

## Experiment 1

### EXP.NO :1.a (System Calls)

**Aim:** To write programs to perform following operations in UNIX:

a) Process Creation

It is used to create process

COMMAND:- fork()

CODE :-

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
    int val;
```

```
    val = fork();
```

```
    printf("%d", val);
```

```
}
```



b) Executing a command

It is used to execute a already existing program in CPU.

COMMAND:-exec

c) Sleep command

It is used to delay for a specified amount of time .

COMMAND:-sleep

d) Sleep command using get pid

e) Signal handling using kill

It command is used to send a signal to a particular process for terminating it.

COMMAND:-kill [-s signal][p]

f) Wait command

It is used to pause until execution of a background process has ended.

COMMAND:- wait

**EXP.NO :1.b (I/O System calls)**

**Aim: To write programs to perform following operations in UNIX:**

a) Reading from a file

This command is used for reading the content of the file.

COMMAND:- cat <filename>

b) Writing into a file

This command is used to write in the file created or in the new file after creation

COMMAND:- cat> filename

<content>

Ctrl+d – exit writing in file

c) File Creation

This command is used for creating a new file in the system

COMMAND:- touch <filename>

d) Implementation of ls command

This command is used to list all the files and directories

COMMAND:- ls

e) Implementation of grep command.

It is used for Searching pattern in each file or standard input

COMMAND:-grep [options] pattern [files]

Here "i" is used to ignore the case sensitive

Here "c" is used to count the pattern that matches with the input

## Experiment 2 (Scheduling)

EXP.NO :2.a (FIRST COME FIRST SERVE)

**Aim:** To write a C program to implement the CPU scheduling algorithm for FIRST COME FIRST SERVE.

### PROBLEM DESCRIPTION:

CPU scheduler will decide which process should be given the CPU for its execution. For this it uses different algorithm to choose among the process. One among that algorithm is FCFS algorithm.

In this algorithm the process which arrives first is given the CPU after finishing its request only it will allow CPU to execute other process.

### ALGORITHM :-

1. Input the Process ID pid[i], Arrival Time at[i], Burst Time bt[i].
2. Compare arrival time at[i] using bubble sort and sort values of process id ,arrival time and burst time by swapping respectively.
3. Calculate completion time ct[j] using  
for(int j=0;j<n;j++)  
{  
    sum+=bt[j];  
    ct[j]=sum;  
}
4. Calculate turn around time tat[j] using  
tat[j]=ct[j]-at[j]
5. Calculate waiting time wt[j] using  
wt[j]=tat[j]-bt[j]
6. Calculate Average waiting time using  
Average wt= totalWT/n

### CODE:-

```
#include<stdio.h>
int main(){
    int pid[10],bt[10],at[10],tat[10],wt[10],ct[10]={0};
```

```

int n,sum=0;
float totalWT=0;
printf("Enter number of processes");
scanf("%d",&n);
printf("Enter process id and arrival time and burst time for each process\n\n\n");
for(int i=0;i<n;i++)
{
printf("process id of the process[%d] ",i+1);
scanf("%d",&pid[i]);
    printf("Arrival time of process[%d]          ",i+1);
    scanf("%d",&at[i]);
    printf("Burst time of process[%d] ",i+1);
    scanf("%d",&bt[i]);
    printf("\n");
}
void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}
void bubbleSort(int at[], int n)
{
    int i, j;
    for(i = 0; i < n-1; i++)
    {
        for (j = 0; j < n-i-1; j++)
        {
            if (at[j] > at[j+1])
            {
                swap(&at[j], &at[j+1]);
                swap(&pid[j],&pid[j+1]);
                swap(&bt[j],&bt[j+1]);
            }
        }
    }
}

```

```

        }
    }
    bubbleSort(at,n);
    for(int j=0;j<n;j++)
    {
        sum+=bt[j];
        ct[j]+=sum;
    }
    for(int k=0;k<n;k++)
    {
        tat[k]=ct[k]-at[k];
    }
    for(int k=0;k<n;k++)
    {
        wt[k]=tat[k]-bt[k];
        totalWT+=wt[k];
    }
    printf("PID\t AT\t BT\t CT\t TAT\t WT\t\n\n");
    for(int i=0;i<n;i++)
    {
        printf("P%d\t %d\t %d\t %d\t %d\t %d\n",pid[i],at[i],bt[i],ct[i],tat[i],wt[i]);
    }
    printf("Average WT = %f\n\n",totalWT/n);
    return 0;
}

```

OUTPUT :

Time(sec) : 0 Memory(MB) : 1.6872196844482

Output:

process id of the process	Arrival time of process	Burst time of process	CT	TAT	WT
P1	0	5	5	5	0

Time(sec) : 0 Memory(MB) : 1.6872196844482

Output:

process id of the process	Arrival time of process	Burst time of process	CT	TAT	WT
P1	0	5	5	5	0
P2	0	7	12	12	5
P3	3	6	18	15	9
P4	5	9	27	22	13

EXP.NO :2.B (SHORTEST JOB FIRST)

Aim: To write a C program to implement the CPU scheduling algorithm for SHORTEST JOB FIRST.

### PROBLEM DESCRIPTION:

CPU scheduler will decide which process should be given the CPU for its execution. For this it uses different algorithm to choose among the process. One among that algorithm is SJF algorithm.

In this algorithm the process which has less service time given the CPU after finishing its request only it will allow CPU to execute other process.

ALGORITHM :-

1. Input the Process ID  $pid[i]$ , Arrival Time  $at[i]$ , Burst Time  $bt[i]$ .
2. Compare burst time  $bt[i]$  using bubble sort and sort values of process id ,arrival time and burst time by swapping respectively.

3. Calculate completion time  $ct[j]$  using  

