manages interrupt signals.
✓ Prioritizes, masks, and routes interrupts to the CPU

Practical Activity 5.4.2: Applying interrupt handlers

Notes to the trainer

- Avail of computer

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the topic and ask trainees to perform the following tasks:
As software developer, you are requested to go to the computer lab and write C Program that will handle the interrupt

Step 2: Provide instruction that will be followed

Step 3: Demonstrate how to create COM file to automate assembly file, while demonstrating, explain the steps to follow.

Step 4: monitor the procedures.

Step 5: Verify whether scheduling algorithm are clearly applied

Step 6: Ask trainees to read key reading 5.4.2. in the Trainee's Manual

Points to Remember

- **Define the Interrupt Handler**: Create a short ISR that contains the logic for handling the interrupt.
- **Register the Interrupt Handler**: Link the ISR to a specific interrupt vector in the IDT by specifying the vector number and handler address.
- **Configure the PIC/APIC**: Set up the PIC or APIC to manage interrupts, including specifying which IRQs to listen to.
- **Enable Interrupts**: Set CPU flags to allow interrupts by executing appropriate assembly instructions.
- **Handle Interrupt Context**: Save and restore the CPU context within the ISR to ensure proper resumption of the interrupted task.
- **Testing and Debugging**: Test the interrupt handler under various conditions and use debugging tools to verify correct behavior.
- **Optimization**: Optimize the ISR for efficiency, minimizing processing within the handler and deferring complex tasks to the main program context.

Practical Activity 5.4.3: Creating an Entry in Interrupt Descriptor Table (IDT)

Notes to the trainer

- Avail of computer

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the topic and ask trainees to perform the following tasks:
As software developer, you are requested to go to the computer lab and write C Program that will create IDT?

Step 2: Provide instruction that will be followed

Step 3: Demonstrate how to create IDT, while demonstrating, explain the steps to follow.

Step 4: monitor the procedures.

Step 5: Verify whether scheduling algorithm are clearly applied

Step 6: Ask trainees to read key reading 5.4.3. in the Trainee's Manual

Points to Remember

- **IDT Structure Definition**: Each IDT entry contains fields for the handler's address, segment selector, reserved space, and type/attributes.
- **IDT Array**: An array of idt_entry_t structures holds the IDT entries, with a defined size (256 entries in this case).
- **Load IDT Function**: An external assembly function is declared to load the IDT into the CPU.
- **Interrupt Handler**: Define an interrupt handler function that contains the logic for handling the interrupt.
- **Set IDT Entry Function**: The set_idt_entry function populates a specific entry in the IDT with the handler's address, selector, and attributes.
- **Setup IDT Function**: The setup_idt function registers an interrupt handler for a specific vector (e.g., IRQ 32) and loads the IDT.
- **Enable Interrupts**: The enable_interrupts function uses inline assembly to set the interrupt flag, allowing the CPU to respond to interrupts.

Practical Activity 5.4.4: Handling an Interrupt

Notes to the trainer

- Avail of computer

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the topic and ask trainees to perform the following tasks:

As software developer, you are requested to go to the computer lab and write C Program that will handling an interrupt

Step 2: Provide instruction that will be followed

Step 3: Demonstrate how to write C programing and explain the steps to follow.

Step 4: Ask trainees to perform the task and monitor the procedures.

Step 5: Ask trainees to read key reading 5.4.4. in the Trainee's Manual

Points to Remember

Steps for Handling an Interrupt

1. **Interrupt Occurrence**: When an interrupt signal is generated, the CPU automatically stops its current task and saves the execution context (registers, program counter).
2. **Context Switching**: The CPU saves the current execution context to ensure the interrupted program can resume later.
3. **Execute the Interrupt Handler**: The registered interrupt handler (found in the IDT) executes, performing the necessary actions to handle the interrupt (e.g., reading data, processing signals).
4. **Restore Context**: After the handler completes, the CPU restores the saved context so the original program can continue seamlessly.
5. **Return from Interrupt**: The CPU uses a special instruction (e.g., iret in x86) to return from the interrupt handler, restoring the system state.

