Green University of Bangladesh
Department of Computer Science and Engineering(CSE)

Faculty of Sciences and Engineering

Semester: (Spring , Year:2024), B.Sc. in CSE (Day)

LAB REPORT NO #04

Course Title: Operating System Lab

Course Code: CSE - 310 Section: 213_D5

Lab Experiment Name: Simulating the MFT and MVT Memory Management Techniques.

Student Details

Name	ID
MD Dulal Hossain	213902116

Lab Date : 15 - 05 - 2024
Submission Date : 22 - 05 - 2024

Course Teacher's Name : Md. Solaiman Mia

[For Teachers use only: Don't Write Anything inside this box]

Lab Report Status

Marks: ………………………………… Signature:....................
Comments:.. Date:.............................

Task :

Title :

Implement a code to solve the Memory Management technique problem?

Input:

Enter the number of Blocks– 4

Block 1 size: 280

Block 2 size: 350

Block 3 size: 300

Block 4 size: 320

Enter the number of processes – 4

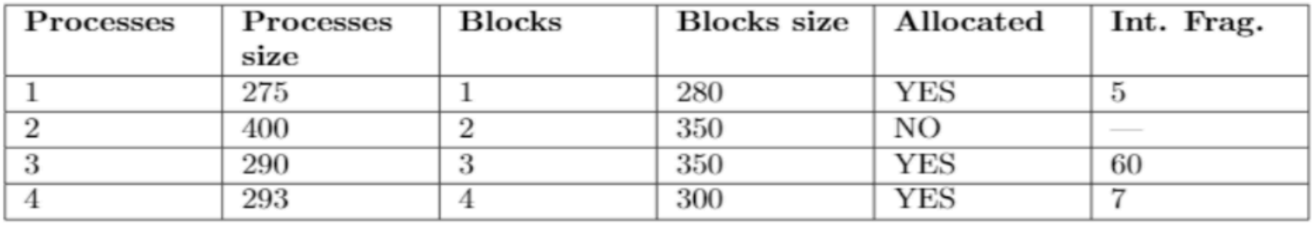
Enter memory required for process 1 – 275

Enter memory required for process 2 – 400

Enter memory required for process 3 – 290

Enter memory required for process 4 – 293

Table 1: **Sample Output**



Processes	Processes size	Blocks	Blocks size	Allocated	Int. Frag.
1	275	1	280	YES	5
2	400	2	350	NO	—
3	290	3	350	YES	60
4	293	4	300	YES	7

Algorithms :

1. The script begins by prompting the user to input the number of memory blocks (num_blocks).
2. It initializes two arrays: blocks (to store block sizes) and block_status (to track whether a block is allocated or not). The variable unused_blocks is set to the initial value of num_blocks.
3. The user is then asked to input the size of each block individually.
4. Next, the script prompts the user to input the number of processes (num_processes).
5. Similar to the blocks, it initializes arrays for processes: processes, process_status, and internal_fragmentation.
6. The script then collects memory requirements for each process from the user.
7. The main part of the script starts with printing a header for the output table: “Processes,” “Processes Size,” “Blocks,” “Blocks Size,” “Allocated,” and “Int. Frag.”
8. For each process, it attempts to allocate it to the first available block (using the First Fit algorithm):

    If the process size is less than or equal to the block size and the block is not already allocated, the process is allocated to that block.

    The internal fragmentation (unused space within the allocated block) is calculated.

