COMSATS UNIVERSITY ISLAMABAD

Project Report
For
Process Manager with Virtual Memory Management

By

Hafsah Mehdi – SP22-BCT-016

Mohammad Fahad Malik – SP22-BCT-024

Supervisor

Mustafa Khattak

Table of Contents

Problem Statement

The current process management system operates without a dedicated memory management module, which leads to suboptimal memory utilization and performance bottlenecks. This project aims to bridge this gap by integrating a specialized memory management module into the system architecture. Without such a module, the system lacks the capability to effectively allocate and manage memory resources, resulting in heightened overhead and fragmentation concerns. Through the implementation of a dedicated memory manager, the system seeks to optimize memory utilization, boost performance, and enhance overall system stability and reliability.

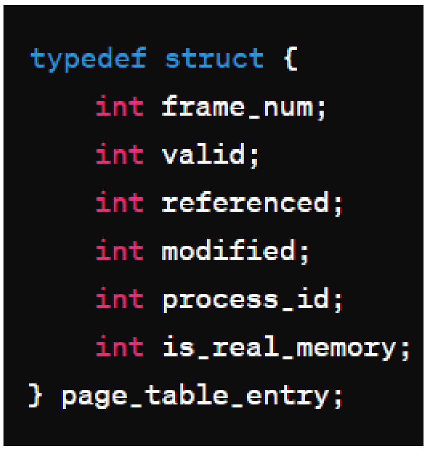
Abstract

This project presents a Process Manager with Virtual Memory Management capabilities. It allows users to create, delete, and manage processes within a virtual memory environment. Additionally, it provides functionalities to display process status, virtual memory status, real memory status, fragmentation details, and all active processes.

STRUCTS:

1. Page_Table_Entry:

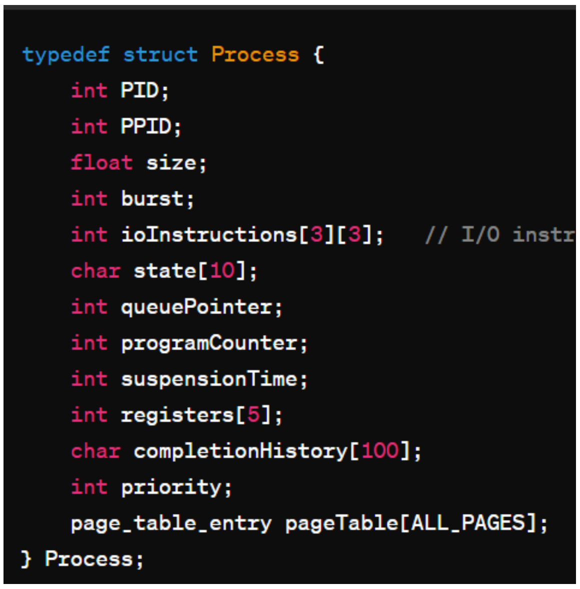
   Represents an entry in the page table, storing information about a specific frame, such as its validity, references, modification status, associated process ID, and whether it resides in real memory.



```c
typedef struct {
    int frame_num;
    int valid;
    int referenced;
    int modified;
    int process_id;
    int is_real_memory;
} page_table_entry;
```

2. Process:

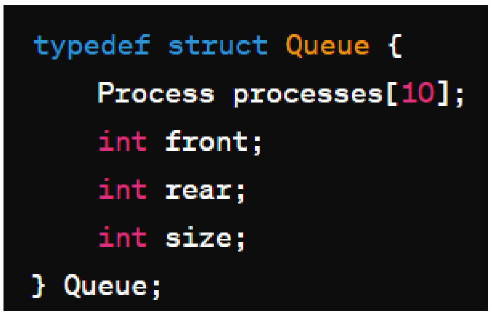
   Represents a process in the system, storing details such as Process ID (PID), Parent Process ID (PPID), size, burst time, I/O instructions, state, queue pointer, program counter, suspension time, registers, completion history, priority, and page table entries.



```c
typedef struct Process {
    int PID;
    int PPID;
    float size;
    int burst;
    int ioInstructions[3][3];    // I/O instr
    char state[10];
    int queuePointer;
    int programCounter;
    int suspensionTime;
    int registers[5];
    char completionHistory[100];
    int priority;
    page_table_entry pageTable[ALL_PAGES];
} Process;
```

