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

Faculty of Sciences and Engineering
Semester: (Spring, Year: 2021), B.Sc. in CSE (Day)

Course Title: Operating System Lab
Course Code: CSE 310 Section: 191 DI

Lab Project Name: Essential Bash Algorithms.

Student Details

	Name	ID
1.	Umme Loara	191002187
2.	Zeneya Sharmin	191002228
3.	Mst. Mahmuda Akter Sumi	191002308

Submission Date	**: 14-09-2021**
Course Teacher's Name	**: Md. Jahidul Islam**

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

Lab Project Status

Marks: …………………………………… **Signature:** ....

Comments: .... **Date:** ....

Table of Contents

Chapter 1

Introduction

1.1 Introduction

A shell script is a computer program designed to be run by a Unix shell, a command-line interpreter. Different dialects of the shell script are considered scripting languages. Common tasks performed by shell scripts include file manipulation, program execution, and printing text. Shell scripts are mostly used to avoid repetitive tasks.

We can write a script to run a set of instructions automatically, instead of typing in one n command after another. Some examples of application shell scripts can be used:

The code compilation process is automated. Running a program or creating a program environment. Shell scripting is used for multiple system level functions such as 'system administration', 'for creating, maintaining and executing system boot scripts', 'automating tedious repetitive tasks', 'setting and performing system tasks', , ‘for automating the installation process for new software or for new software updates across the organization’, ‘for scheduling data backup process’, ‘etc. It is the most user friendly programming language for anything and everything related system level operations.

1.2 Design Objective

- To know about the uses of loop (for, while do), conditions (if, else, switch) in Shell Scripting Language.
- To enrich our knowledge in Shell Scripting Language.
- To develop the idea of function.
- To gather knowledge of memory management techniques, CPU Scheduling Algorithm, contagious Memory Allocation Technique

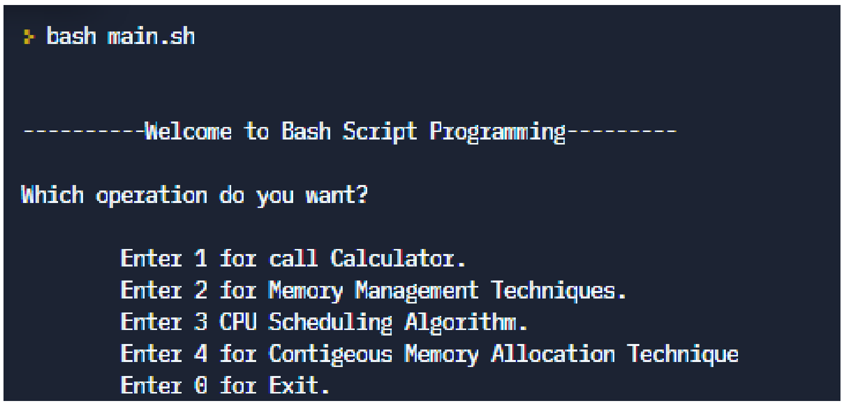
Chapter 2

Implementation of the Project

2.1 Commands

- **if…else…fi Statement:** The if...else...fi statement is the next form of control statement that allows Shell to execute statements in a controlled way and make the right choice.
- **for statement :** The for loop operate on lists of items. It repeats a set of commands for every item in a list.
- **while statement:** Here command is evaluated and based on the result loop will executed, if command raise to false then loop will be terminated.
- **Switch case in shell scripts:** The concept of the switch case statements is that we provide different cases (conditions) to the statement that, when fulfilled, will execute specific blocks of commands.
- **Function:** Shell functions are a way to group commands for later execution using a single name for the group. They are executed just like a "regular" command. When the name of a shell function is used as a simple command name, the list of commands associated with that function name is executed.

2.2 Used algorithms:

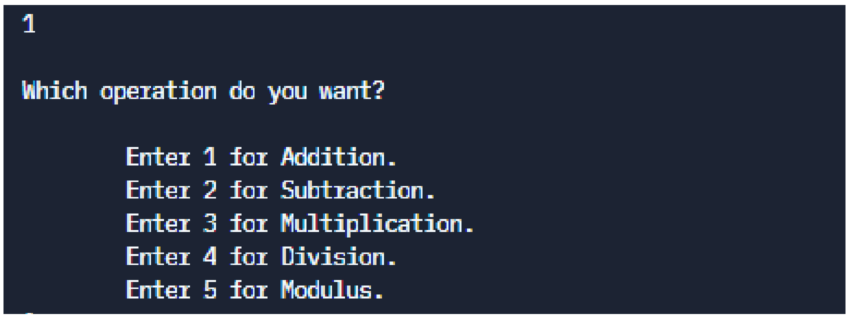


```
> bash main.sh


----------Welcome to Bash Script Programming---------

Which operation do you want?

        Enter 1 for call Calculator.
        Enter 2 for Memory Management Techniques.
        Enter 3 CPU Scheduling Algorithm.
        Enter 4 for Contigeous Memory Allocation Technique
        Enter 0 for Exit.
```

Fig 2.0: User interface

2.2.1 Calculator:

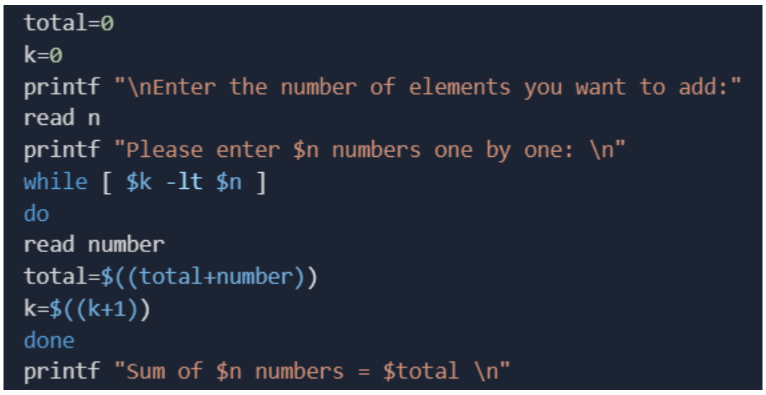
A calculator is a machine which allows people to do math operations more easily and performs arithmetic operations on numbers. The simplest calculators can do only addition, subtraction, multiplication, and division. More sophisticated calculators can handle exponential operations, roots, logarithms, trigonometric functions, and hyperbolic functions.
On our calculator portion we added 5 operations:



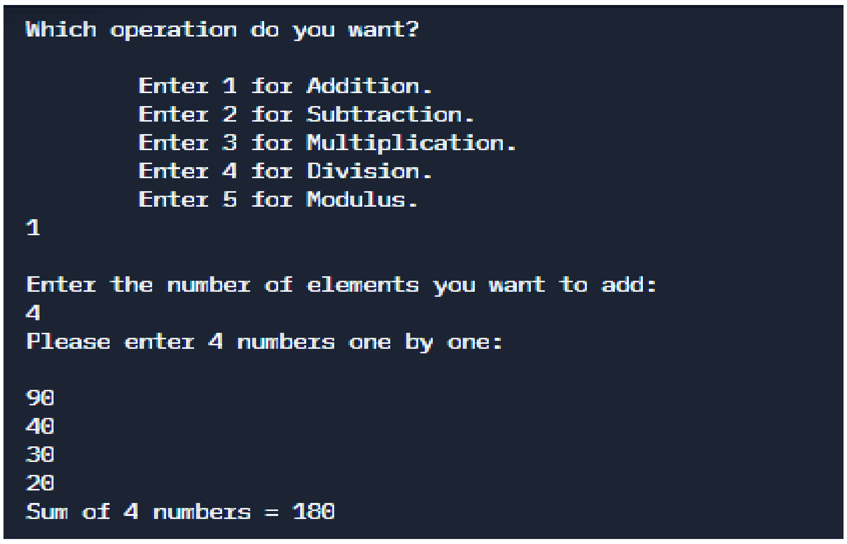
```
1

Which operation do you want?

        Enter 1 for Addition.
        Enter 2 for Subtraction.
        Enter 3 for Multiplication.
        Enter 4 for Division.
        Enter 5 for Modulus.
```

Fig 2.1: Arithmetic Operation

➢ **Addition:** It added the numbers of element what we entered.

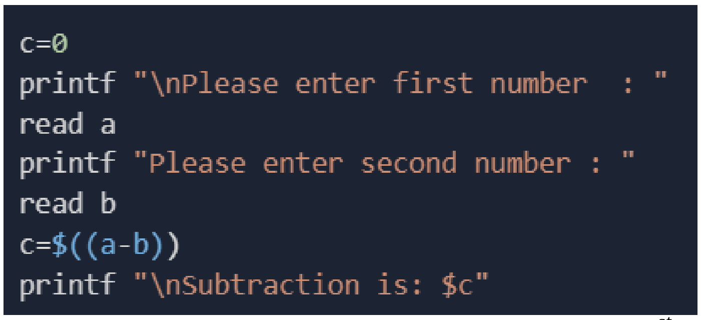


```
total=0
k=0
printf "\nEnter the number of elements you want to add:"
read n
printf "Please enter $n numbers one by one: \n"
while [ $k -lt $n ]
do
read number
total=$((total+number))
k=$((k+1))
done
printf "Sum of $n numbers = $total \n"
```

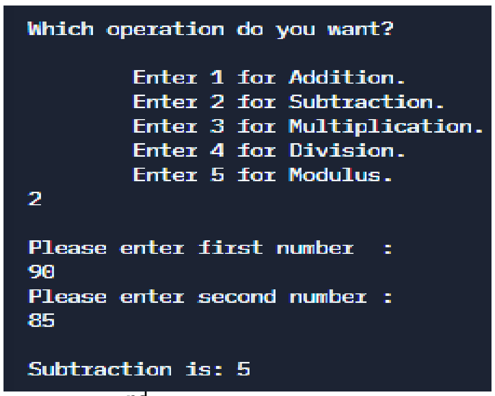


```
Which operation do you want?

        Enter 1 for Addition.
        Enter 2 for Subtraction.
        Enter 3 for Multiplication.
        Enter 4 for Division.
        Enter 5 for Modulus.
1

Enter the number of elements you want to add:
4
Please enter 4 numbers one by one:

90
40
30
20
Sum of 4 numbers = 180
```

➢ **Subtraction:** It subtracted 2numbers it works like:[1st number – 2nd number]

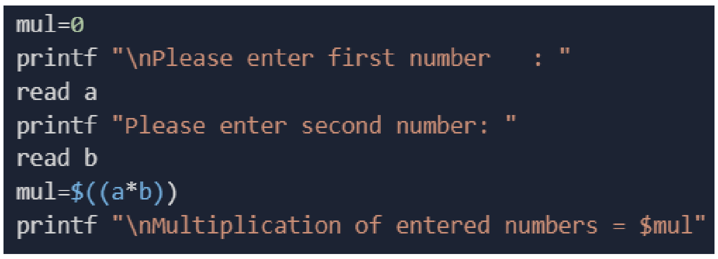


```
c=0
printf "\nPlease enter first number   : "
read a
printf "Please enter second number : "
read b
c=$((a-b))
printf "\nSubtraction is: $c"
```