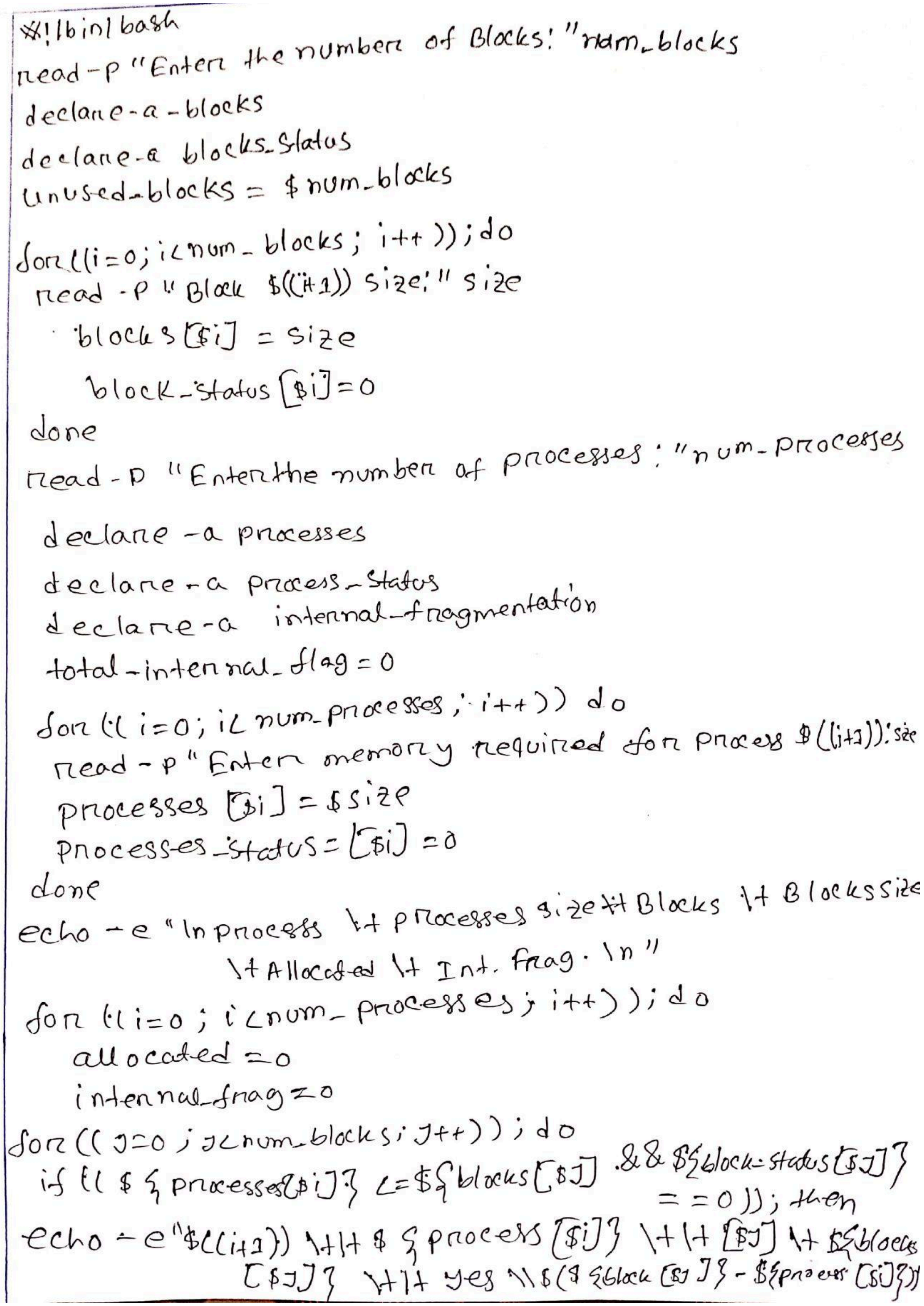
    The block_status is updated to indicate that the block is now allocated.

    The total internal fragmentation is updated.

    The number of unused blocks is decremented.

9. If a process cannot be allocated to any block, it is marked as “NO” in the output.
10. The script prints the total internal fragmentation and the number of unused blocks.
11. If there are unused blocks, it lists their sizes.

Source Code in Hand :



```
#!/bin/bash
read-p "Enter the number of Blocks: "num_blocks
declare-a -blocks
declare-a blocks_Status
unused-blocks = $num_blocks

for((i=0;i<num_blocks; i++));do
 read -p "Block $((i+1)) size:" size
   blocks[$i] = size
    block-status[$i]=0
 done
 read - p "Enter the number of processes:"num-processes

  declare -a processes
  declare-a process_Status
  declare-a  internal_fragmentation
  total-internal_flag = 0
 for ((i=0; i<num_processes;· i++)) do
  read - p "Enter memory required for process $((i+1)):size
  processes [$i] = $size
  processes_status= [$i] =0
 done
echo -e "\nprocess \t processes size\t Blocks \t Blocks size
               \t Allocated \t Int. frag. \n"
for ((i=0 ; i<num_processes; i++));do
   allocated =0
   internal_frag =0
for (( j=0 ; j<num_blocks; j++)) ; do
 if (( ${processes[$i]} <=${blocks[$j] && ${block-status[$j]}
                                          = =0 )); then
echo -e"$((i+1)) \t\t ${process [$i]} \t\t [$j] \t ${blocks
               [$j]} \t\t yes \\$(${block [$j]} - ${process [$i]})
```