```
for(int j=0;j<n;j++)
{
    sum+=bt[j];
    ct[j]=sum;
}
```
4. Calculate turn around time  $tat[j]$  using  
 $tat[j]=ct[j]-at[j]$
5. Calculate waiting time  $wt[j]$  using  
 $wt[j]=tat[j]-bt[j]$
6. Calculate Average waiting time using  
 $Average\ wt = totalWT/n$

CODE :-

```
#include<stdio.h>

int main(){
int pid[10],bt[10],at[10],tat[10],wt[10],ct[10]={0};
int n,sum=0;
float totalWT=0;
printf("Enter number of processes");
scanf("%d",&n);
printf("Enter process id and arrival time and burst time for each process\n\n\n");
for(int i=0;i<n;i++)
{
    printf("process id of the process[%d] ",i+1);
    scanf("%d",&pid[i]);
    printf("Arrival time of process[%d] ",i+1);
    scanf("%d",&at[i]);
    printf("Burst time of process[%d] ",i+1);
    scanf("%d",&bt[i]);
    printf("\n");
}

void swap(int *xp, int *yp)
```

```

    {
        int temp = *xp;
        *xp = *yp;
        *yp = temp;
    }
void bubbleSort(int bt[], int n)
{
    int i, j;
    for(i = 0; i < n-1; i++)
    {
        for (j = 0; j < n-i-1; j++)
            { if (bt[j] > bt[j+1])
                {
                    swap(&bt[j], &bt[j+1]);
                    swap(&pid[j], &pid[j+1]);
                    swap(&at[j], &at[j]);
                }
            }
    }
}
bubbleSort(bt,n);
bubbleSort(at,n);
for(int j=0;j<n;j++)
{
    sum+=bt[j];
    ct[j]+=sum;
}
for(int k=0;k<n;k++)
{
    tat[k]=ct[k]-at[k];
}
for(int k=0;k<n;k++)
{
    wt[k]=tat[k]-bt[k];

```

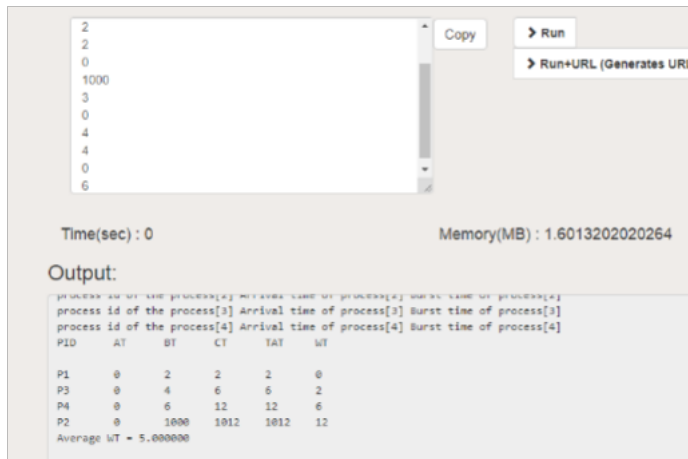


```

totalWT+=wt[k];
}
printf("PID\t AT\t BT\t CT\t TAT\t WT\t\n\n");
for(int i=0;i<n;i++)
{
printf("P%d\t %d\t %d\t %d\t %d\t %d\n",pid[i],at[i],bt[i],ct[i],tat[i],wt[i]);
}
printf("Average WT = %f\n\n",totalWT/n);
return 0;
}

```

OUTPUT :-



The screenshot shows an online code editor with the following components:

- Code Editor:** Contains the C program code for process scheduling.
- Input Area:** A text area with the following input:
 

```

2
2
0
1000
3
0
4
4
0
6
      
```
- Buttons:** "Copy", "Run", and "Run+URL (Generates URI)".
- Execution Metrics:**
  - Time(sec) : 0
  - Memory(MB) : 1.6013202020264
- Output:** A text area displaying the program's output:
 

```

process id of the process[3] Arrival time of process[3] Burst time of process[3]
process id of the process[4] Arrival time of process[4] Burst time of process[4]
PID      AT      BT      CT      TAT     WT
P1       0       2       2       2       0
P3       0       4       6       6       2
P4       0       6       12      12       6
P2       0      1000    1012    1012    12
Average WT = 5.000000
      
```