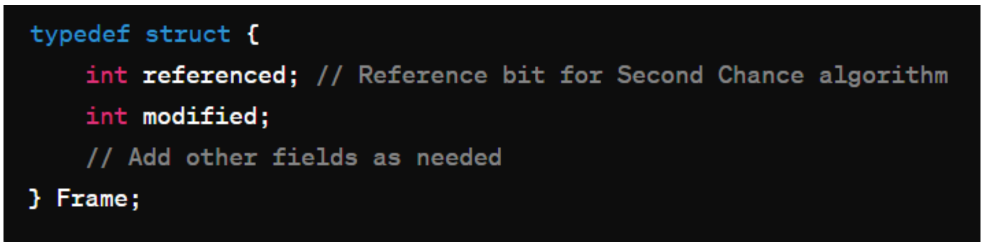
3. Queue:

   Represents a queue data structure to store processes. It contains an array of processes, along with front, rear, and size attributes to manage the queue.



```c
typedef struct Queue {
    Process processes[10];
    int front;
    int rear;
    int size;
} Queue;
```

4. Frame:
   Represents a frame in memory, containing fields for reference and modification bits. Additional fields can be added as needed.



```c
typedef struct {
    int referenced; // Reference bit for Second Chance algorithm
    int modified;
    // Add other fields as needed
} Frame;
```

FUNCTIONS:

initialize_memory:

This function is responsible for initializing the memory management system, including both real and virtual memory.

Functionality:

Initialize Real Memory:

- Loop Through Real Frames: It iterates through each frame in the real memory.
- Mark Frames as Free: It sets the corresponding entry in the real free frame table to 1, indicating that the frame is free and available for allocation.
- Initialize Reference Bit: It sets the reference bit of each frame to 0, which is used in the Second Chance algorithm for page replacement.
- Note: Other fields of the frame structure for real memory may also be initialized here if necessary, although they are not explicitly mentioned in the provided code.

Initialize Virtual Memory:

- Loop Through Virtual Frames: It iterates through each frame in the virtual memory.
- Mark Frames as Free: It sets the corresponding entry in the virtual free frame table to 1, indicating that the frame is free and available for allocation.
- Initialize Reference Bit: It sets the reference bit of each frame to 0, which is used in the Second Chance algorithm for page replacement.
- Note: Similar to real memory, other fields of the frame structure for virtual memory may also be initialized here if necessary, although they are not explicitly mentioned in the provided code.

allocate_frame Function:

This function is responsible for allocating a frame in either real or virtual memory for a process.

Parameters:

- is_virtual: A flag indicating whether to allocate a frame in virtual memory (1) or real memory (0).

Functionality:

- Determine Memory Type: It determines whether to allocate a frame in real or virtual memory based on the value of is_virtual.
- Check Availability: If there are free frames available in the specified memory type, it allocates the frame.
- For virtual memory, it checks if next_free_virtual_frame is less than NUM_VIRTUAL_FRAMES.
- For real memory, it checks if next_free_real_frame is less than NUM_REAL_FRAMES.
- Allocate Frame: If a free frame is available, it retrieves the frame number from the respective array of empty frame indexes (empty_indexes_in_virtual_memory or empty_indexes_in_real_memory), marks the frame as allocated in the free frame table, and returns the frame number.
- Victim Frame Selection: If no free frame is available, it applies the Second Chance algorithm to find a victim frame for replacement. It iterates through the frames, starting from next_frame_to_replace, and selects a frame with a reference bit of 0 (indicating it hasn't been recently accessed) for replacement. If all frames have a reference bit of 1, it resets the reference bit of each frame and continues the search.
- Return Value: If a free frame or a victim frame is found, its frame number is returned. Otherwise, it returns -1 to indicate that no free frame is available.

create_process Function:

This function is used to create a new process and allocate memory for it.

Parameters:

- RQ: Pointer to the Ready Queue where the new process will be added.
- PID: Process ID of the new process.
- PPID: Parent Process ID of the new process.
- size: Size of the new process in KB.
- burst: Burst time of the new process.

ioInstructions: Array containing I/O instructions for the new process.

priority: Priority of the new process.

Functionality:

- Check Ready Queue Capacity: It checks if the Ready Queue has space to accommodate a new process.
- Check Duplicate Process: It checks if a process with the same PID already exists in the Ready Queue.