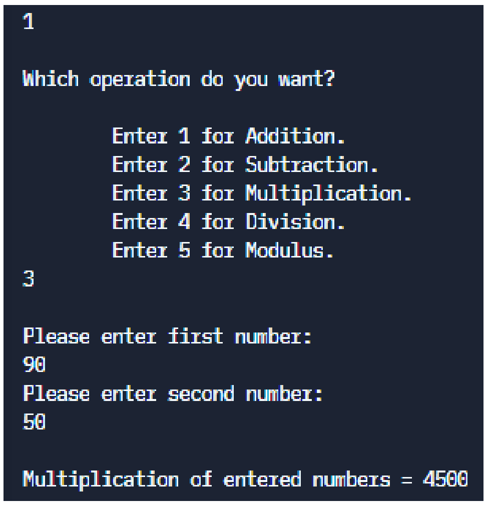


```
Which operation do you want?

        Enter 1 for Addition.
        Enter 2 for Subtraction.
        Enter 3 for Multiplication.
        Enter 4 for Division.
        Enter 5 for Modulus.
2

Please enter first number   :
90
Please enter second number :
85

Subtraction is: 5
```

➢ **<u>Multiplication</u>**: It multiplied 2numbers like: [1st number * 2nd number]

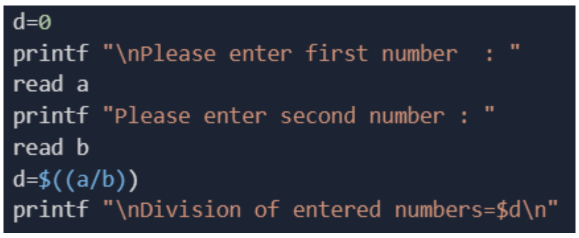


```
mul=0
printf "\nPlease enter first number   : "
read a
printf "Please enter second number: "
read b
mul=$((a*b))
printf "\nMultiplication of entered numbers = $mul"
```

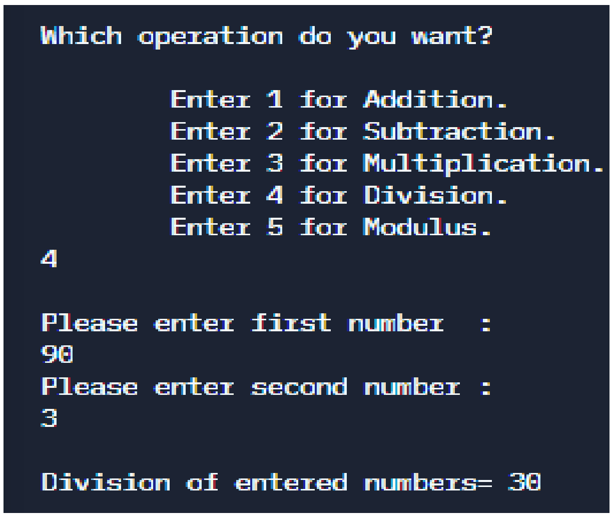


```
1

Which operation do you want?

        Enter 1 for Addition.
        Enter 2 for Subtraction.
        Enter 3 for Multiplication.
        Enter 4 for Division.
        Enter 5 for Modulus.
3

Please enter first number:
90
Please enter second number:
50

Multiplication of entered numbers = 4500
```

➢ **<u>Division:</u>** division of 2 numbers though it doesn't include for float number.

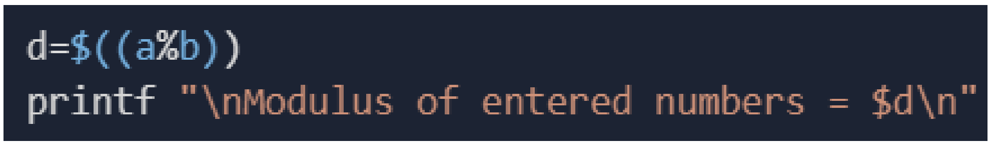


```
d=0
printf "\nPlease enter first number   : "
read a
printf "Please enter second number : "
read b
d=$((a/b))
printf "\nDivision of entered numbers=$d\n"
```

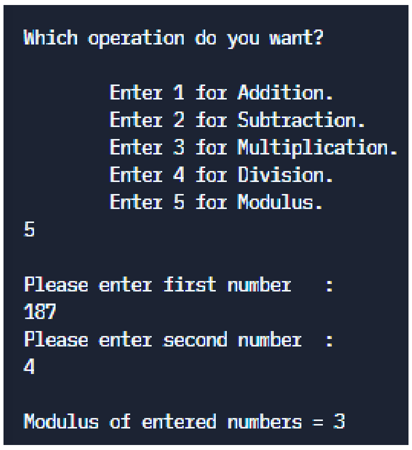


```
Which operation do you want?

        Enter 1 for Addition.
        Enter 2 for Subtraction.
        Enter 3 for Multiplication.
        Enter 4 for Division.
        Enter 5 for Modulus.
4

Please enter first number   :
90
Please enter second number :
3

Division of entered numbers= 30
```

<u>Modulus:</u> when we divided our number there can be a reminder and that is represent by modulus.



```
d=$((a%b))
printf "\nModulus of entered numbers = $d\n"
```

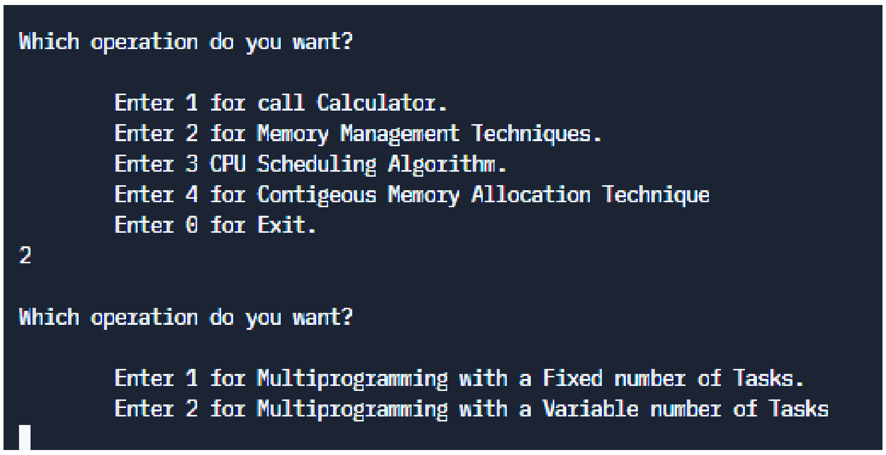


```
Which operation do you want?

        Enter 1 for Addition.
        Enter 2 for Subtraction.
        Enter 3 for Multiplication.
        Enter 4 for Division.
        Enter 5 for Modulus.
5

Please enter first number   :
187
Please enter second number  :
4

Modulus of entered numbers = 3
```

2.2.2 Memory Management System:

It has 2 parts:

1. Multiprogramming with a Fixed number of Tasks;
2. Multiprogramming with a Variable number of Tasks;

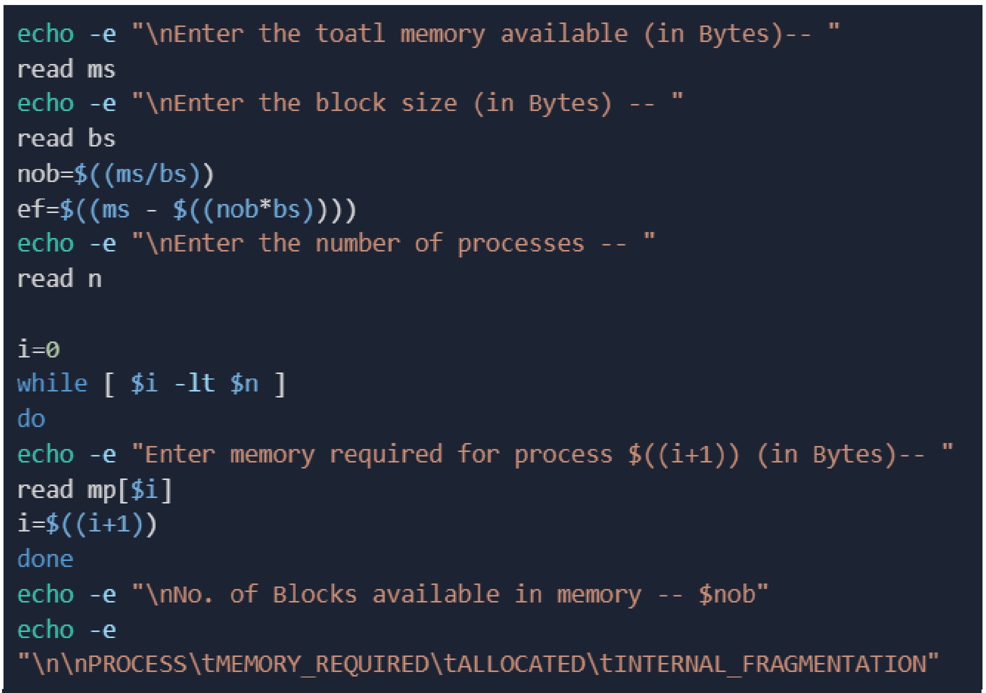


```
Which operation do you want?

        Enter 1 for call Calculator.
        Enter 2 for Memory Management Techniques.
        Enter 3 CPU Scheduling Algorithm.
        Enter 4 for Contigeous Memory Allocation Technique
        Enter 0 for Exit.
2

Which operation do you want?

        Enter 1 for Multiprogramming with a Fixed number of Tasks.
        Enter 2 for Multiprogramming with a Variable number of Tasks
```

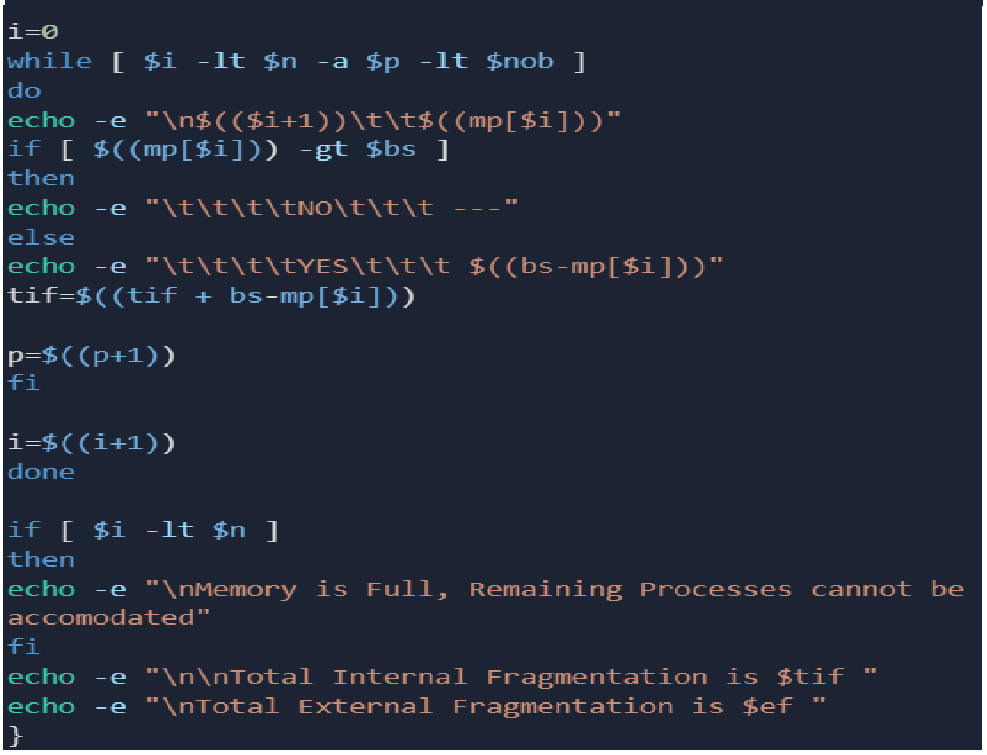
Fig 2.3: Memory Management Techniques Operations

MFT:

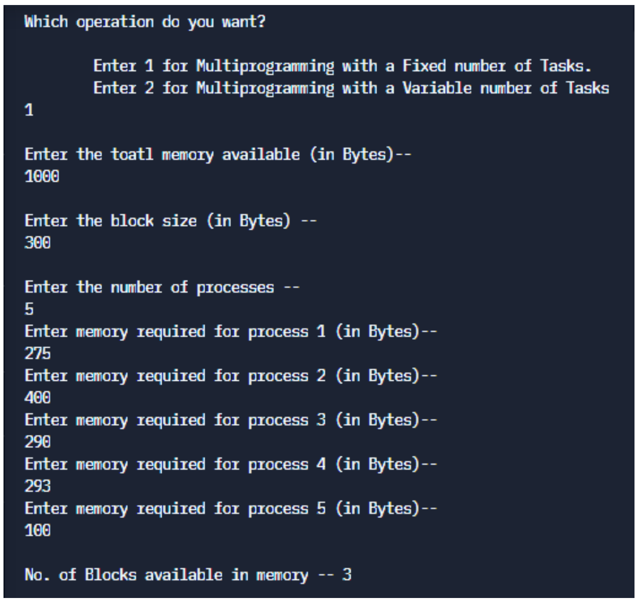
Managed file transfer (MFT) is a technology platform that uses administrative controls, support for security protocols (like HTTPS, SFTP, FTPS), and automation capabilities to help companies securely share various types of data, including sensitive or compliance-protected data as well as high-volume data. MFT is a more reliable and efficient means for secure data and file transfer, outpacing and outperforming applications such as file transfer protocol (FTP), hypertext transfer protocol (HTTP), secure file transfer protocol (SFTP) and other methods.
MFT (Multiprogramming with a Fixed number of Tasks) is one of the old memory management techniques in which the memory is partitioned into fixed size partitions and each job is assigned to a partition. The memory assigned to a partition does not change.