Practical Activity 5.4.5: Creating a generic Interrupt Handler

Notes to the trainer

- Avail of computer

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the topic and ask trainees to perform the following tasks:
As software developer, you are requested to go to the computer lab and write C Program that will create a generic Interrupt Handler

Step 2: Provide instruction that will be followed

Step 3: Demonstrate how to create a generic interrupt handler and Explain the steps to follow.

Step 4: Explain the task and monitor the procedures.

Step 5: Ask trainees to read key reading 5.4.5. in the Trainee's Manual

Points to Remember

Creating a Generic Interrupt Handler

- **Design the Handler Function**: The handler should accept parameters to identify the type of interrupt, typically using the interrupt vector number.
- **Define the IDT Entry**: Register the generic handler in the IDT so that it can handle multiple interrupt types based on the vector number.
- **Implement Context Saving and Restoring**: Ensure the handler saves and restores the CPU context properly, allowing the original task to resume.

Practical Activity 5.4.6: Applying Programmable Interrupt Controller (PIC)

Notes to the trainer

- Avail of computer

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the topic and ask trainees to perform the following tasks:

As software developer, you are requested to go to the computer lab and write C Program that will apply PIC in assembly file

Step 2: Provide instruction that will be followed

Step 3: Demonstrate how apply PIC in assembly file and Explain the steps to follow.

Step 4: Perform the task and monitor the procedures.

Step 5: Verify whether PIC are clearly applied

Step 6: Ask trainees to read key reading 5.4.2. in the Trainee's Manual

Points to Remember

Applying the Programmable Interrupt Controller (PIC)

- **Initialize the PIC**: Send initialization commands to set up the PIC's operation mode, vector offsets, and which interrupts (IRQs) to enable.
- **Set Up Interrupt Vectors**: Define the vector numbers for the PIC to route interrupts to the appropriate interrupt service routines (ISRs) in your IDT.
- **Enable Interrupts**: Enable the necessary interrupts on the PIC to allow it to start handling requests from devices.
- **Handle Interrupts**: Implement the logic in your ISRs to manage the specific tasks required when an interrupt occurs.

Application of learning 5.4.

Scenario

ABC Technologies Ltd, located in Kigali, specializes in embedded systems and real-time applications. They are developing a custom operating system for a new line of IoT devices that require efficient event handling. The current prototype lacks a robust interrupt handling mechanism, causing delays in response to critical events. As an OS Developer, you are tasked with implementing an effective interrupt handling system that will enable timely responses to hardware and software interrupts.

Task

1. Implement an interrupt handling system to manage hardware and software interrupts effectively.
2. Create entries in the Interrupt Descriptor Table (IDT) for each interrupt handler.
3. Develop a generic interrupt handler capable of managing various types of interrupts.
4. Configure the Programmable Interrupt Controller (PIC) to handle interrupt requests efficiently.

Instructions

1. Define the concept of interrupts and their role in system responsiveness.
2. Create an entry in the Interrupt Descriptor Table (IDT) for each interrupt.
3. Handle interrupts by implementing the logic required for each specific interrupt.

4. Write a generic interrupt handler that can accommodate different interrupt types.
5. Configure the PIC for interrupt prioritization and management.
6. Ensure the system can handle multiple simultaneous interrupts without failure.
7. Test the interrupt handling mechanism in both simulation and on physical hardware.

Checklist

S/N	Assessment criteria (Based on performance criteria)	Indicator/element to check	Observation	
			Yes	No
1.	**Working Environment is well prepared**	Tools and equipment are selected		
		Software tools are installed		
		Virtual environment is configured		
2.	**Interrupts handlers is well implemented**	Interrupt handlers are defined		
		Interrupt Descriptor Table (IDT) is create		
		Generic interrupt handler is implemented		
3.	**Programmable interrupt controller is well configured**	PIC is configured for interrupt management		
		Prioritization of interrupt is implemented		
		Interrupt controller is configured		

Indicative content 5.5: Implementing a Function to Display Console Output on the Screen on the Kernel

Duration: 5 hrs

Practical Activity 5.5.1: Console output (Hello World OS) on the screen in kernel

Notes to the trainer

- While delivering this content, you are required to write assembler program with as extension file to be compiled in binary format.