- Calculate Number of Pages: It calculates the number of pages required for the process based on its size.
- Initialize New Process: It initializes the attributes of the new process such as PID, PPID, size, priority, burst time, and I/O instructions.
- Allocate Memory: It allocates memory for the process, first in real memory and then in virtual memory if necessary. It uses the allocate_frame function to allocate frames.
- Add Process to Ready Queue: Once memory allocation is successful, the new process is added to the Ready Queue.
- Return Value: It returns 1 if the process is successfully created and added to the Ready Queue. Otherwise, it returns 0 or 2 to indicate failure due to lack of space or duplicate process, respectively.

create_process

This function is responsible for creating a new process and adding it to the ready queue. Here's how it works:

Input Parameters:

- Queue *RQ: Pointer to the ready queue where the new process will be added.
- int PID: Process ID of the new process.
- int PPID: Parent Process ID of the new process.
- float size: Size of the process in kilobytes.
- int burst: Burst time of the process.
- int ioInstructions[3][3]: Array containing I/O instructions for the process.
- int priority: Priority of the process.

Functionality:

- Check Ready Queue Availability:

It checks if the ready queue has space to accommodate a new process. If the rear pointer incremented by 1 modulo the queue size equals the front pointer, it means the queue is full, and the function returns without creating a process, indicating that the process creation failed.

- Check Process Existence:

It checks if a process with the same PID already exists in the ready queue. If such a process exists, the function returns without creating a process, indicating that the process creation failed due to duplication.

- Calculate Number of Pages:

It calculates the number of pages required for the process based on its size and the page size.

- Initialize New Process:

It initializes a new Process structure with the provided parameters such as PID, PPID, size, burst, I/O instructions, and priority.

- Initialize Page Table:

It initializes the page table for the new process, marking all pages as invalid initially.

- Allocate Memory Frames:

It allocates memory frames for the process, both in real and virtual memory, based on the calculated number of pages.

For each page, it calls the allocate_frame() function to allocate a frame, updates the page table entry with the allocated frame number, and sets the page as valid.

- Add Process to Ready Queue:

It adds the newly created process to the ready queue at the rear position.

- Update Rear Pointer:

It updates the rear pointer of the ready queue to point to the next available position for adding a process.

- Return Status:

It returns a flag (isProcessCreated) indicating whether the process creation was successful (1) or failed (0).

delete_process:

This function is responsible for deleting a process from the ready queue based on its PID. Here's how it works:

Input Parameters:

Queue *RQ: Pointer to the ready queue from which the process will be deleted.

int PID: Process ID of the process to be deleted.

Functionality:

- Check Ready Queue Emptiness:

It first checks if the ready queue is empty by comparing the front and rear pointers. If they are equal, it means the queue is empty, and the function returns without further action.

- Find Process by PID:

It iterates through the ready queue to find the process with the specified PID.

If found, it proceeds with the deletion process; otherwise, it returns without any action.

- Delete Process:

Once the process with the given PID is found, it removes it by shifting elements in the queue.

It starts removing the process from the position where it was found (i), shifting subsequent processes to fill the gap.

- Free Allocated Memory:

It deallocates memory that was previously allocated to the process.

It iterates through the page table of the process to identify valid pages and frees the corresponding memory frames.

For each valid page, it determines whether the frame belongs to real or virtual memory and frees it accordingly.

It updates the free frame tables and the next available frame pointers accordingly.

Additionally, it prints the total number of real memory frames freed during the process deletion.

- Update Rear Pointer:

After removing the process, it updates the rear pointer of the ready queue to reflect the new size of the queue.

- Return Value:

It returns a flag (isProcessDeleted) indicating whether the process deletion was successful (1) or failed (0).

display_virtual_memory_status:

This function is responsible for displaying the current status of virtual memory. Here's how it works:

Functionality:

- Print Information:

It prints out the heading "Virtual Memory Status:" to indicate that the following information pertains to virtual memory.

- Print Total Frames:

It prints the total number of frames in virtual memory, which is defined by the constant NUM_VIRTUAL_FRAMES.

- Print Free Frames:

It calculates the number of free frames in virtual memory by subtracting the value of next_free_virtual_frame from the total number of frames.

next_free_virtual_frame keeps track of the index of the next available free frame in virtual memory.

- Print Occupied Frames:

It prints the number of occupied frames in virtual memory, which is equal to next_free_virtual_frame.

This indicates how many frames are currently in use or allocated in virtual memory.