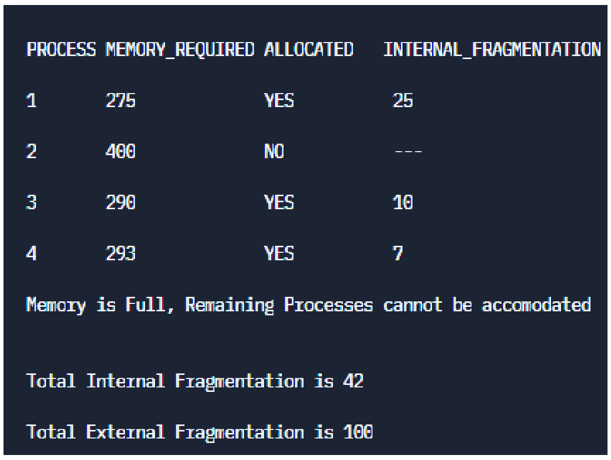
```
echo -e "\nEnter the toatl memory available (in Bytes)-- "
read ms
echo -e "\nEnter the block size (in Bytes) -- "
read bs
nob=$((ms/bs))
ef=$((ms - $((nob*bs))))
echo -e "\nEnter the number of processes -- "
read n

i=0
while [ $i -lt $n ]
do
echo -e "Enter memory required for process $((i+1)) (in Bytes)-- "
read mp[$i]
i=$((i+1))
done
echo -e "\nNo. of Blocks available in memory -- $nob"
echo -e
"\n\nPROCESS\tMEMORY_REQUIRED\tALLOCATED\tINTERNAL_FRAGMENTATION"
```



```
i=0
while [ $i -lt $n -a $p -lt $nob ]
do
echo -e "\n$(($i+1))\t\t$((mp[$i]))"
if [ $((mp[$i])) -gt $bs ]
then
echo -e "\t\t\t\tNO\t\t\t ---"
else
echo -e "\t\t\t\tYES\t\t\t $((bs-mp[$i]))"
tif=$((tif + bs-mp[$i]))

p=$((p+1))
fi

i=$((i+1))
done

if [ $i -lt $n ]
then
echo -e "\nMemory is Full, Remaining Processes cannot be
accomodated"
fi
echo -e "\n\nTotal Internal Fragmentation is $tif "
echo -e "\nTotal External Fragmentation is $ef "
}
```



```
Which operation do you want?

        Enter 1 for Multiprogramming with a Fixed number of Tasks.
        Enter 2 for Multiprogramming with a Variable number of Tasks
1

Enter the toatl memory available (in Bytes)--
1000

Enter the block size (in Bytes) --
300

Enter the number of processes --
5
Enter memory required for process 1 (in Bytes)--
275
Enter memory required for process 2 (in Bytes)--
400
Enter memory required for process 3 (in Bytes)--
290
Enter memory required for process 4 (in Bytes)--
293
Enter memory required for process 5 (in Bytes)--
100

No. of Blocks available in memory -- 3
```

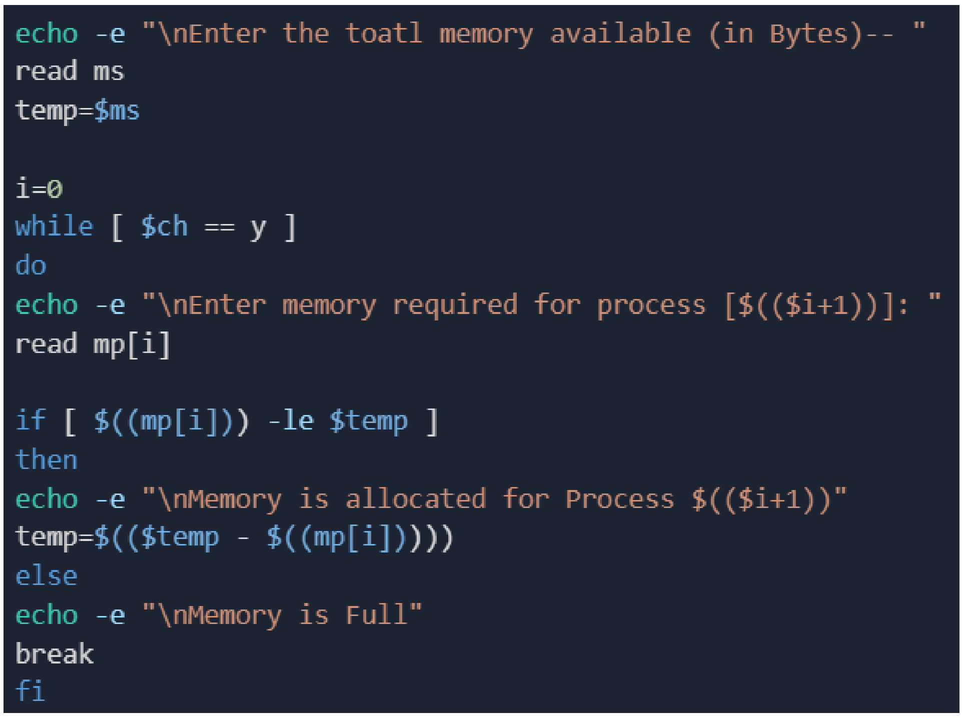


```
PROCESS MEMORY_REQUIRED ALLOCATED   INTERNAL_FRAGMENTATION

1       275             YES          25

2       400             NO           ---

3       290             YES          10

4       293             YES          7

Memory is Full, Remaining Processes cannot be accomodated


Total Internal Fragmentation is 42

Total External Fragmentation is 100
```

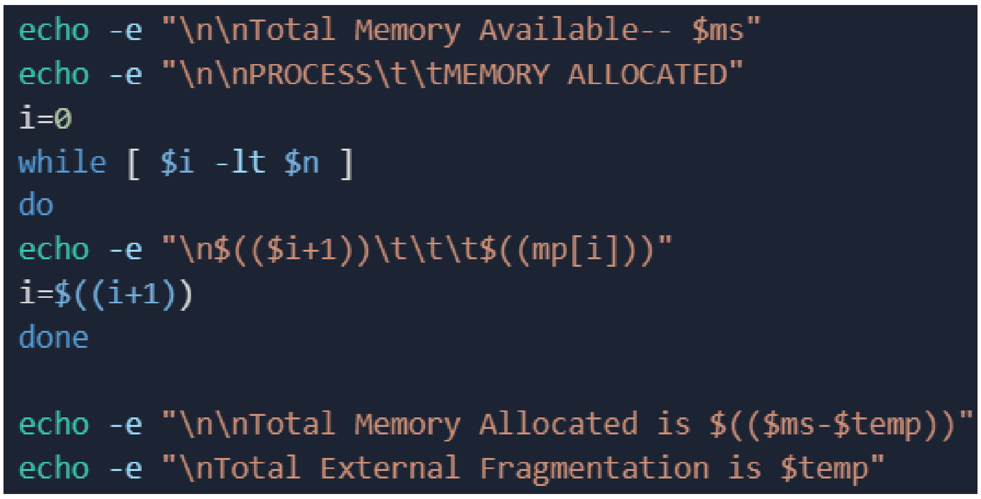
Fig 2.4: MFT

MVT:

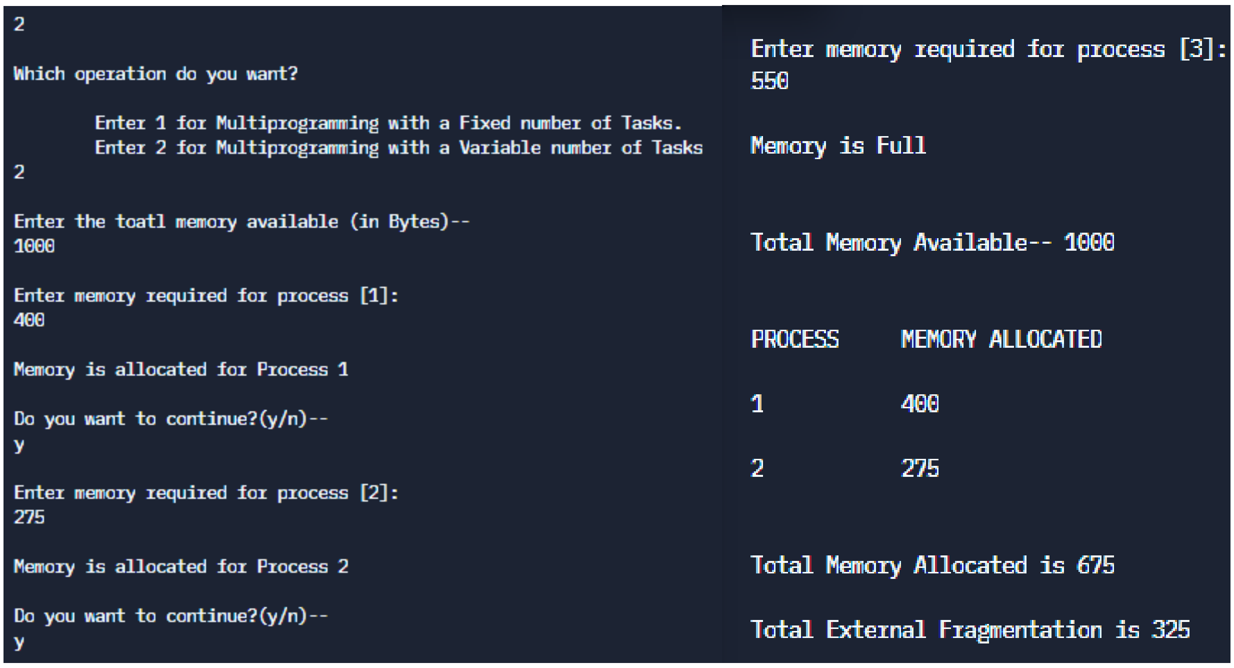
Multiprogramming with a Variable number of Tasks is the memory management technique in which each job gets just the amount of memory it needs. That is, the partitioning of memory is dynamic and changes as jobs enter and leave the system. MVT is a more ``efficient" user of resources. MFT suffers with the problem of internal fragmentation and MVT suffers with external fragmentation.



```
echo -e "\nEnter the toatl memory available (in Bytes)-- "
read ms
temp=$ms

i=0
while [ $ch == y ]
do
echo -e "\nEnter memory required for process [$(($i+1))]: "
read mp[i]

if [ $((mp[i])) -le $temp ]
then
echo -e "\nMemory is allocated for Process $(($i+1))"
temp=$(($temp - $((mp[i]))))
else
echo -e "\nMemory is Full"
break
fi
```



```
echo -e "\n\nTotal Memory Available-- $ms"
echo -e "\n\nPROCESS\t\tMEMORY ALLOCATED"
i=0
while [ $i -lt $n ]
do
echo -e "\n$(($i+1))\t\t\t$((mp[i]))"
i=$((i+1))
done

echo -e "\n\nTotal Memory Allocated is $(($ms-$temp))"
echo -e "\nTotal External Fragmentation is $temp"
```



```
2

Which operation do you want?

        Enter 1 for Multiprogramming with a Fixed number of Tasks.
        Enter 2 for Multiprogramming with a Variable number of Tasks
2

Enter the toatl memory available (in Bytes)--
1000

Enter memory required for process [1]:
400

Memory is allocated for Process 1

Do you want to continue?(y/n)--
y

Enter memory required for process [2]:
275

Memory is allocated for Process 2

Do you want to continue?(y/n)--
y
```