Key steps:

While delivering this activity, pass through the following steps:

step1 Introduce the topic and ask trainees to do the task below:
As a trainee from level 5 computer system and architecture, you are requested to open the computer then write an assembler program to display on screen Hello, World OS, this program will be interlinked with others in the kernel to display on console Hello, World OS, when kernel OS loaded from the computers' memory.

step2 Explain the task and provide clear guiding instructions.

step3 Demonstrate how to console out hello world OS on screen of kernel

step4 Ask trainees to perform the task provided in step 1

step5 Guide trainees to keep verifying and debugging in case they meet any code defects

step6 Directs trainees to copy the code from key reading 5.5.1. From trainee's manual

Points to Remember

- **Creating a "Hello, World OS**" operating system that prints "Hello, World OS" to the screen requires an understanding of low-level programming and the boot process of a computer.
- **Steps:**
  - Write the Assembly Code in vs code (ex:hello.asm)
  - Assemble the Code into a Bootable Binary in MSYS2 :

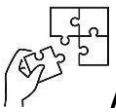
nasm -f bin -o hello.bin hello.asm

  - Run the Code in QEMU:

qemu-system-x86_64 -drive format=raw,file=hello.bin

  - If you have many files you can Link all object files into a single kernel binary.

i686-elf-ld -o kernel.bin -Ttext 0x1000 kernel.o screen.o bios.o --oformat binary



Application of learning 5.5.

As a trainee of level 5 computer system and architecture you have learnt basics in assembler programming language that can be used to interact with the bios handling interrupts and display Hello, World OS" on the screen when booted using mys2 and nasm. From the above mentioned task as a trainee from L5CSA you are obliged to create application using assembler programming language implementing console output of a message "**Hello, World OS**" on screen into a kernel of OS.

Checklist

S/N	Assessment criteria (Based on performance criteria)	Indicator/element to check	Observation	
			Yes	No
1.	**Assembly program is well written**	Assembly code in vs code is available		
		Section in assembly codes containing message Hello World OS is available		
2.	**Compilation is well enhanced**	Compilation is done		
		Bin file is produced in the directory		
3.	**Output is well provided as it should be**	Qemu system is used to run binary file		
		Hello World OS is displayed		

Indicative content 5.6: Implementing System Call

Duration: 5 hrs

Theoretical Activity 5.6.1: Description of system call

Notes to the trainer:

- While delivering this content, a small group can be used for describing system call

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the activity and ask trainees to answer the following questions:

I. What is system call?
II. Describe the types of system call in OS
III. Provide Examples of System Calls in Windows and Unix
IV. What are the Rules for Passing parameters to the System Calls?
V. What are different Services Provided by System Calls
VI. What are the features and its advantages of System Calls?

Step 2: Asks trainee to write answers provided on flip-chart/paper.

Step 3: Asks trainees to share the findings

Step 4: Provides expert view and addresses any questions or concerns if any.

Step 5: Directs the trainees to read the key reading 5.6.1 from trainee's manual.

Points to Remember

- **Implementing a System Call:** requires modifying both the kernel and user-space program to enable communication through a well-defined interface.
- **Types of System Calls in OS are**: Process control, File management, Device management, Information maintenance and Communication.
- **Services Provided by System Calls includes**: Memory management, Process management, Device handling, File system access and Inter-process communication (IPC).

Practical Activity 5.6.2: Develop C files for system call, modify files to integrate our system call, Recompile and reboot and then after test the system call

Notes to the trainer

- While delivering this content, you are required to write C source codes just implementing system call linked and recompiled into nasm in the binary file
- Avail video tutorials

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the topic and ask trainees to read the task below:

As a trainee from level 5 computer system and architecture, you are required to write C program files related to system call compile and reboot into kernel. In to implement our system call.

