Reg No : 22BCE3799
Name : Apurba Koirala
Slot : L39+L40

Exercise 5 – Page Faults, File Allocation and Virtualization

QN.1 Write a C program to implement page replacement algorithm and to calculate miss and hit ratio by considering FIFO, optimal and LRU schemes. Assume a minimum of 15 pages are need to be loaded while you can keep only 4 frames in the main memory.

AIM:

To implement a page replacement algorithm in C that includes FIFO, Optimal, and LRU techniques and calculates both the hit and miss ratios. Assume that a minimum of 15 pages will be loaded, and the main memory consists of 4 frames.

ALGORITHM:

1. **FIFO (First in First Out):**
    - Begin with empty frames.
    - For each page reference, check if it is already in a frame.
    - If not present, replace the oldest page (the one that was added first) and record a miss.
    - If it is present, record a hit.
2. **Optimal Page Replacement:**
    - Begin with empty frames.
    - For each page reference, check if it is already in a frame.
    - If not present, replace the page that will not be used for the longest time in the future and record a miss.
    - If it is present, record a hit.
3. **LRU (Least Recently Used):**
    - Begin with empty frames.
    - For each page reference, check if it is already in a frame.
    - If not present, replace the page that was least recently used and record a miss.
    - If it is present, update its usage as recent and record a hit.

Reg No : 22BCE3799
Name : Apurba Koirala
Slot : L39+L40

SOURCE CODE :

```
#include <stdio.h>

#include <stdlib.h>


#define FRAME_COUNT 4


void fifo_replacement(int *page_refs, int total_pages) {

   int memory_frames[FRAME_COUNT] = {-1}, insert_pos = 0, hit_count = 0, miss_count =
0;     for (int i = 0; i < total_pages; i++) {

int found = 0;         for (int j = 0; j <

FRAME_COUNT; j++) {             if

(memory_frames[j] == page_refs[i]) {

hit_count++;                found = 1;

break;

         }                      }                       if   (!found)   {

memory_frames[insert_pos]           =           page_refs[i];

insert_pos   =   (insert_pos   +   1)   %   FRAME_COUNT;

miss_count++;

       }

   }

   printf("FIFO - Page Hits: %d, Page Misses: %d, Hit Rate: %.2f, Miss Rate: %.2f\n",
hit_count, miss_count, (float)hit_count / total_pages, (float)miss_count / total_pages);

}
```

Reg No : 22BCE3799
Name : Apurba Koirala
Slot : L39+L40

```
void optimal_replacement(int *page_refs, int total_pages) {      int

memory_frames[FRAME_COUNT] = {-1}, hit_count = 0, miss_count = 0;

   for (int i = 0; i < total_pages; i++) {

      int found = 0, j;         for (j = 0; j <

FRAME_COUNT; j++) {             if

(memory_frames[j] == page_refs[i]) {

hit_count++;                 found = 1;

break;

         }       }       if (!found) {            int

farthest_use = -1, replace_index = -1;             for

(j = 0; j < FRAME_COUNT; j++) {

            int k;                 for (k = i + 1; k <

total_pages; k++) {                     if

(memory_frames[j] == page_refs[k]) {

if (k > farthest_use) {                          farthest_use

= k;                         replace_index = j;

                  }

break;

                 }

            }

            if (k == total_pages) {

replace_index           =           j;

break;

            }

         }

         memory_frames[replace_index == -1 ? 0 : replace_index] = page_refs[i];

miss_count++;
```