- Output:

The function prints the total frames, free frames, and occupied frames of virtual memory, providing an overview of its current status.

display_real_memory_status:

Functionality:

- Print Information:

It prints out the heading "Real Memory Status:" to indicate that the following information pertains to real memory.

- Print Total Frames:

It prints the total number of frames in real memory, defined by the constant NUM_REAL_FRAMES.

- Print Free Frames:

It calculates the number of free frames in real memory by subtracting the value of next_free_real_frame from the total number of frames.

next_free_real_frame keeps track of the index of the next available free frame in real memory.

- Print Occupied Frames:

It prints the number of occupied frames in real memory, which is equal to next_free_real_frame.

This indicates how many frames are currently in use or allocated in real memory.

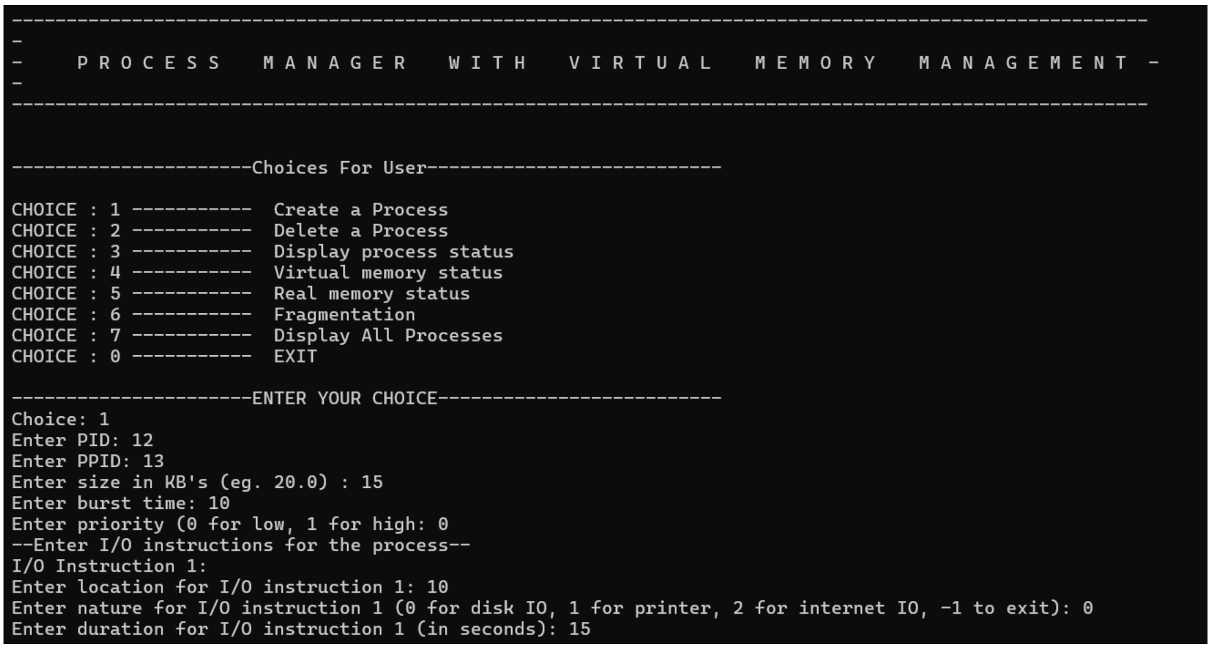
- Output:

Similar to display_virtual_memory_status(), this function prints the total frames, free frames, and occupied frames of real memory, providing an overview of its current status.

These functions collectively manage the memory and process creation/deletion within the system. The Second Chance algorithm is utilized in the allocate_frame function to select a victim frame for replacement when needed. This algorithm prevents thrashing by giving pages a "second chance" before being replaced, based on their reference bit.

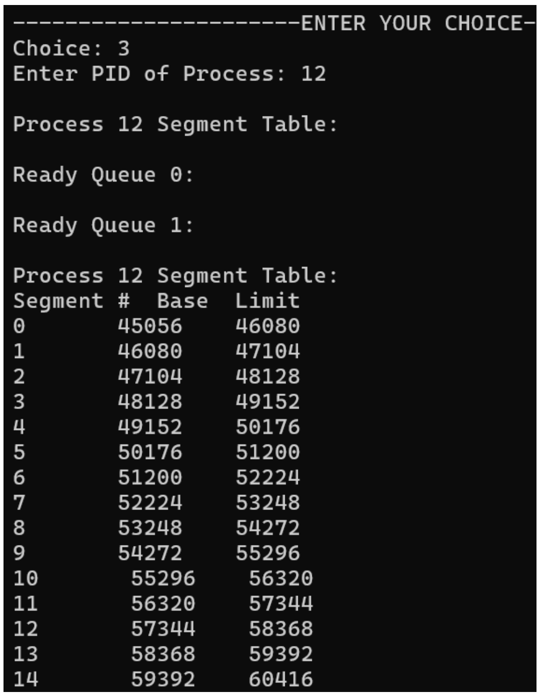
SCREENSHOTS:

CREATING A PROCESS:

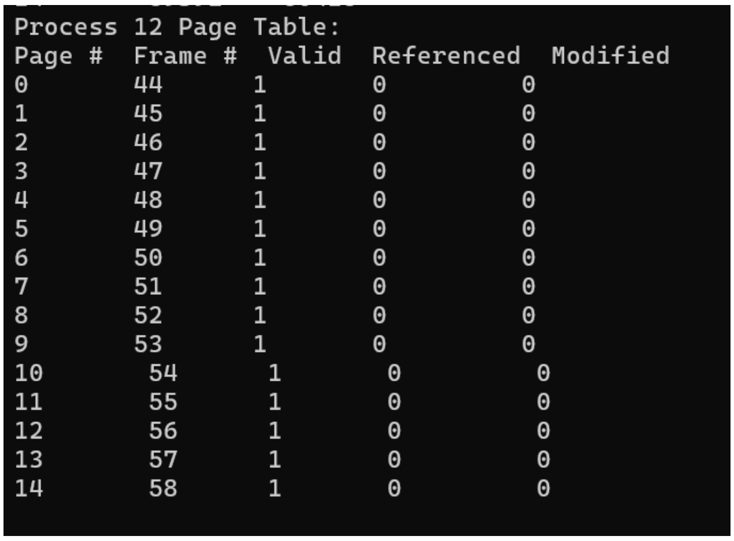


```
----------------------------------------------------------------------------------------------------
-
-    P R O C E S S   M A N A G E R   W I T H   V I R T U A L   M E M O R Y   M A N A G E M E N T -
-
----------------------------------------------------------------------------------------------------



---------------------Choices For User--------------------------

CHOICE : 1 -----------  Create a Process
CHOICE : 2 -----------  Delete a Process
CHOICE : 3 -----------  Display process status
CHOICE : 4 -----------  Virtual memory status
CHOICE : 5 -----------  Real memory status
CHOICE : 6 -----------  Fragmentation
CHOICE : 7 -----------  Display All Processes
CHOICE : 0 -----------  EXIT

---------------------ENTER YOUR CHOICE-------------------------
Choice: 1
Enter PID: 12
Enter PPID: 13
Enter size in KB's (eg. 20.0) : 15
Enter burst time: 10
Enter priority (0 for low, 1 for high: 0
--Enter I/O instructions for the process--
I/O Instruction 1:
Enter location for I/O instruction 1: 10
Enter nature for I/O instruction 1 (0 for disk IO, 1 for printer, 2 for internet IO, -1 to exit): 0
Enter duration for I/O instruction 1 (in seconds): 15
```

Process is stored in ready queue 1, so viewing the process status:

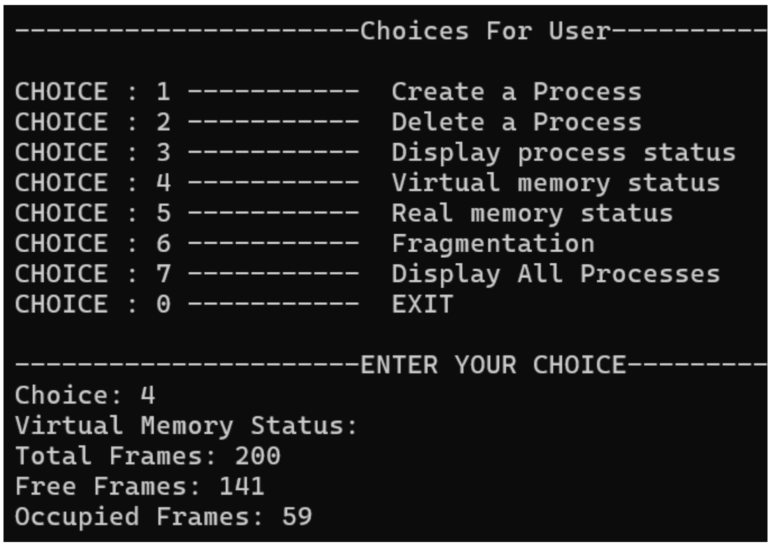


```
----------------------ENTER YOUR CHOICE-
Choice: 3
Enter PID of Process: 12

Process 12 Segment Table:

Ready Queue 0:

Ready Queue 1:

Process 12 Segment Table:
Segment #  Base  Limit
0       45056    46080
1       46080    47104
2       47104    48128
3       48128    49152
4       49152    50176
5       50176    51200
6       51200    52224
7       52224    53248
8       53248    54272
9       54272    55296
10       55296    56320
11       56320    57344
12       57344    58368
13       58368    59392
14       59392    60416
```