Step 2: Explain the task and provide clear guiding instructions.

Step 3: Demonstrate how to write C source code just implementing system call linked and recompiled into nasm in the binary file.

Step 4: Asks trainees to use provided video tutorial demonstrating them

Step 5: Guide trainees to keep verifying and debugging in case they meet any code defects

Step 6: Directs trainees to read the key reading 5.6.2. From trainee's manual.

Points to Remember

- Practical implementation of system call requires the following steps:

Step 1. Writing different C program files related to system call

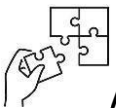
Step 2. Modifying Necessary Files to Integrate Our System Call

Step 3. Recompiling and Rebooting by the following commands:

```
nasm -f elf32 idt.asm -o idt.o
i686-elf-gcc -ffreestanding -c syscall.c -o syscall.o
i686-elf-gcc -ffreestanding -c kernel.c -o kernel.o
i686-elf-gcc -ffreestanding -c user_program.c -o user_program.o
i686-elf-ld -o kernel.bin -Ttext 0x1000 kernel.o syscall.o idt.o user_program.o
--oformat binary
```

Step 4. Run the Kernel in QEMU:

```
qemu-system-x86_64 -kernel kernel.bin
```



Application of learning 5.6.

As a trainee from level 5 computer system and architecture you have learnt different programming languages such C and C++ and in assembler, you acquired knowledge and skills. SIBOME LTD is a Tech company wanting to hire an expert developer who will write a program file in C recompiled in kernel.asm to binary format in lower-level programming which will integrate our system calls then after recompile and reboot using virtual machine.

CHECK LIST

S/N	Assessment criteria (Based on performance criteria)	Indicator/element to check	Observation	
			Yes	No
1.	**C-files for system calls are well developed**	C program files to implement system call are available in vs code		
		Modification to necessary files are achieved		
		Integration to our system call is done		
2.	**Recompilation and rebooting is well enhanced**	Recompilation in nasm is done		
		Rebooting to multiple files is done		
		Run the Kernel in QEMU is used		
3.	Testing system calls is well achieved	Boot up, initialize the kernel, and set up the system call is achieved		
		Message from user program is enhanced		

Indicative content 5.7: Implement Publication of our System

Duration: 5 hrs

Practical Activity 5.7.1: Implement Publication of Our System

Notes to the trainer

- While delivering this content, you are required to Build an .iso file from our system test it and Generate OS documentation

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the topic and ask trainees to do the task below:
As a trainee from level 5 computer system and architecture, you are hired to Set up a directory structure for ISO creation, build Script and this script will compile the kernel, copy it to the ISO directory, and create the ISO using grub-mkrescue.

Step 2: Explain the task and provide clear guiding instructions.

Step 3: Demonstrate how to build an iso file from our system test it and generate OS documentation

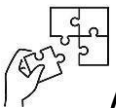
Step 4: Ask trainees to perform the task provided in step 1

Step 5: Guide trainees to keep verifying and debugging in case they meet any code defects

Step 6: Directs trainees to copy the code from key reading 5.7.1. From trainee's manual

Points to Remember

- Set up a directory structure for ISO creation
- Build Script (build_iso.sh)
- Test the ISO File
- Using VirtualBox
- Run the ISO on a Physical Machine (Optional)
- Generate OS Documentation: Write clear documentation, including an overview, system calls, and architecture



Application of learning 5.7.

As a trainee from level 5 computer system and architecture you have learnt different programming languages such C and C++ you acquired knowledge and skills on hobby kernel development using C.

SIBOME LTD is a Tech company wanting to hire an expert developer who will write same program files in C others in lower-level programming(assembly) which will implement the publication of our system of hobby kernel os. You are requested to build an .iso file from our system.