Reg No : 22BCE3799
Name : Apurba Koirala
Slot : L39+L40

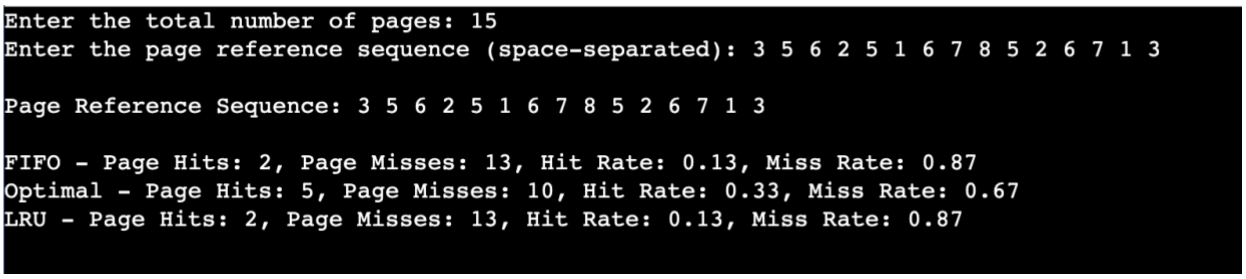
```
        }

    }
    printf("Optimal - Page Hits: %d, Page Misses: %d, Hit Rate: %.2f, Miss Rate: %.2f\n",
hit_count, miss_count, (float)hit_count / total_pages, (float)miss_count / total_pages);

}


void lru_replacement(int *page_refs, int total_pages) {      int

memory_frames[FRAME_COUNT] = {-1}, hit_count = 0, miss_count = 0;

int recent_usage[FRAME_COUNT] = {0};

    for (int i = 0; i < total_pages; i++) {          int

found = 0;          for (int j = 0; j <

FRAME_COUNT; j++) {              if

(memory_frames[j] == page_refs[i]) {

hit_count++;                  recent_usage[j] = i;

found = 1;                  break;

            }          }          if (!found) {              int least_recent =

0;              for (int j = 1; j < FRAME_COUNT; j++) {

if (recent_usage[j] < recent_usage[least_recent])

least_recent = j;

            }

            memory_frames[least_recent] = page_refs[i];

recent_usage[least_recent] = i;              miss_count++;

        }

    }
    printf("LRU - Page Hits: %d, Page Misses: %d, Hit Rate: %.2f, Miss Rate: %.2f\n",
hit_count, miss_count, (float)hit_count / total_pages, (float)miss_count / total_pages);

}
```

Reg No : 22BCE3799
Name : Apurba Koirala
Slot : L39+L40

```
int main() {     int
total_pages;
printf("Enter the
total number of
pages: ");
scanf("%d",
&total_pages);

   int page_refs[total_pages];        printf("Enter the page reference
sequence (space-separated): ");     for (int i = 0; i < total_pages; i++)
{        scanf("%d", &page_refs[i]);
   }

   printf("\nPage Reference Sequence: ");
for (int i = 0; i < total_pages; i++) {
printf("%d ", page_refs[i]);
   }
printf("\n\n");

   fifo_replacement(page_refs,         total_pages);
optimal_replacement(page_refs,      total_pages);
lru_replacement(page_refs, total_pages);

   return 0;
}
```

OUTPUT SCREEN SHOT:

Reg No : 22BCE3799
Name : Apurba Koirala
Slot : L39+L40



```
Enter the total number of pages: 15
Enter the page reference sequence (space-separated): 3 5 6 2 5 1 6 7 8 5 2 6 7 1 3

Page Reference Sequence: 3 5 6 2 5 1 6 7 8 5 2 6 7 1 3

FIFO - Page Hits: 2, Page Misses: 13, Hit Rate: 0.13, Miss Rate: 0.87
Optimal - Page Hits: 5, Page Misses: 10, Hit Rate: 0.33, Miss Rate: 0.67
LRU - Page Hits: 2, Page Misses: 13, Hit Rate: 0.13, Miss Rate: 0.87
```

Reg No : 22BCE3799
Name : Apurba Koirala
Slot : L39+L40

RESULTS:

FIFO - Page Hits: 2, Page Misses: 13, Hit Rate: 0.13, Miss Rate: 0.87
Optimal - Page Hits: 5, Page Misses: 10, Hit Rate: 0.33, Miss Rate: 0.67
LRU - Page Hits: 2, Page Misses: 13, Hit Rate: 0.13, Miss Rate: 0.87

Reg No : 22BCE3799
Name : Apurba Koirala
Slot : L39+L40

QN.2 Write a C program to demonstrate the below mentioned file allocation mechanisms

(i) Sequential

(ii) Indexed

(iii) Linked

AIM:

To implement file allocation techniques in C, including Sequential, Indexed, and Linked allocation methods, and demonstrate file allocation for each method based on user-provided input.