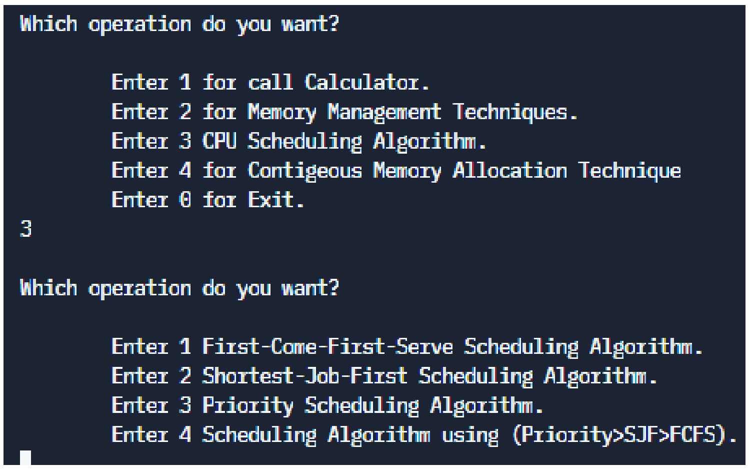
```
Enter memory required for process [3]:
550

Memory is Full


Total Memory Available-- 1000


PROCESS      MEMORY ALLOCATED

1            400

2            275


Total Memory Allocated is 675

Total External Fragmentation is 325
```

Fig 2.4: MVT

2.2.3 CPU Scheduling Algorithm:

CPU Scheduling is a process of determining which process will own CPU for execution while another process is on hold. The main task of CPU scheduling is to make sure that whenever the CPU remains idle, the OS at least select one of the processes available in the ready queue for execution. The selection process will be carried out by the CPU scheduler. It selects one of the processes in memory that are ready for execution.

1. First Come First Serve (FCFS)
2. Shortest Job First (SJF)
3. Priority Scheduling Algorithm
4. Multilevel Queue Algorithm



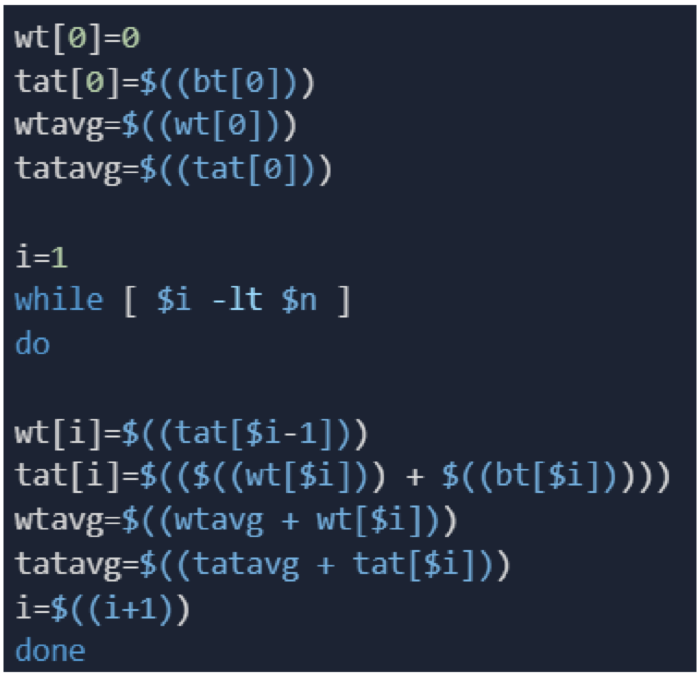
```
Which operation do you want?

        Enter 1 for call Calculator.
        Enter 2 for Memory Management Techniques.
        Enter 3 CPU Scheduling Algorithm.
        Enter 4 for Contigeous Memory Allocation Technique
        Enter 0 for Exit.
3

Which operation do you want?

        Enter 1 First-Come-First-Serve Scheduling Algorithm.
        Enter 2 Shortest-Job-First Scheduling Algorithm.
        Enter 3 Priority Scheduling Algorithm.
        Enter 4 Scheduling Algorithm using (Priority>SJF>FCFS).
```

Fig 2.5: CPU Scheduling Interface

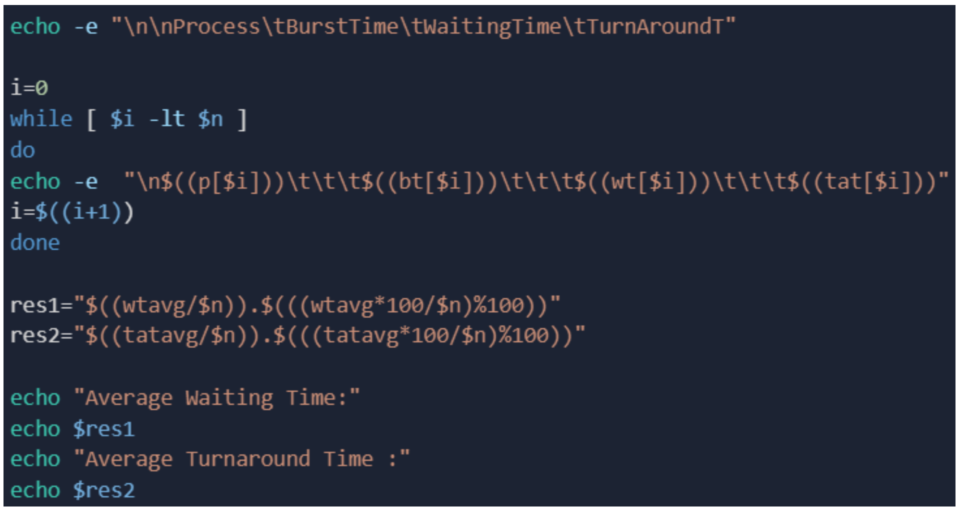
FCFS:

When process enters the ready queue, its PCB (Process Control Block) is linked with the tail of the queue. So, when CPU becomes free, it should be assigned to the process at the beginning of the queue.

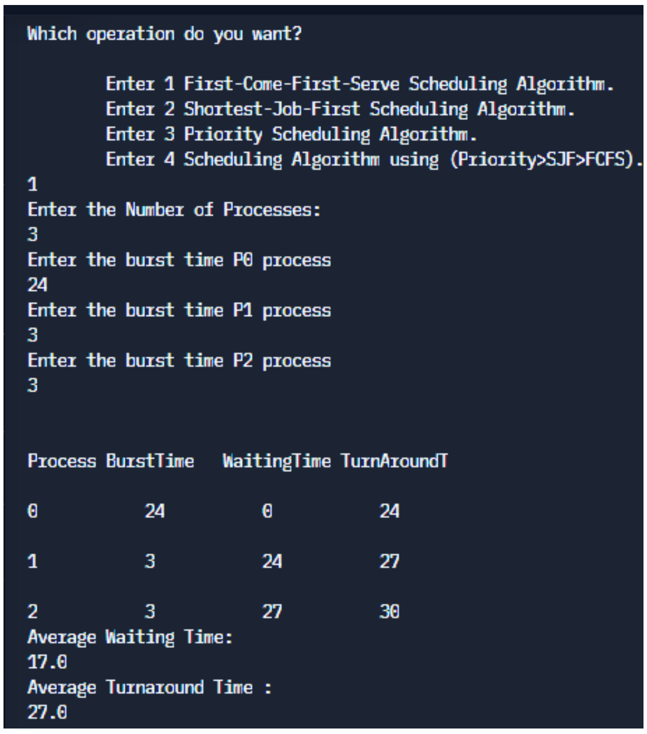
- It offers non-preemptive and pre-emptive scheduling algorithm.
- Jobs are always executed on a first-come, first-serve basis
- It is easy to implement and use.
- However, this method is poor in performance, and the general wait time is quite high.



```
wt[0]=0
tat[0]=$((bt[0]))
wtavg=$((wt[0]))
tatavg=$((tat[0]))

i=1
while [ $i -lt $n ]
do

wt[i]=$((tat[$i-1]))
tat[i]=$(($((wt[$i])) + $((bt[$i]))))
wtavg=$((wtavg + wt[$i]))
tatavg=$((tatavg + tat[$i]))
i=$((i+1))
done
```



```
echo -e "\n\nProcess\tBurstTime\tWaitingTime\tTurnAroundT"

i=0
while [ $i -lt $n ]
do
echo -e  "\n$((p[$i]))\t\t\t$((bt[$i]))\t\t\t$((wt[$i]))\t\t\t$((tat[$i]))"
i=$((i+1))
done

res1="$((wtavg/$n)).$(((wtavg*100/$n)%100))"
res2="$((tatavg/$n)).$(((tatavg*100/$n)%100))"

echo "Average Waiting Time:"
echo $res1
echo "Average Turnaround Time :"
echo $res2
```



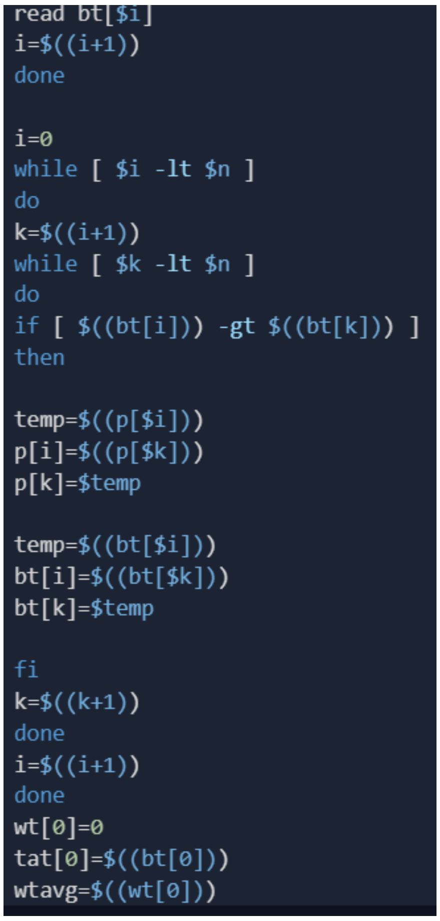
```
Which operation do you want?

        Enter 1 First-Come-First-Serve Scheduling Algorithm.
        Enter 2 Shortest-Job-First Scheduling Algorithm.
        Enter 3 Priority Scheduling Algorithm.
        Enter 4 Scheduling Algorithm using (Priority>SJF>FCFS).
1
Enter the Number of Processes:
3
Enter the burst time P0 process
24
Enter the burst time P1 process
3
Enter the burst time P2 process
3


Process BurstTime   WaitingTime TurnAroundT

0           24          0           24

1           3           24          27

2           3           27          30
Average Waiting Time:
17.0
Average Turnaround Time :
27.0
```