CHECKLIST

SN	Assessment criteria (Based on performance criteria)	Indicator/element to check	Observation	
			Yes	No
1.	**Directory Structure for ISO files well established**	Required files inside directory to generate ISO are organized		
		GRUB Configuration File is done		
1.	**Build Script are well written**	Script will compile the kernel, to create ISO and to copy it in the iso directory using grub-mkrescue is achieved		
		Testing ISO using qemu is achieved successfully		
2.	**Running Iso and generating documentation are well created**	Run the ISO on a Physical Machine is working		
		Documentation in structured ways is generated		

Learning outcome 5 end assessment

Theoretical assessment

I. Multiple-Choice Questions

Which file extension is typically used for assembly source files written in NASM?

a) .exe

b) .asm

c). o

d) .out

Answer: b) .asm

2.What is the primary role of the linker in compiling and running assembly code?

a) To convert assembly code to machine code

b) To link external libraries and produce an executable

c) To optimize assembly code for faster performance

d) To display runtime errors

Answer: b) To link external libraries and produce an executable

3.In NASM, which segment is used to declare initialized data?

a) .text

b) .data

c). bss

d). stack

Answer: b) .data

4.Which register is commonly used as the base pointer for the stack segment in x86 assembly?

a) eax

b) ebx

c) esp

d) ebp

Answer: d) ebp

5.What endianness does the x86 architecture follow?

a) Big-endian

b) Little-endian

c) Mid-endian

d) None of the above

Answer: b) Little-endian

6.Which register type is used to control the execution of code, such as EIP (Extended Instruction Pointer)?

a) General-purpose registers

b) Control registers

c) Segment registers

d) None of the above

Answer: b) Control registers

7.Which instruction is used to halt the CPU in an assembly program?

a) mov
b) add
c) jmp
d) hlt

Answer: d) hlt

8.What is the primary purpose of a bootloader?

a. To perform file management tasks
b. To load the operating system into memory
c. To run applications
d. To manage hardware settings

Answer: B) To load the operating system into memory

9.Which of the following languages is primarily used to write bootloaders?

a) Python
b) Assembly Language
c) Java
d) HTML

Answer: B) Assembly Language

10.Which tool can be used to create a bootable USB drive for testing a bootloader?

a) Microsoft Word
b) dd
c) Notepad
d) Excel

Answer: B) dd

II. Answer True or False on the following questions

a) A COM file can only be run on real hardware, not in virtual environments.
   Answer: False
b) Turbo Debugger allows users to set breakpoints and monitor variable states during execution.
   Answer: True
c) The bootloader is the first piece of code that runs when a computer is powered on.
   Answer: True

III. Describe the steps to test a bootloader in a virtual machine.

Answer:

Step 1: Create a new virtual machine in VMware.
Step 2: Configure settings (OS type, memory, etc.).
Step 3: Attach a bootable floppy image containing the bootloader.
Step 4: Start the VM and ensure it boots from the floppy drive.
Step 5: Observe the output for expected behavior.

IV. Explain the significance of using a Makefile in bootloader development**.**

Answer: A Makefile automates the build process, allowing for easy compilation of source files into a final binary. It ensures consistent builds, simplifies complex commands, and saves time during development.

V.Discuss the importance of debugging tools in the development of a bootloader and provide examples of how they can assist in troubleshooting.

Answer: Debugging tools are critical in bootloader development as they help identify and resolve issues in low-level code. Tools like Turbo Debugger allow developers to set breakpoints and inspect memory and registers, facilitating the tracking of execution flow and identification of logical errors. CodeView provides a user-friendly interface for monitoring variables and stepping through code, while D86 offers a simple command set for examining program behavior. These tools enable developers to ensure the reliability and functionality of the bootloader before deployment.

VI. Matching the tools with their description

Answer	Tool	Description
A: 3	A) Turbo Debugger	1. A straightforward DOS debugger for assembly code.
B: 2	B) CodeView	2. A powerful debugger with extensive features.
C: 1	C) D86	3. Allows you to set breakpoints and monitor execution.
D: 4	D) Bosch	4. Less common but effective for assembly debugging.
		5. is the first piece of code that runs when a computer is powered on

VII. Fill-in-the-Blank Questions:

1. In the provided Hello, World OS, the command nasm -f bin -o hello.bin hello.asm is used to assemble the code into a bootable _____.