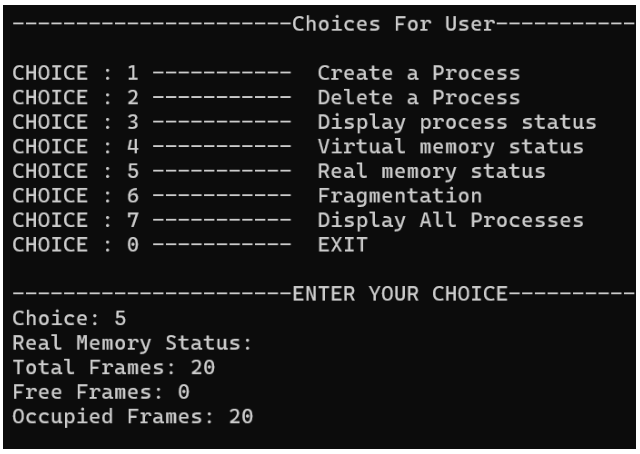
```
Process 12 Page Table:
Page #  Frame #  Valid  Referenced  Modified
0       44      1       0          0
1       45      1       0          0
2       46      1       0          0
3       47      1       0          0
4       48      1       0          0
5       49      1       0          0
6       50      1       0          0
7       51      1       0          0
8       52      1       0          0
9       53      1       0          0
10       54      1       0          0
11       55      1       0          0
12       56      1       0          0
13       57      1       0          0
14       58      1       0          0
```

Viewing virtual memory after process allocation:



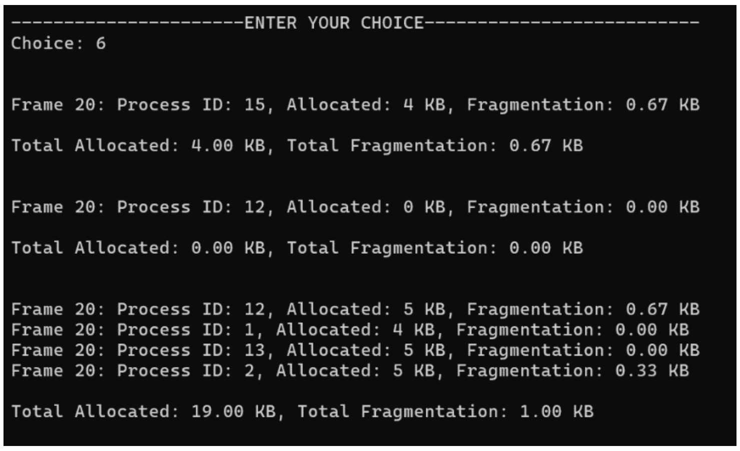
```
-----------------------Choices For User----------

CHOICE : 1 -----------  Create a Process
CHOICE : 2 -----------  Delete a Process
CHOICE : 3 -----------  Display process status
CHOICE : 4 -----------  Virtual memory status
CHOICE : 5 -----------  Real memory status
CHOICE : 6 -----------  Fragmentation
CHOICE : 7 -----------  Display All Processes
CHOICE : 0 -----------  EXIT

----------------------ENTER YOUR CHOICE---------
Choice: 4
Virtual Memory Status:
Total Frames: 200
Free Frames: 141
Occupied Frames: 59
```

All frames of real memory are occupied, so real memory will be:



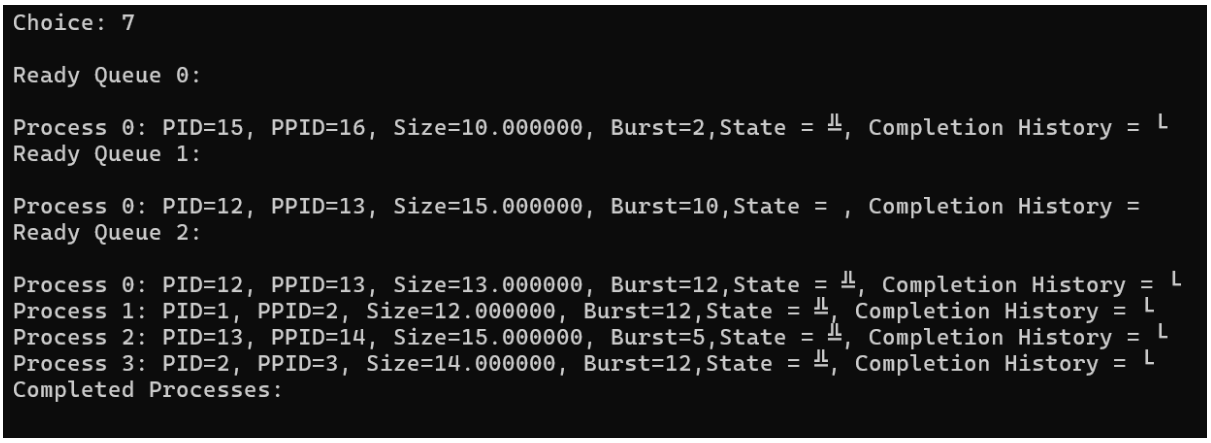
```
----------------------Choices For User-----------

CHOICE : 1 -----------  Create a Process
CHOICE : 2 -----------  Delete a Process
CHOICE : 3 -----------  Display process status
CHOICE : 4 -----------  Virtual memory status
CHOICE : 5 -----------  Real memory status
CHOICE : 6 -----------  Fragmentation
CHOICE : 7 -----------  Display All Processes
CHOICE : 0 -----------  EXIT

---------------------ENTER YOUR CHOICE----------
Choice: 5
Real Memory Status:
Total Frames: 20
Free Frames: 0
Occupied Frames: 20
```

Viewing all processes and fragmentation after complete allocation of real memory:



```
----------------------ENTER YOUR CHOICE-------------------------
Choice: 6


Frame 20: Process ID: 15, Allocated: 4 KB, Fragmentation: 0.67 KB

Total Allocated: 4.00 KB, Total Fragmentation: 0.67 KB


Frame 20: Process ID: 12, Allocated: 0 KB, Fragmentation: 0.00 KB

Total Allocated: 0.00 KB, Total Fragmentation: 0.00 KB


Frame 20: Process ID: 12, Allocated: 5 KB, Fragmentation: 0.67 KB
Frame 20: Process ID: 1, Allocated: 4 KB, Fragmentation: 0.00 KB
Frame 20: Process ID: 13, Allocated: 5 KB, Fragmentation: 0.00 KB
Frame 20: Process ID: 2, Allocated: 5 KB, Fragmentation: 0.33 KB

Total Allocated: 19.00 KB, Total Fragmentation: 1.00 KB
```

Viewing all processes with completion history:



```
Choice: 7

Ready Queue 0:

Process 0: PID=15, PPID=16, Size=10.000000, Burst=2,State = ╨, Completion History = └
Ready Queue 1:

Process 0: PID=12, PPID=13, Size=15.000000, Burst=10,State = , Completion History =
Ready Queue 2:

Process 0: PID=12, PPID=13, Size=13.000000, Burst=12,State = ╨, Completion History = └
Process 1: PID=1, PPID=2, Size=12.000000, Burst=12,State = ╨, Completion History = └
Process 2: PID=13, PPID=14, Size=15.000000, Burst=5,State = ╨, Completion History = └
Process 3: PID=2, PPID=3, Size=14.000000, Burst=12,State = ╨, Completion History = └
Completed Processes:
```

CONCLUSION:

In conclusion, the integration of a specialized memory management module into the existing process management system marks a significant step forward in optimizing system performance and resource utilization. By addressing the gap in memory management, this project aims to enhance the efficiency of memory allocation and management, leading to improved system stability, reliability, and overall performance. With the introduction of dedicated memory management capabilities, the system is better equipped to handle memory resources, mitigate fragmentation issues, and reduce overhead, ultimately resulting in a more robust and responsive computing environment. This project lays the foundation for a more efficient and effective system architecture, poised to meet the evolving demands of modern computing environments.