ALGORITHM:

Sequential Allocation:

- **Allocate consecutive disk blocks for a file.**
- **Input the starting block and file length from the user.**
- **Assign blocks sequentially, starting from the given block, for the specified file length.**
- **If any block in the sequence is already occupied, the allocation will fail.**

2. Indexed Allocation:

- **Use an index block to store the addresses of allocated blocks.**
- **Input the index block number and the number of required blocks from the user.**
- **Allocate the requested blocks and store their addresses in the index block.**
- **If any block is already in use, the allocation will fail.**

3. Linked Allocation:

- **Allocate blocks by linking each block to the next in the sequence.**
- **Input the starting block from the user.**
- **Assign blocks in a non-contiguous order, with each block containing a pointer to the next block.**
- **If any block is already in use, the allocation will fail.**

Reg No : 22BCE3799
Name : Apurba Koirala
Slot : L39+L40

SOURCE CODE :

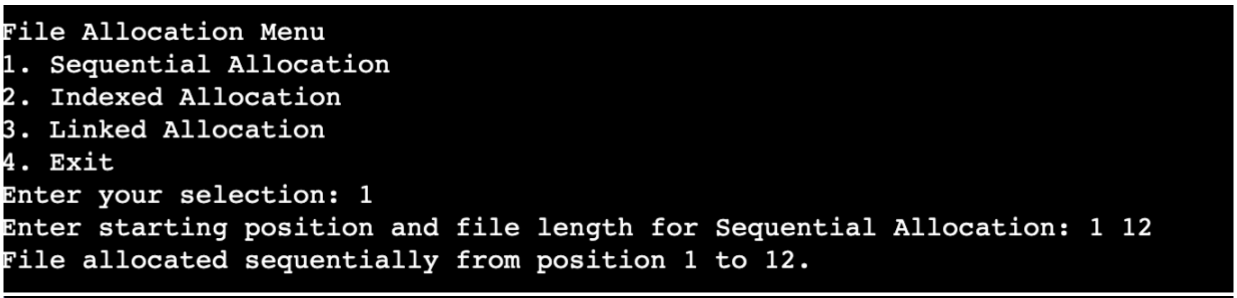
```
#include <stdio.h>
#define MAX_STORAGE 100

int storage[MAX_STORAGE] = {0};

void seq_allocation() {
    int start_pos, file_len;
    printf("Enter starting position and file length for Sequential Allocation: ");
    scanf("%d %d", &start_pos, &file_len);

    if (start_pos + file_len > MAX_STORAGE) {
        printf("Insufficient disk space.\n");
        return;
    }

    for (int i = start_pos; i < start_pos + file_len; i++) {
        if (storage[i] == 1) {
            printf("Position %d is already occupied.\n", i);
            return;
        }
    }

    for (int i = start_pos; i < start_pos + file_len; i++) {
        storage[i] = 1;
    }
```