Fig 2.6: FCFS

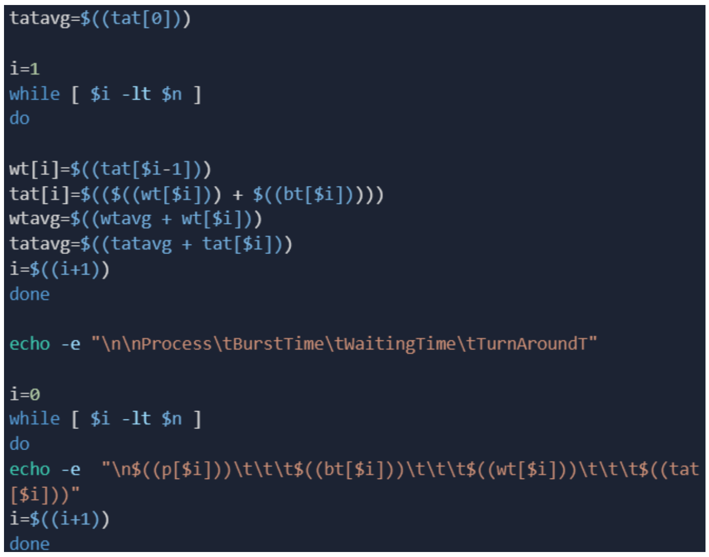
SJF:

In this method, the process will be allocated to the task, which is closest to its completion. This method prevents a newer ready state process from holding the completion of an older process.

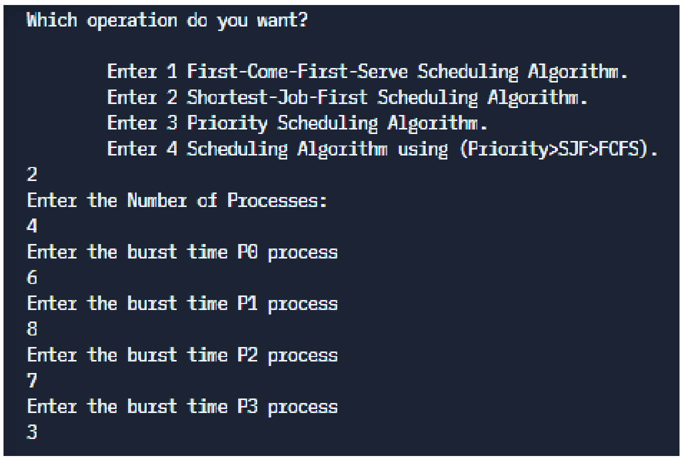
- This method is mostly applied in batch environments where short jobs are required to be given preference.
- This is not an ideal method to implement it in a shared system where the required CPU time is unknown.
- Associate with each process as the length of its next CPU burst. So that operating system uses these lengths, which helps to schedule the process with the shortest possible time.



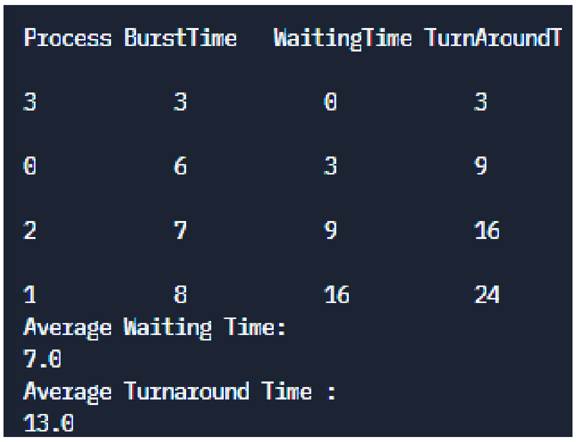
```
read bt[$i]
i=$((i+1))
done

i=0
while [ $i -lt $n ]
do
k=$((i+1))
while [ $k -lt $n ]
do
if [ $((bt[i])) -gt $((bt[k])) ]
then

temp=$((p[$i]))
p[i]=$((p[$k]))
p[k]=$temp

temp=$((bt[$i]))
bt[i]=$((bt[$k]))
bt[k]=$temp

fi
k=$((k+1))
done
i=$((i+1))
done
wt[0]=0
tat[0]=$((bt[0]))
wtavg=$((wt[0]))
```



```
tatavg=$((tat[0]))

i=1
while [ $i -lt $n ]
do

wt[i]=$((tat[$i-1]))
tat[i]=$(($((wt[$i])) + $((bt[$i]))))
wtavg=$((wtavg + wt[$i]))
tatavg=$((tatavg + tat[$i]))
i=$((i+1))
done

echo -e "\n\nProcess\tBurstTime\tWaitingTime\tTurnAroundT"

i=0
while [ $i -lt $n ]
do
echo -e  "\n$((p[$i]))\t\t\t$((bt[$i]))\t\t\t$((wt[$i]))\t\t\t$((tat
[$i]))"
i=$((i+1))
done
```



```
Which operation do you want?

        Enter 1 First-Come-First-Serve Scheduling Algorithm.
        Enter 2 Shortest-Job-First Scheduling Algorithm.
        Enter 3 Priority Scheduling Algorithm.
        Enter 4 Scheduling Algorithm using (Priority>SJF>FCFS).
2
Enter the Number of Processes:
4
Enter the burst time P0 process
6
Enter the burst time P1 process
8
Enter the burst time P2 process
7
Enter the burst time P3 process
3
```

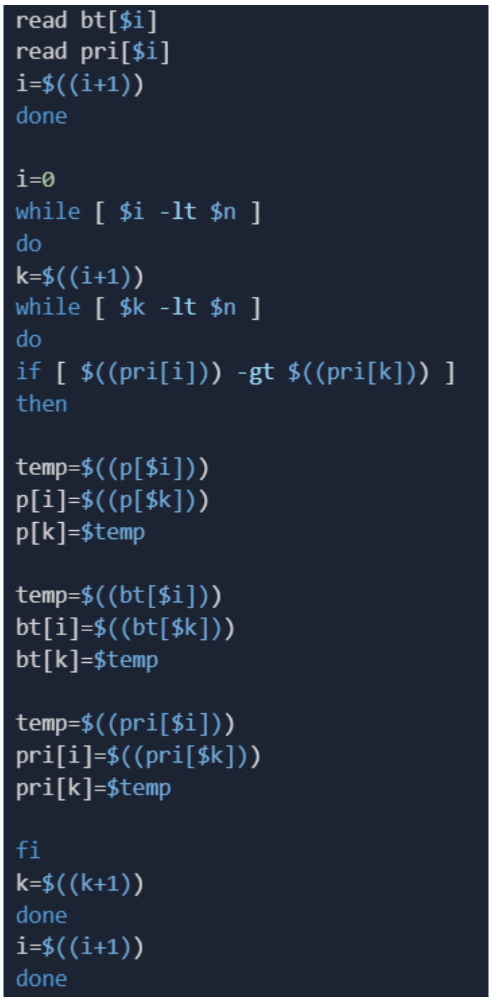


```
Process BurstTime   WaitingTime TurnAroundT

3           3           0           3

0           6           3           9

2           7           9           16

1           8           16          24
Average Waiting Time:
7.0
Average Turnaround Time :
13.0
```

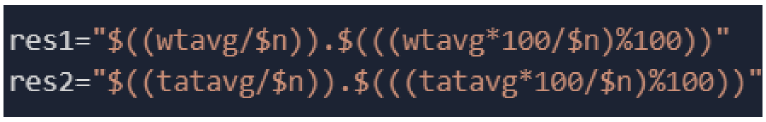
Fig 2.6: SJF

Priority Scheduling Algorithm:

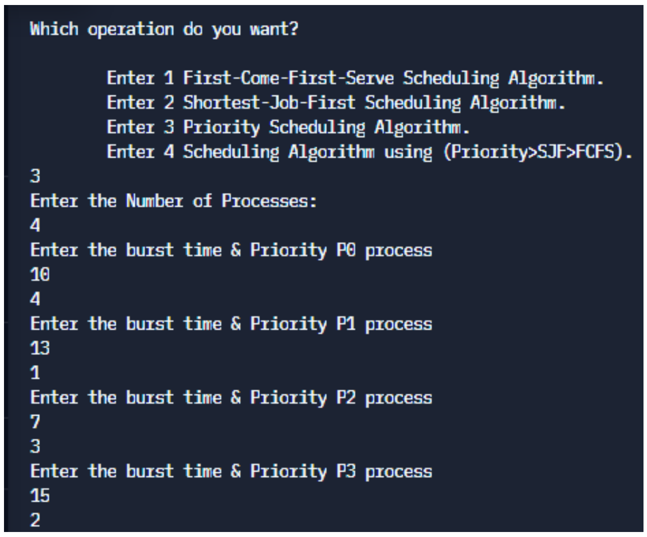
Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems. Each process is assigned first arrival time (less arrival time process first) if two processes have same arrival time, then compare to priorities (highest process first).



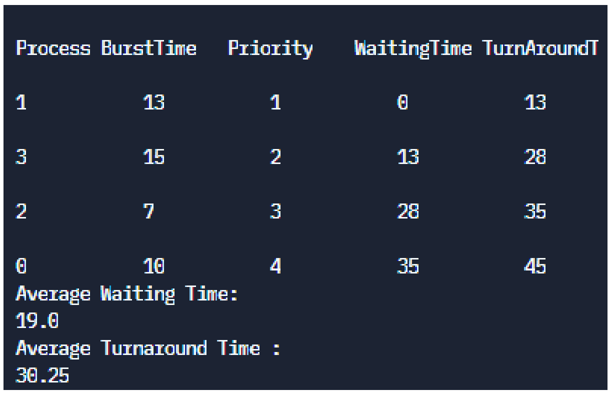
```
read bt[$i]
read pri[$i]
i=$((i+1))
done

i=0
while [ $i -lt $n ]
do
k=$((i+1))
while [ $k -lt $n ]
do
if [ $((pri[i])) -gt $((pri[k])) ]
then

temp=$((p[$i]))
p[i]=$((p[$k]))
p[k]=$temp

temp=$((bt[$i]))
bt[i]=$((bt[$k]))
bt[k]=$temp

temp=$((pri[$i]))
pri[i]=$((pri[$k]))
pri[k]=$temp

fi
k=$((k+1))
done
i=$((i+1))
done
```



```
res1="$((wtavg/$n)).$(((wtavg*100/$n)%100))"
res2="$((tatavg/$n)).$(((tatavg*100/$n)%100))"
```



```
Which operation do you want?

        Enter 1 First-Come-First-Serve Scheduling Algorithm.
        Enter 2 Shortest-Job-First Scheduling Algorithm.
        Enter 3 Priority Scheduling Algorithm.
        Enter 4 Scheduling Algorithm using (Priority>SJF>FCFS).
3
Enter the Number of Processes:
4
Enter the burst time & Priority P0 process
10
4
Enter the burst time & Priority P1 process
13
1
Enter the burst time & Priority P2 process
7
3
Enter the burst time & Priority P3 process
15
2
```