   Binary

2. The scroll_screen function is triggered when the cursor reaches the _____ of the screen.

   Bottom

3. System calls help to enforce _____ and provide _____ access to system resources.

   Isolation, controlled

II. Answer True or False

4. The int 0x80 interrupt is used to implement system calls in the kernel.

   True

5. The fill_screen_with_color() function changes both text and background colors across the entire screen.

   True

6. The scroll_screen() function directly uses BIOS interrupt 0x10 to scroll the screen.

   False (It manually moves lines in the VGA buffer to scroll)

7. set_cursor_position uses a NASM function and BIOS interrupt to set the position of the cursor on the screen.

True

8. The system call dispatcher function (syscall_handler) processes each system call based on its ID and performs the requested action.

   True

Practical assessment

System software make computer operational so from here the development of hobby kernel operating system is crucial because the kernel is the core part of an operating system.

As a level5 computer system and architecture, you are part of a development team working on creating a basic operating system (OS) that includes essential system functions and handles system-level tasks efficiently. It requires applying both Assembly Language concepts and OS-level functionalities, particularly focusing on kernel and bootloader development, interrupt handling, system calls, and OS publication. You will use x86 Assembly Language and C programming, compiling and linking code using NASM and GCC, and creating an executable kernel. The goal is to demonstrate practical skills in low-level programming and understanding of system software.

You are going to develop a simple project performing the following vital tasks on kernel:

1. Write a "Hello, World OS" program in Assembly Language.
2. Write assembly program to apply various addressing modes, including register and memory addressing then after write a program that demonstrates the use of memory segments (data, code, stack).
3. Write a bootloader in Assembly that loads into memory, clears the screen, and outputs a welcome message.
4. Set up the Interrupt Descriptor Table (IDT) and configure the Programmable Interrupt Controller (PIC).
5. Use a linker script to merge Assembly and C object files into an executable kernel file and then test the kernel file by running it on a virtual machine.
6. Recompile, reboot, and test to ensure the system call functions as expected.
7. Debug any issues found during testing, document your processes, and prepare OS documentation.

Checklist

S/N	Assessment criteria (Based on performance criteria)	Indicator/element to check	Observation	
			Yes	No
1.	**Environment to write the program is well created**	Mys2, nasm and gcc installed		
		Vs code editor is configured into your project		
2.	**Writing the c files and assembly files in project directory are well done**	Hello, World OS Nasm file is created		
		C-files and nasm file related to memory addressing and memory segments is created		
		Bootloader in Assembly that loads into memory is created		
		Programmable Interrupt Controller (PIC) file is created		
		A linker script to merge Assembly and C object files into an executable kernel file is Created		
		C-files and assembly files are organized in directory		
3.	**Quality and expected output are well provided**	Testing the kernel file by running it on a qemu is achieved		
		Recompilation, rebooting, and testing to ensure the system calls are functioning as expected		
		Debugging and generating OS documentation is done.		

Further information to the trainer

Bartlett, J., & Bruno, D. (n.d.). *Programming from the Ground Up*. https://download-mirror.savannah.gnu.org/releases/pgubook/ProgrammingGroundUp-1-0-booksize.pdf

Love, R. (2020). *Linux Kernel Development, 3rd Edition | InformIT*. Informit.com. https://www.informit.com/store/linux-kernel-development-9780672329463

Operating Systems Design and Implementation: Tanenbaum, Andrew, Woodhull, Albert: 9780131429383: Amazon.com: Books. (2022). Amazon.com. https://www.amazon.com/Operating-Systems-Design-Implementation-3rd/dp/0131429388

Operating Systems: Three Easy Pieces. (n.d.). Pages.cs.wisc.edu. https://pages.cs.wisc.edu/~remzi/OSTEP/

The Art of Assembly Language, 2nd Edition: Hyde, Randall: 9781593272074: Amazon.com: Books. (n.d.). https://www.amazon.com/Art-Assembly-Language-2nd/dp/1593272073

RTB | RWANDA TVET BOARD

October 2024