Reg No : 22BCE3799
Name : Apurba Koirala
Slot : L39+L40

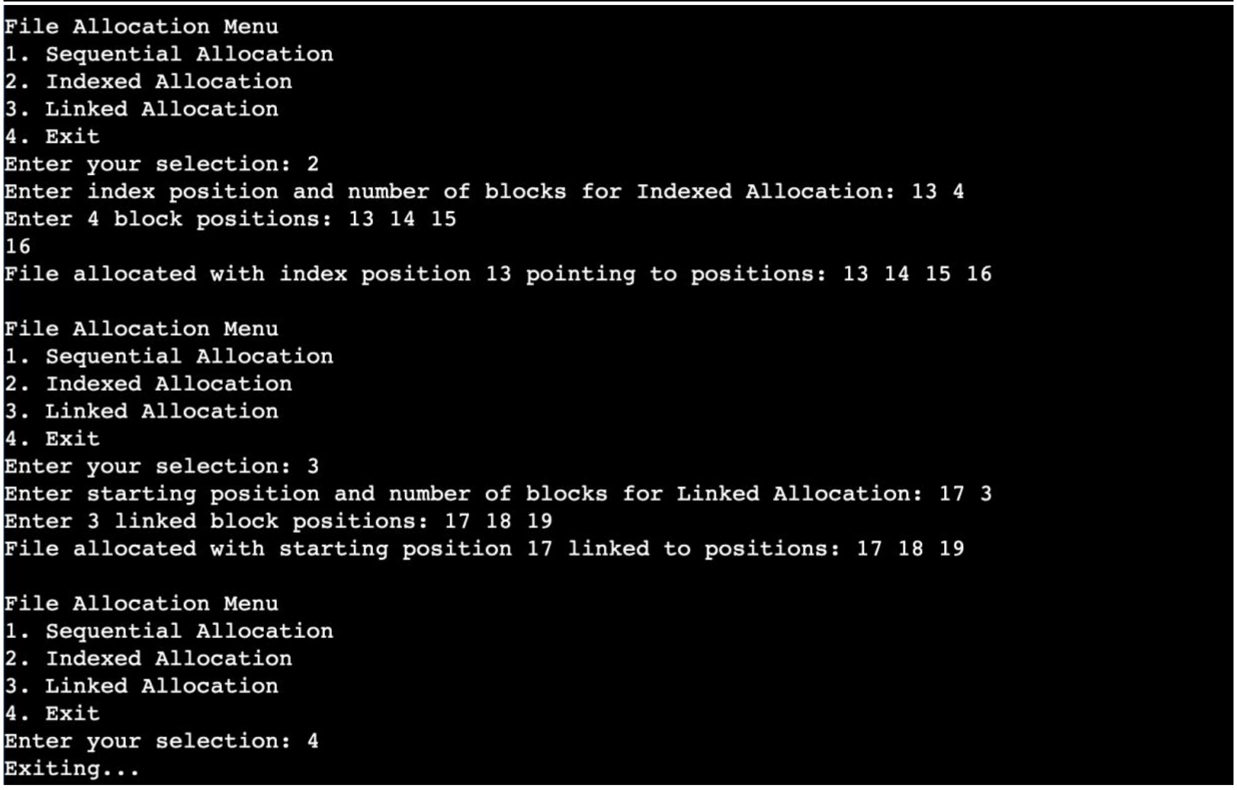
```
    printf("File allocated sequentially from position %d to %d.\n", start_pos, start_pos + file_len
- 1);
}


void idx_allocation() {
    int idx_pos, total_blocks;
    printf("Enter index position and number of blocks for Indexed Allocation: ");
    scanf("%d %d", &idx_pos, &total_blocks);

    if (idx_pos >= MAX_STORAGE || storage[idx_pos] == 1) {
        printf("Index position %d is already occupied or out of bounds.\n", idx_pos);
        return;
    }

    int block_list[total_blocks];
    printf("Enter %d block positions: ", total_blocks);
    for (int i = 0; i < total_blocks; i++) {
        scanf("%d", &block_list[i]);
        if (block_list[i] >= MAX_STORAGE || storage[block_list[i]] == 1) {
            printf("Position %d is already occupied or out of bounds.\n", block_list[i]);
            return;
        }
    }

    storage[idx_pos] = 1;
    for (int i = 0; i < total_blocks; i++) {
        storage[block_list[i]] = 1;
    }
    printf("File allocated with index position %d pointing to positions: ", idx_pos);
    for (int i = 0; i < total_blocks; i++) {
        printf("%d ", block_list[i]);
    }
    printf("\n");
}

void link_allocation() {
    int start_block, block_count;
    printf("Enter starting position and number of blocks for Linked Allocation: ");
    scanf("%d %d", &start_block, &block_count);

    if (start_block >= MAX_STORAGE || storage[start_block] == 1) {
        printf("Starting position %d is already occupied or out of bounds.\n", start_block);
        return;
    }

    int linked_blocks[block_count];
    printf("Enter %d linked block positions: ", block_count);
    for (int i = 0; i < block_count; i++) {
        scanf("%d", &linked_blocks[i]);
        if (linked_blocks[i] >= MAX_STORAGE || storage[linked_blocks[i]] == 1) {
            printf("Position %d is already occupied or out of bounds.\n", linked_blocks[i]);
            return;
        }
    }

    storage[start_block] = 1;
    for (int i = 0; i < block_count; i++) {
        storage[linked_blocks[i]] = 1;
    }

    printf("File allocated with starting position %d linked to positions: ", start_block);
    for (int i = 0; i < block_count; i++) {
        printf("%d ", linked_blocks[i]);
    }
    printf("\n");
}

int main() {
    int selection;
    while (1) {
        printf("\nFile Allocation Menu\n");
        printf("1. Sequential Allocation\n");
        printf("2. Indexed Allocation\n");
        printf("3. Linked Allocation\n");
        printf("4. Exit\n");
        printf("Enter your selection: ");
        scanf("%d", &selection);

        switch (selection) {
            case 1:
                seq_allocation();
                break;
            case 2:
                idx_allocation();
                break;
            case 3:
                link_allocation();
                break;
            case 4:
                printf("Exiting...\n");
                return 0;
            default:
                printf("Invalid selection, please try again.\n");
        }
    }
}
```