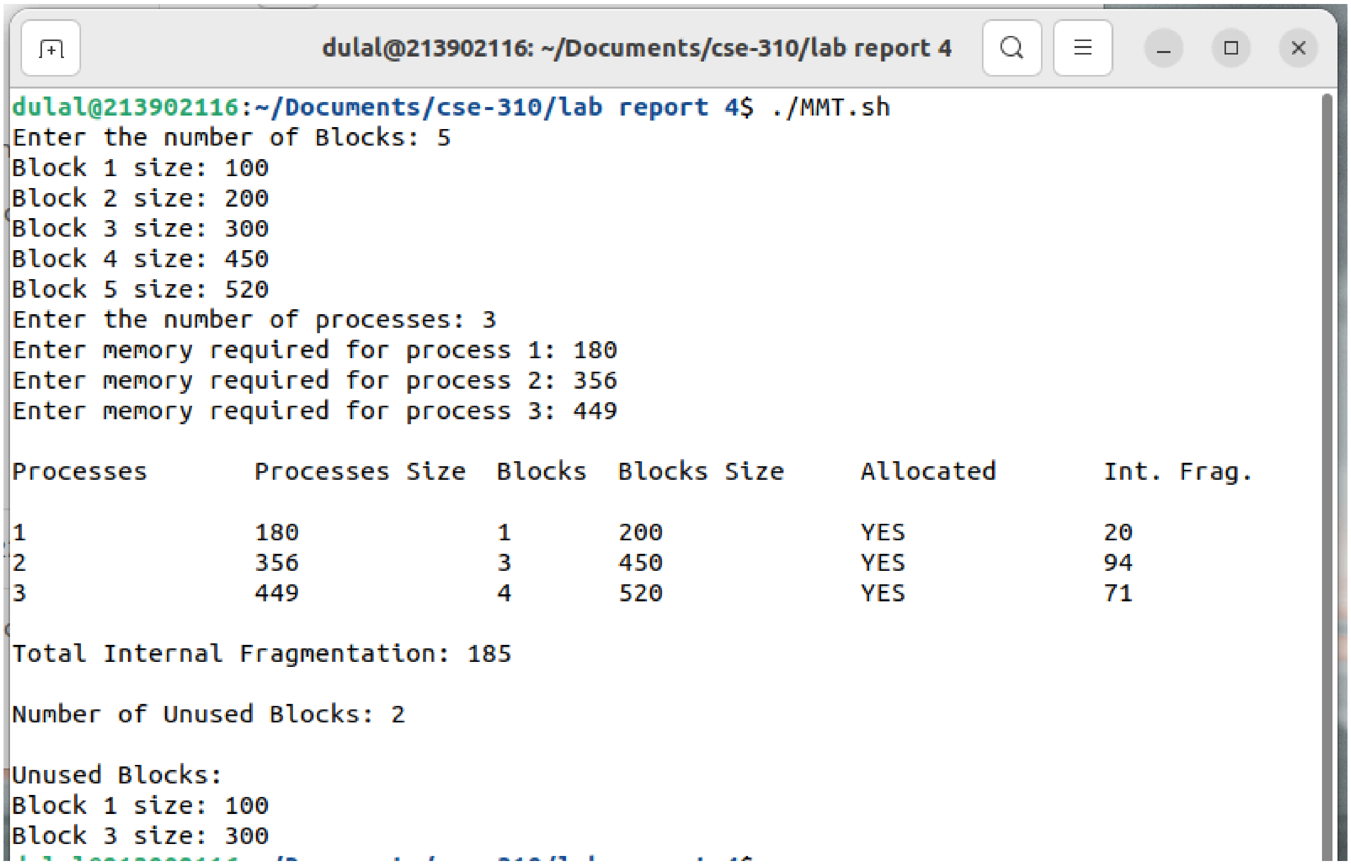
```
        block-status[$J] =1
       allocated =1
    internal-frag = $(( ${blocks[$J]} - ${process[$i]} ))
    ((total-internal-frag += internal_frag))
     ((unused-blocks--))
         break
    fi
done
       if ((allocated ==0 )); then
       echo -e "$ ((i+1)) \t\t $ {process[$i]} \t- \t- \tNO\t-"
       if
 internal_fragmantation[$i]=$internal_frag
done
 echo -e "\nTotal internal Fragmantation: $ total_internal_frag\n"
echo - e "number of unused Blocks: $ unused-blocks"
if ((unused_blocks >0)); then
      echo -e "\nunused Blocks:"
   for ((i=0; i< num-blocks; i++ ));do
        if (( ${block status[$i]} ==0 )); then
         echo -e "Block$((i+1)) size: ${blocks[$i]}"
         if
      done
 fi
exit ().
```