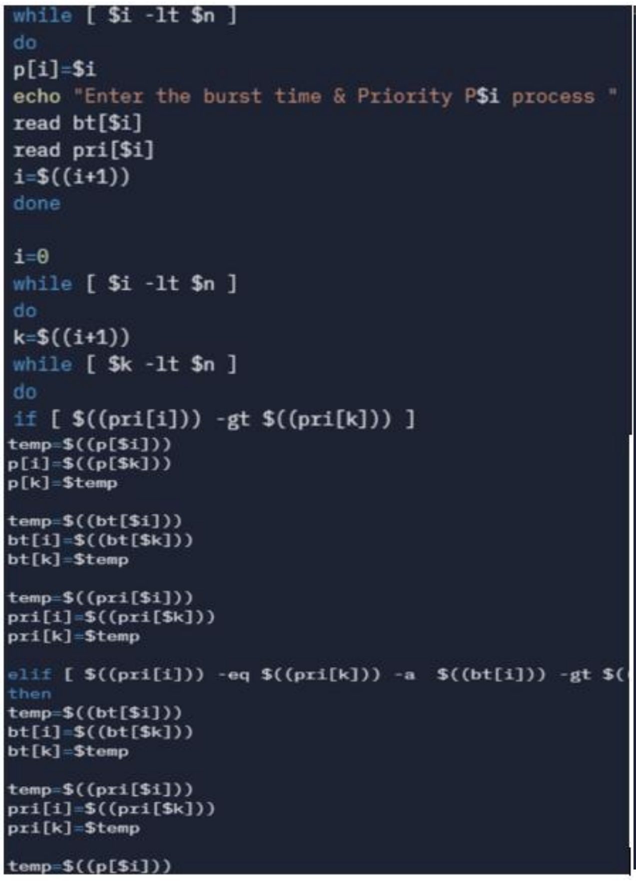


```
Process BurstTime    Priority    WaitingTime TurnAroundT

1           13          1           0           13

3           15          2           13          28

2           7           3           28          35

0           10          4           35          45
Average Waiting Time:
19.0
Average Turnaround Time :
30.25
```

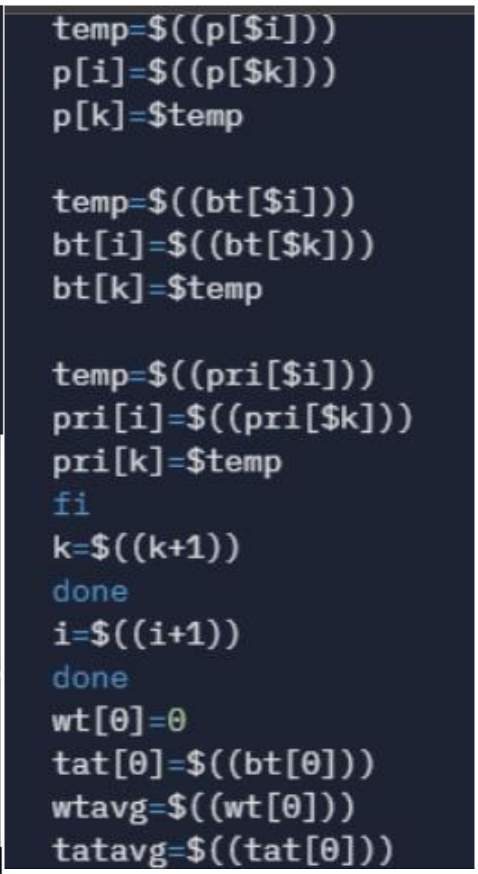
Fig 2.7: Priority Scheduling Algorithm

Mixed Algorithm:

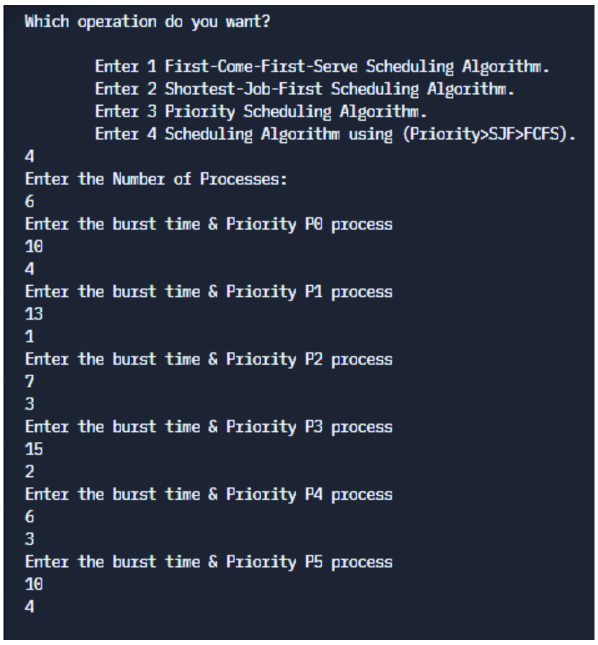
This is a mixed type algorithm. It uses FCFS-> SJF-> Priority. This problem some time follow priority scheduling algorithm and some time shortest-job-first when priority of any two or more processes are same and sometimes first-come-first-serve algorithm when priority and burst time any two or more processes are same.



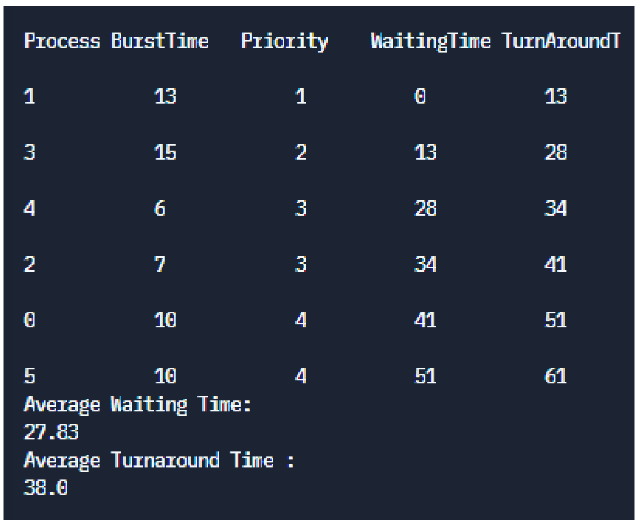
```
while [ $i -lt $n ]
do
p[i]=$i
echo "Enter the burst time & Priority P$i process "
read bt[$i]
read pri[$i]
i=$((i+1))
done

i=0
while [ $i -lt $n ]
do
k=$((i+1))
while [ $k -lt $n ]
do
if [ $((pri[i])) -gt $((pri[k])) ]
temp=$((p[$i]))
p[i]=$((p[$k]))
p[k]=$temp

temp=$((bt[$i]))
bt[i]=$((bt[$k]))
bt[k]=$temp

temp=$((pri[$i]))
pri[i]=$((pri[$k]))
pri[k]=$temp

elif [ $((pri[i])) -eq $((pri[k])) -a  $((bt[i])) -gt $((
then
temp=$((bt[$i]))
bt[i]=$((bt[$k]))
bt[k]=$temp

temp=$((pri[$i]))
pri[i]=$((pri[$k]))
pri[k]=$temp

temp=$((p[$i]))
```



```
temp=$((p[$i]))
p[i]=$((p[$k]))
p[k]=$temp

temp=$((bt[$i]))
bt[i]=$((bt[$k]))
bt[k]=$temp

temp=$((pri[$i]))
pri[i]=$((pri[$k]))
pri[k]=$temp
fi
k=$((k+1))
done
i=$((i+1))
done
wt[0]=0
tat[0]=$((bt[0]))
wtavg=$((wt[0]))
tatavg=$((tat[0]))
```



```
Which operation do you want?

        Enter 1 First-Come-First-Serve Scheduling Algorithm.
        Enter 2 Shortest-Job-First Scheduling Algorithm.
        Enter 3 Priority Scheduling Algorithm.
        Enter 4 Scheduling Algorithm using (Priority>SJF>FCFS).
4
Enter the Number of Processes:
6
Enter the burst time & Priority P0 process
10
4
Enter the burst time & Priority P1 process
13
1
Enter the burst time & Priority P2 process
7
3
Enter the burst time & Priority P3 process
15
2
Enter the burst time & Priority P4 process
6
3
Enter the burst time & Priority P5 process
10
4
```



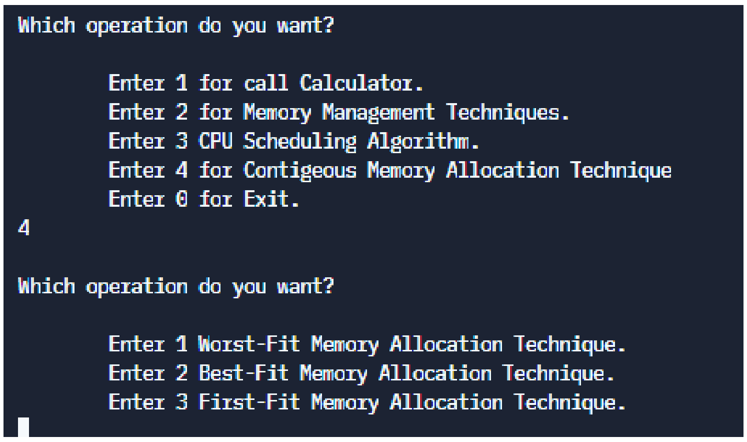
```
Process BurstTime   Priority    WaitingTime TurnAroundT

1           13          1           0           13

3           15          2           13          28

4           6           3           28          34

2           7           3           34          41

0           10          4           41          51

5           10          4           51          61
Average Waiting Time:
27.83
Average Turnaround Time :
38.0
```

Fig 2.7: Scheduling Algorithm (Priority>SJF>FCFS)

2.2.4 Contiguous Memory Allocation Technique:

It has three parts:

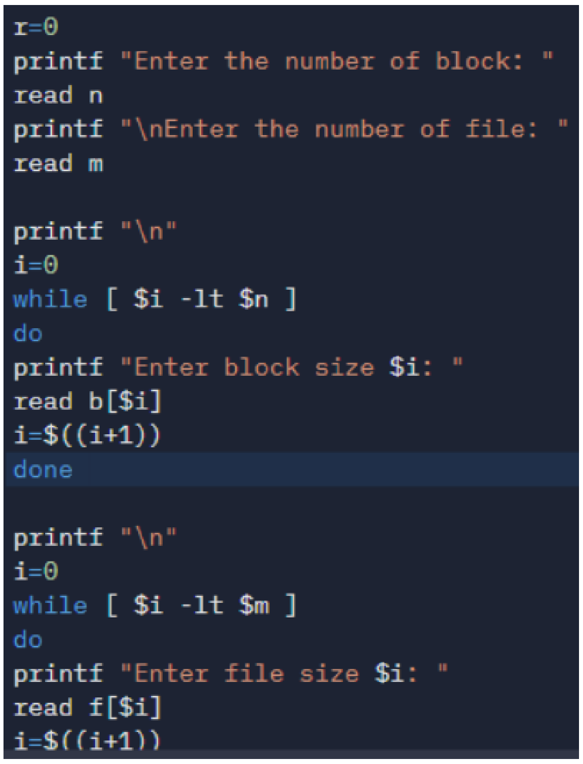
1. Worst-fit memory allocation technique
2. Best-fit memory allocation technique
3. First-fit memory allocation technique