OUTPUT SCREEN SHOT:

Reg No : 22BCE3799
Name : Apurba Koirala
Slot : L39+L40



```
File Allocation Menu
1. Sequential Allocation
2. Indexed Allocation
3. Linked Allocation
4. Exit
Enter your selection: 1
Enter starting position and file length for Sequential Allocation: 1 12
File allocated sequentially from position 1 to 12.
```



```
File Allocation Menu
1. Sequential Allocation
2. Indexed Allocation
3. Linked Allocation
4. Exit
Enter your selection: 2
Enter index position and number of blocks for Indexed Allocation: 13 4
Enter 4 block positions: 13 14 15
16
File allocated with index position 13 pointing to positions: 13 14 15 16

File Allocation Menu
1. Sequential Allocation
2. Indexed Allocation
3. Linked Allocation
4. Exit
Enter your selection: 3
Enter starting position and number of blocks for Linked Allocation: 17 3
Enter 3 linked block positions: 17 18 19
File allocated with starting position 17 linked to positions: 17 18 19

File Allocation Menu
1. Sequential Allocation
2. Indexed Allocation
3. Linked Allocation
4. Exit
Enter your selection: 4
Exiting...
```

RESULTS:

The program effectively implements and showcases the three file allocation methods: Sequential, Indexed, and Linked.
Users can allocate blocks using any of these methods and are notified if any blocks are already in use.
It provides flexibility for users to define custom block allocations and illustrates how each method handles file storage on disk.

QN.3 Write a detailed study report on setup of Type1 and Type 2 hypervisors. (The setup study can be written in generic and a product specific)

Answer:

Reg No : 22BCE3799
Name : Apurba Koirala
Slot : L39+L40

Introduction

A hypervisor, also known as a Virtual Machine Monitor (VMM), is a software layer that enables the creation and management of virtual machines (VMs). There are two primary types of hypervisors: **Type 1** (Bare-metal) and **Type 2** (Hosted). This report outlines the generic and product-specific setup processes for both types of hypervisors.

Type 1 Hypervisors

Overview

Type 1 hypervisors run directly on the physical hardware without requiring a host operating system. These are typically used in enterprise environments for their efficiency and performance.

Generic Setup Process

1. **Pre-requisites:**
   - A server with virtualization support enabled in the BIOS/UEFI (e.g., Intel VT-x or AMD-V).
   - Compatible hardware (check the hypervisor's hardware compatibility list).
   - ISO file of the Type 1 hypervisor.
2. **Installation Steps:**
   - **Boot from Installation Media:** Burn the ISO file onto a USB drive or DVD and boot the server.
   - **Configure BIOS/UEFI Settings:** Enable virtualization features and set the boot priority to the installation media.
   - **Install the Hypervisor:** Follow the on-screen installation wizard to:
     - Partition the disks.
     - Configure network settings (static IP is preferred).
     - Set up administrative credentials.
   - **Post-Installation Configuration:**
     - Access the hypervisor's management interface via a web browser or dedicated client (e.g., vSphere Client for VMware ESXi).
     - Create storage pools, virtual networks, and virtual machines.
3. **Maintenance:**
   - Regularly update the hypervisor software for patches and new features.
   - Monitor resource usage and manage virtual machine lifecycle.