Source Code in write :

```
#!/bin/bash

read -p "Enter the number of Blocks: " num_blocks
declare -a blocks
declare -a block_status
unused_blocks=$num_blocks

for (( i=0; i<num_blocks; i++ )); do
   read -p "Block $((i+1)) size: " size
   blocks[$i]=$size
   block_status[$i]=0
done
read -p "Enter the number of processes: " num_processes
declare -a processes
declare -a process_status
declare -a internal_fragmentation
total_internal_frag=0

for (( i=0; i<num_processes; i++ )); do
   read -p "Enter memory required for process $((i+1)): " size
   processes[$i]=$size
   process_status[$i]=0
done
echo -e "\nProcesses \tProcesses Size \tBlocks \tBlocks Size \tAllocated \tInt. Frag.\n"

for (( i=0; i<num_processes; i++ )); do
   allocated=0
   internal_frag=0

   for (( j=0; j<num_blocks; j++ )); do
      if (( ${processes[$i]} <= ${blocks[$j]} && ${block_status[$j]} == 0 )); then
         echo -e "$((i+1)) \t\t${processes[$i]} \t\t$j \t${blocks[$j]} \t\tYES \t\t$(( ${blocks[$j]} -
${processes[$i]} ))"
         block_status[$j]=1
         allocated=1
         internal_frag=$(( ${blocks[$j]} - ${processes[$i]} ))
         ((total_internal_frag += internal_frag))
         ((unused_blocks--))
         break
      fi
```

```
    done
    if (( allocated == 0 )); then
        echo -e "$((i+1)) \t\t${processes[$i]} \t- \t- \tNO \t-"
    fi
    internal_fragmentation[$i]=$internal_frag
done
echo -e "\nTotal Internal Fragmentation: $total_internal_frag\n"
echo -e "Number of Unused Blocks: $unused_blocks"
if (( unused_blocks > 0 )); then
    echo -e "\nUnused Blocks:"

    for (( i=0; i<num_blocks; i++ )); do
        if (( ${block_status[$i]} == 0 )); then
            echo -e "Block $((i+1)) size: ${blocks[$i]}"
        fi
    done
fi
exit 0
```

Output :



```
dulal@213902116:~/Documents/cse-310/lab report 4$ ./MMT.sh
Enter the number of Blocks: 5
Block 1 size: 100
Block 2 size: 200
Block 3 size: 300
Block 4 size: 450
Block 5 size: 520
Enter the number of processes: 3
Enter memory required for process 1: 180
Enter memory required for process 2: 356
Enter memory required for process 3: 449

Processes       Processes Size  Blocks  Blocks Size     Allocated       Int. Frag.

1               180             1       200             YES             20
2               356             3       450             YES             94
3               449             4       520             YES             71

Total Internal Fragmentation: 185

Number of Unused Blocks: 2

Unused Blocks:
Block 1 size: 100
Block 3 size: 300
```

Figure 1.1 : Output in show Successfully.

Explain Output :

Processes : Process number (1-based index).
Processes Size : Size of the process.
Blocks : Block number (0-based index) where the process is allocated (if allocated).
Blocks Size : Size of the block (if allocated).
Allocated : Whether the process is allocated to a block (YES/NO).
Int. Frag. : Internal fragmentation (unused space within the allocated block).

For example:
Block 1 size : 100
Block 2 size : 200
Block 3 size : 300
Block 4 size : 450
Block 5 size : 520
Process 1 with size 180 is allocated to Block 2 with size 200; internal fragmentation is 20.
Process 2 with size 356 is allocated to Block 5 with size 450; internal fragmentation is 94.
Process 3 with size 449 is allocated to Block 4 with size 520; internal fragmentation is 71.

Total Internal Fragmentation : 20 + 94 + 71 = 185

Number of Unused Blocks ; 2

Unused Blocks:
Block 1 size: 100
Block 3 size: 300