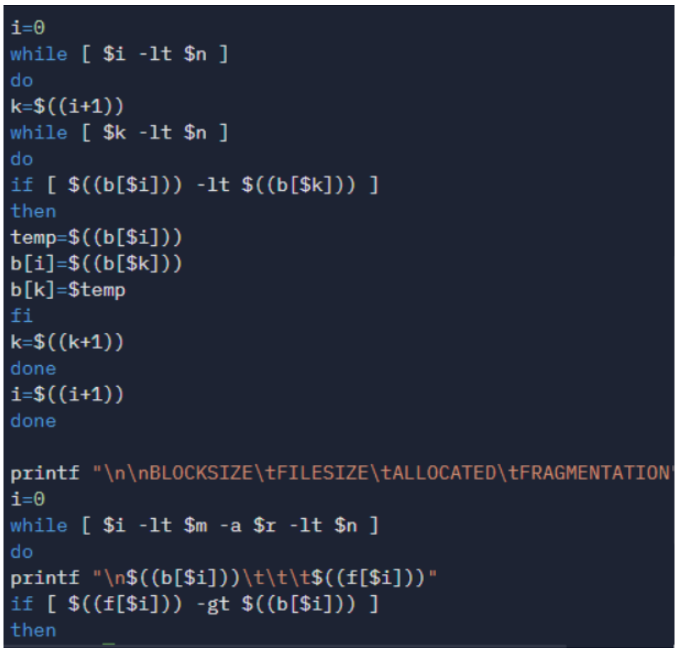
```
Which operation do you want?

        Enter 1 for call Calculator.
        Enter 2 for Memory Management Techniques.
        Enter 3 CPU Scheduling Algorithm.
        Enter 4 for Contigeous Memory Allocation Technique
        Enter 0 for Exit.
4

Which operation do you want?

        Enter 1 Worst-Fit Memory Allocation Technique.
        Enter 2 Best-Fit Memory Allocation Technique.
        Enter 3 First-Fit Memory Allocation Technique.
```

Fig 2.8: Contiguous Memory Allocation Interface

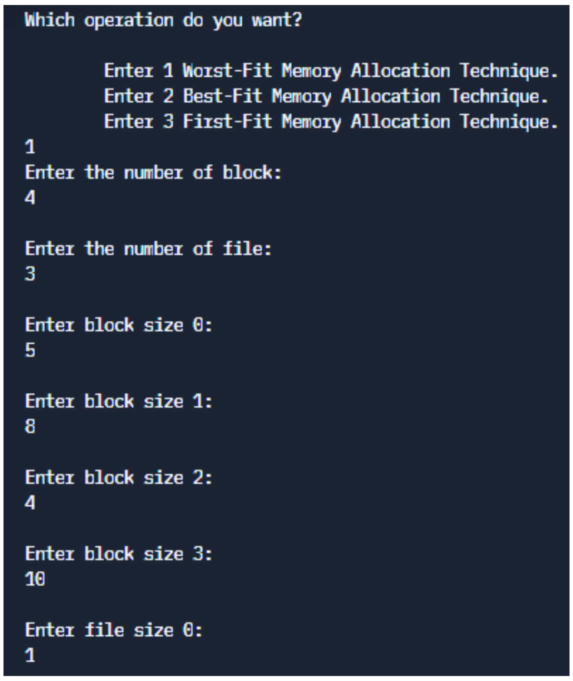
Worst- fit memory allocation technique: Worst Fit allocates a process to the partition which is largest sufficient among the freely available partitions available in the main memory. If a large process comes at a later stage, then memory will not have space to accommodate it.



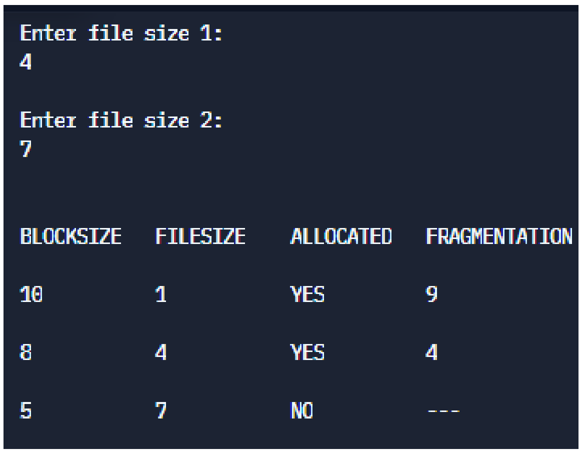
```
r=0
printf "Enter the number of block: "
read n
printf "\nEnter the number of file: "
read m

printf "\n"
i=0
while [ $i -lt $n ]
do
printf "Enter block size $i: "
read b[$i]
i=$((i+1))
done

printf "\n"
i=0
while [ $i -lt $m ]
do
printf "Enter file size $i: "
read f[$i]
i=$((i+1))
```



```
i=0
while [ $i -lt $n ]
do
k=$((i+1))
while [ $k -lt $n ]
do
if [ $((b[$i])) -lt $((b[$k])) ]
then
temp=$((b[$i]))
b[i]=$((b[$k]))
b[k]=$temp
fi
k=$((k+1))
done
i=$((i+1))
done

printf "\n\nBLOCKSIZE\tFILESIZE\tALLOCATED\tFRAGMENTATION
i=0
while [ $i -lt $m -a $r -lt $n ]
do
printf "\n$((b[$i]))\t\t\t$((f[$i]))"
if [ $((f[$i])) -gt $((b[$i])) ]
then
```



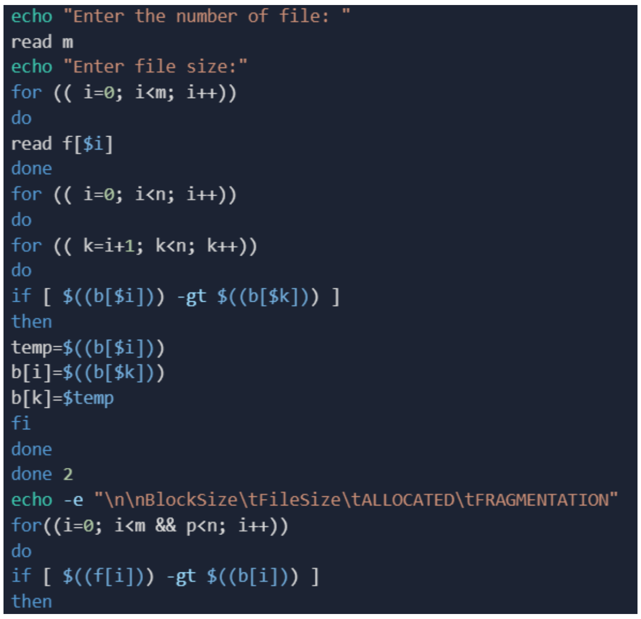
```
Which operation do you want?

        Enter 1 Worst-Fit Memory Allocation Technique.
        Enter 2 Best-Fit Memory Allocation Technique.
        Enter 3 First-Fit Memory Allocation Technique.
1
Enter the number of block:
4

Enter the number of file:
3

Enter block size 0:
5

Enter block size 1:
8

Enter block size 2:
4

Enter block size 3:
10

Enter file size 0:
1
```



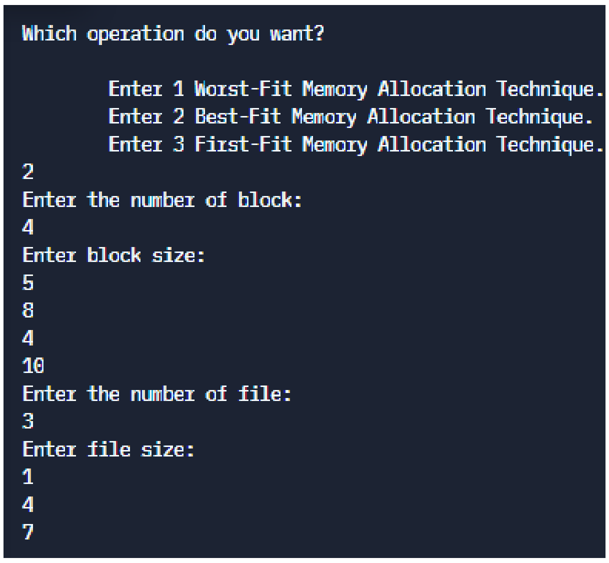
```
Enter file size 1:
4

Enter file size 2:
7


BLOCKSIZE     FILESIZE     ALLOCATED     FRAGMENTATION

10            1            YES           9

8             4            YES           4

5             7            NO            ---
```

Fig 2.9: Worst-fit

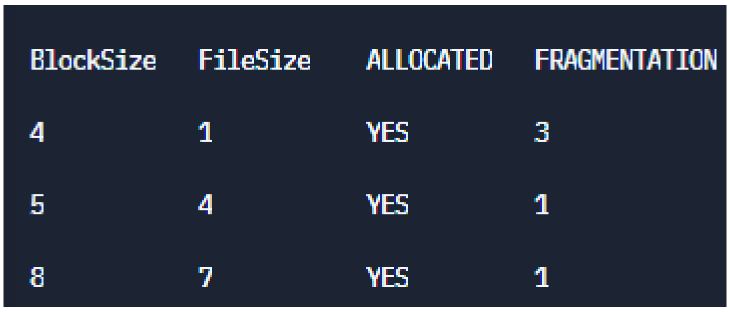
Best- fit memory allocation technique: Best fit allocates the process to a partition which is the smallest sufficient partition among the free available partitions. This method keeps the free/busy list in order by size – smallest to largest. In this method, the operating system first searches the whole of the memory according to the size of the given job and allocates it to the closest-fitting free partition in the memory, making it able to use memory efficiently. Here the jobs are in the order from smallest job to largest job.



```
echo "Enter the number of file: "
read m
echo "Enter file size:"
for (( i=0; i<m; i++))
do
read f[$i]
done
for (( i=0; i<n; i++))
do
for (( k=i+1; k<n; k++))
do
if [ $((b[$i])) -gt $((b[$k])) ]
then
temp=$((b[$i]))
b[i]=$((b[$k]))
b[k]=$temp
fi
done
done 2
echo -e "\n\nBlockSize\tFileSize\tALLOCATED\tFRAGMENTATION"
for((i=0; i<m && p<n; i++))
do
if [ $((f[i])) -gt $((b[i])) ]
then
```



```
Which operation do you want?

        Enter 1 Worst-Fit Memory Allocation Technique.
        Enter 2 Best-Fit Memory Allocation Technique.
        Enter 3 First-Fit Memory Allocation Technique.
2
Enter the number of block:
4
Enter block size:
5
8
4
10
Enter the number of file:
3
Enter file size:
1
4
7
```



```
BlockSize   FileSize    ALLOCATED   FRAGMENTATION

4           1           YES         3

5           4           YES         1

8           7           YES         1
```

Fig 2.9: Best-fit

First fit- memory allocation technique: In the first fit approach is to allocate the first free partition or hole large enough which can accommodate the process. It finishes after finding the first suitable free partition.

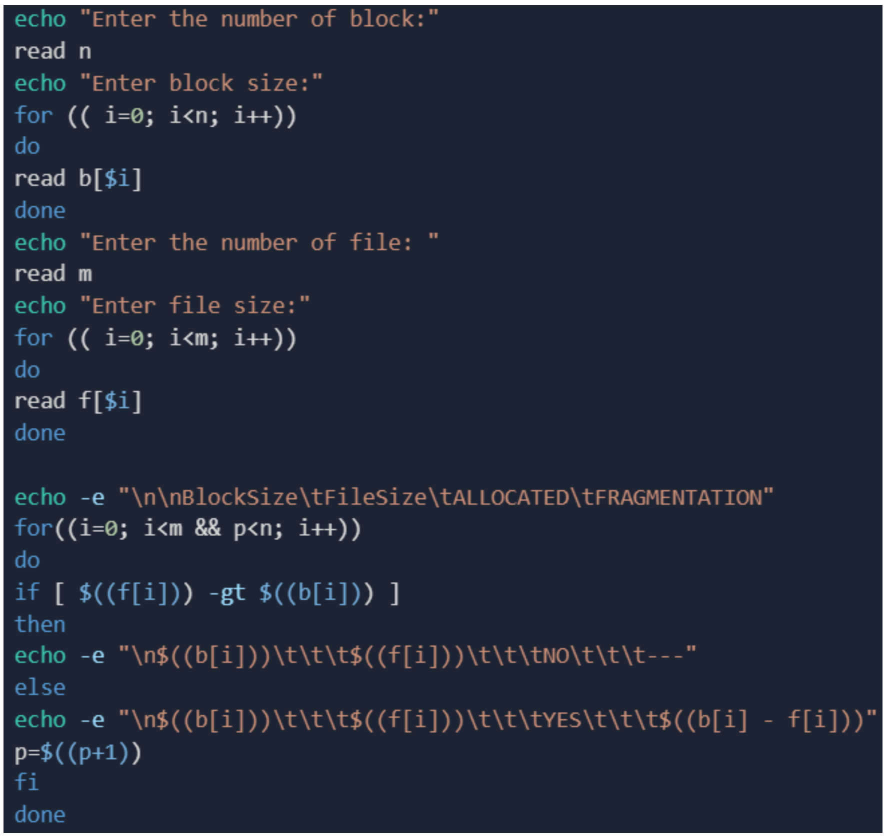
It is the simplest technique of allocating the memory block to the processes amongst all. In this algorithm, the pointer keeps track of all the free blocks in the memory and accepts the request of allocating a memory block to the coming process.