Product-Specific Example: VMware ESXi

1. **Requirements:**
   - Minimum 2 GHz CPU, 4 GB RAM, and supported NIC.
   - ESXi ISO image downloaded from VMware's official site.
2. **Installation Process:**
   - Boot from the ESXi installation media.
   - Follow the installation wizard to set up partitions and configure basic settings.
   - After installation, access the **VMware Host Client** via a browser using the server's IP address.
3. **Configuration:**
   - Add storage datastores.

- Set up networking (vSwitches).
- Deploy virtual machines using the ESXi web interface or vSphere Client.

Type 2 Hypervisors

Overview

Type 2 hypervisors run on top of a host operating system, making them ideal for development, testing, and personal use.

Generic Setup Process

1. **Pre-requisites:**
    - A host operating system (e.g., Windows, Linux, or macOS).
    - Type 2 hypervisor software (e.g., VirtualBox or VMware Workstation).
    - Sufficient hardware resources (RAM, CPU, and disk space).
2. **Installation Steps:**
    - **Download and Install the Hypervisor:**
        - Obtain the software installer from the official website.
        - Run the installer and follow the prompts to install the hypervisor on the host OS.
    - **Create Virtual Machines:**
        - Launch the hypervisor and create a new VM.
        - Allocate CPU, RAM, and disk space to the VM.
        - Attach the operating system ISO for the VM.
    - **Start and Configure the VM:**
        - Boot the VM and install the guest operating system.
        - Install hypervisor tools (e.g., VMware Tools or VirtualBox Guest Additions) for better integration and performance.
3. **Usage and Management:**
    - Start, stop, and snapshot VMs as needed.
    - Share folders and peripherals between the host and guest systems.

Product-Specific Example: Oracle VirtualBox

1. **Requirements:**
    - Host OS: Windows 10 or later, macOS, or Linux.
    - Minimum 4 GB RAM and 10 GB free disk space.
2. **Installation Process:**
    - Download VirtualBox from the official website.
    - Run the installer and configure network adapters (e.g., NAT or Bridged).
    - Install Extension Pack for additional features like USB 3.0 support.
3. **VM Setup:**
    - Create a VM and specify its settings (e.g., Linux or Windows OS).
    - Attach the ISO image for the guest OS and proceed with installation.
    - Install VirtualBox Guest Additions for improved performance and integration.

Comparison:

Type 1 and Type 2 hypervisors differ primarily in how they interact with the system's hardware and operating systems.

Reg No : 22BCE3799
Name : Apurba Koirala
Slot : L39+L40

Performance: Type 1 hypervisors, also known as bare-metal hypervisors, run directly on the hardware. This direct access to the physical resources results in high performance and efficiency. On the other hand, Type 2 hypervisors, or hosted hypervisors, run on top of an existing host operating system. This setup introduces some overhead due to the additional layer, which can impact performance and resource management.

Use Case: Type 1 hypervisors are typically used in enterprise environments and data centers, where the need for high performance, scalability, and reliability is crucial. They are ideal for running multiple virtual machines (VMs) on powerful servers. Type 2 hypervisors are more commonly used in personal or development environments, where ease of use and convenience are prioritized over raw performance. They are suitable for users who need to run virtualized environments on their personal computers without significant hardware modifications.

Setup Complexity: The installation and configuration of Type 1 hypervisors tend to be more complex, as they require direct installation on the hardware and often involve setting up the underlying infrastructure. In contrast, Type 2 hypervisors are easier to set up since they run within an existing operating system and do not require direct interaction with the hardware.

Examples: Examples of Type 1 hypervisors include VMware ESXi and Microsoft Hyper-V, which are typically used in large-scale, professional environments. On the other hand, Type 2 hypervisors include VMware Workstation and Oracle VirtualBox, which are commonly used for personal use or development on desktop systems.

Conclusion

Setting up Type 1 and Type 2 hypervisors requires understanding their respective environments and requirements. Type 1 hypervisors are ideal for enterprise environments due to their efficiency and scalability, while Type 2 hypervisors are suited for personal and development purposes due to their ease of setup. By following the outlined generic and product-specific steps, users can efficiently configure and utilize both types of hypervisors for their respective use cases.