Advantage:

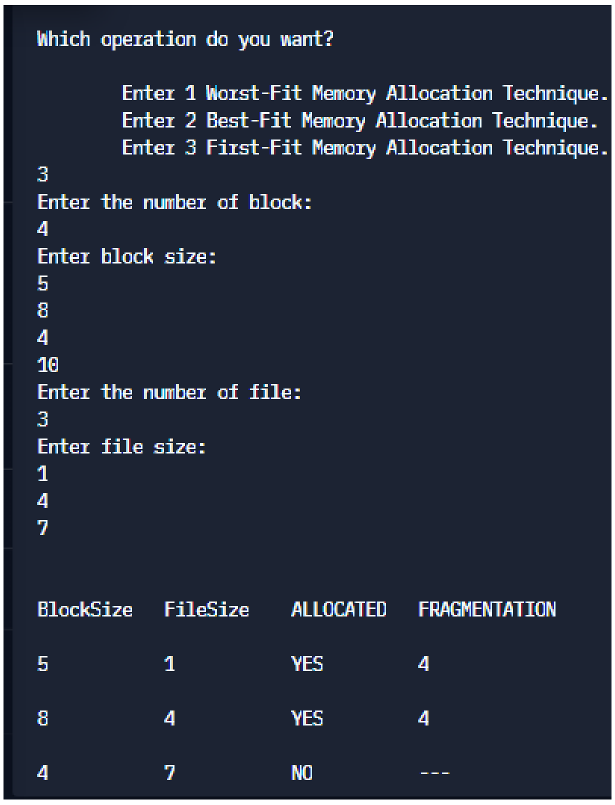
Fastest algorithm because it searches as little as possible;

Disadvantage

The remaining unused memory areas left after allocation become waste if it is too smaller. Thus request for larger memory requirement cannot be accomplished.



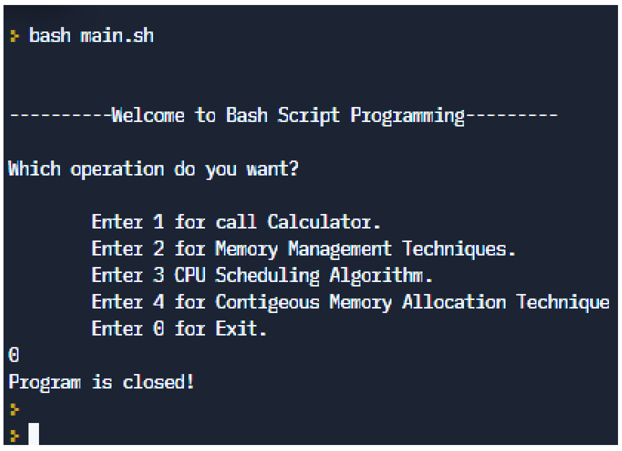
```
echo "Enter the number of block:"
read n
echo "Enter block size:"
for (( i=0; i<n; i++))
do
read b[$i]
done
echo "Enter the number of file: "
read m
echo "Enter file size:"
for (( i=0; i<m; i++))
do
read f[$i]
done

echo -e "\n\nBlockSize\tFileSize\tALLOCATED\tFRAGMENTATION"
for((i=0; i<m && p<n; i++))
do
if [ $((f[i])) -gt $((b[i])) ]
then
echo -e "\n$((b[i]))\t\t\t$((f[i]))\t\t\tNO\t\t\t---"
else
echo -e "\n$((b[i]))\t\t\t$((f[i]))\t\t\tYES\t\t\t$((b[i] - f[i]))"
p=$((p+1))
fi
done
```



```
Which operation do you want?

        Enter 1 Worst-Fit Memory Allocation Technique.
        Enter 2 Best-Fit Memory Allocation Technique.
        Enter 3 First-Fit Memory Allocation Technique.
3
Enter the number of block:
4
Enter block size:
5
8
4
10
Enter the number of file:
3
Enter file size:
1
4
7


BlockSize   FileSize    ALLOCATED   FRAGMENTATION

5           1           YES         4

8           4           YES         4

4           7           NO          ---
```

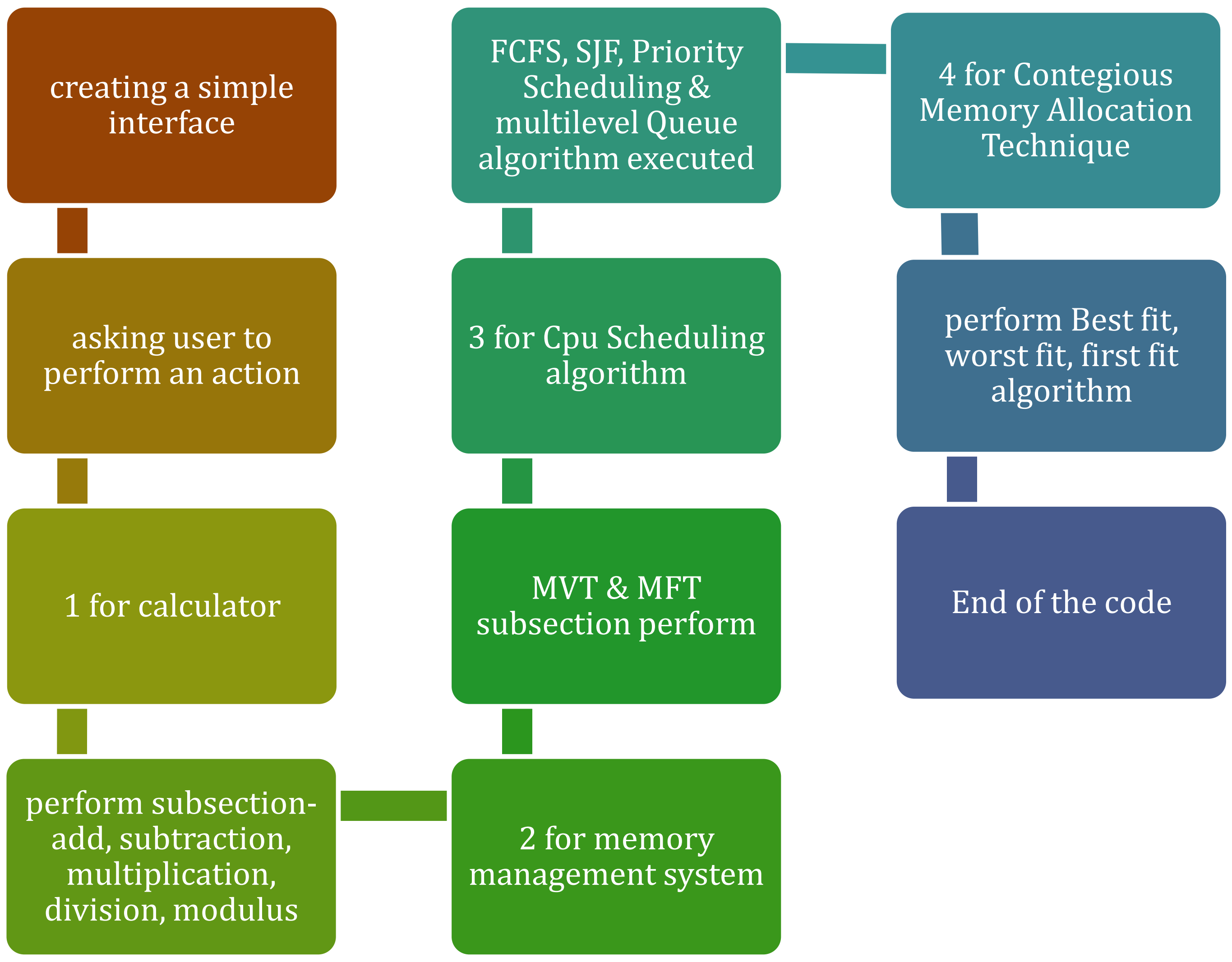
Fig 2.10: First-fit

2.2.5 Exit Program:



```
> bash main.sh


----------Welcome to Bash Script Programming---------

Which operation do you want?

        Enter 1 for call Calculator.
        Enter 2 for Memory Management Techniques.
        Enter 3 CPU Scheduling Algorithm.
        Enter 4 for Contigeous Memory Allocation Technique
        Enter 0 for Exit.
0
Program is closed!
>
>
```

2.3 Flow Chart:


creating a simple interface
asking user to perform an action
1 for calculator
perform subsection- add, subtraction, multiplication, division, modulus
2 for memory management system
MVT & MFT subsection perform
3 for Cpu Scheduling algorithm
FCFS, SJF, Priority Scheduling & multilevel Queue algorithm executed
4 for Contegious Memory Allocation Technique
perform Best fit, worst fit, first fit algorithm
End of the code

Chapter 3

Performance Evaluation

Completion rate: 100%

3.1 Results and Discussions

3.1.1 Results

We tried to make a simple book for a beginner and implemented some of Operating system based algorithm using shell scripting.

3.1.2 Analysis and Outcome

- ✓ Successfully completed the basic commands
- ✓ Successfully completed algorithms of FCFS, SJF, Multilevel Queue & Priority Scheduling
- ✓ Successfully completed Contagious Memory Allocation Technique
- ✓ Successfully completed Memory Management System
- ✓ Successfully completed Calculator.

Chapter4

Conclusion

4.1 Introduction

From this project we tried to summarize the whole idea and algorithms that we practiced and learned on our whole course and described what the algorithm is. A person can demonstrate the basic concept, algorithms, shell commands and their proper usage.

4.1 Practical Implications

- ✓ Our project can help a beginner to learn the basic of shell
- ✓ It helps someone to enrich their knowledge with functions and algorithms
- ✓ It helps someone to find out the better algorithm with the comparison.

4.2 Scope of Future Work

- We will try to add an interface here.
- We will add all the algorithms of operating system here.
- And will try to make it just like an advance book or application.
- Another thing that is, we will try to make a converter where anyone finds the value of all the algorithm of operating system.

References

[1] Andrew S. Tanenbaum and Albert S. Woodhull (2007). Operating Systems: Design and Implementation, Prentice Hall, 3 rd Edition.

[2] Sandeep Jain; Last Updated: 28 Jun, 2021;Last visit: 9/12/21; https://www.geeksforgeeks.org/operating-systems/

[3] Mohammad Mohtashim; Last visit: 9/12/21; https://www.tutorialspoint.com/operating_system/

[4] Abhishek Ahlawat; Last visit: 9/12/21; https://www.studytonight.com/operating-system/

[5] Last visit: 9/12/21; https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/