

# Operating System Lab Observation

**1a**

**Program:**

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

```
#include <unistd.h>
```

```
#include <fcntl.h>
```

```
int main()
```

```
{
```

```
    int fd;
```

```
    char buffer[50];
```

```
    /* open() – create and open a file */
```

```
    fd = open("sample.txt", O_CREAT | O_RDWR, 0644);
```

```
    if (fd < 0)
```

```
    {
```

```
        printf("File opening failed\n");
```

```
        return 1;
```

```
    }
```

```
    /* write() – write data into file */
```

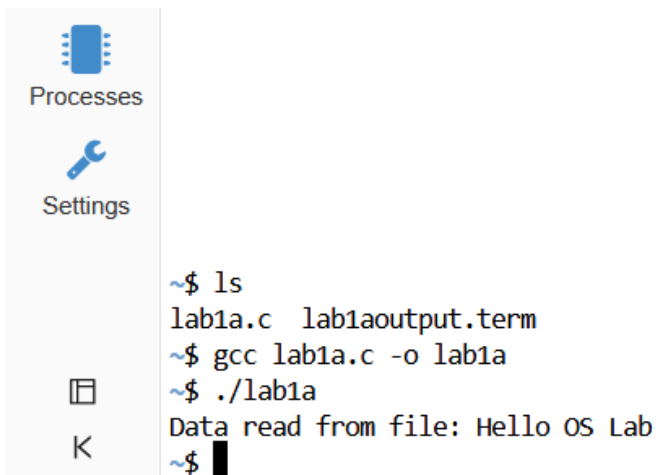
```
    write(fd, "Hello OS Lab", 12);
```

```
    /* move file pointer to beginning */
```

```
    lseek(fd, 0, SEEK_SET);
```

```
/* read() – read data from file */  
read(fd, buffer, 12);  
  
/* display data */  
printf("Data read from file: %s\n", buffer);  
  
/* close() – close the file */  
close(fd);  
  
return 0;  
}
```

### Output:



```
~$ ls  
lab1a.c  lab1aoutput.term  
~$ gcc lab1a.c -o lab1a  
~$ ./lab1a  
Data read from file: Hello OS Lab  
~$
```

**1b:**

**Program:**

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

```
#include <dirent.h>
```

```
int main()
```

```
{
```

```
    DIR *d;
```

```
    struct dirent *entry;
```

```
    /* open current directory */
```

```
    d = opendir(".");
```

```
    if (d == NULL)
```

```
    {
```

```
        printf("Directory cannot be opened\n");
```

```
        return 1;
```

```
    }
```

```
    /* read directory entries */
```

```
    printf("Directory contents:\n");
```

```
    while ((entry = readdir(d)) != NULL)
```

```
    {
```

```
        printf("%s\n", entry->d_name);
```

```
    }
```

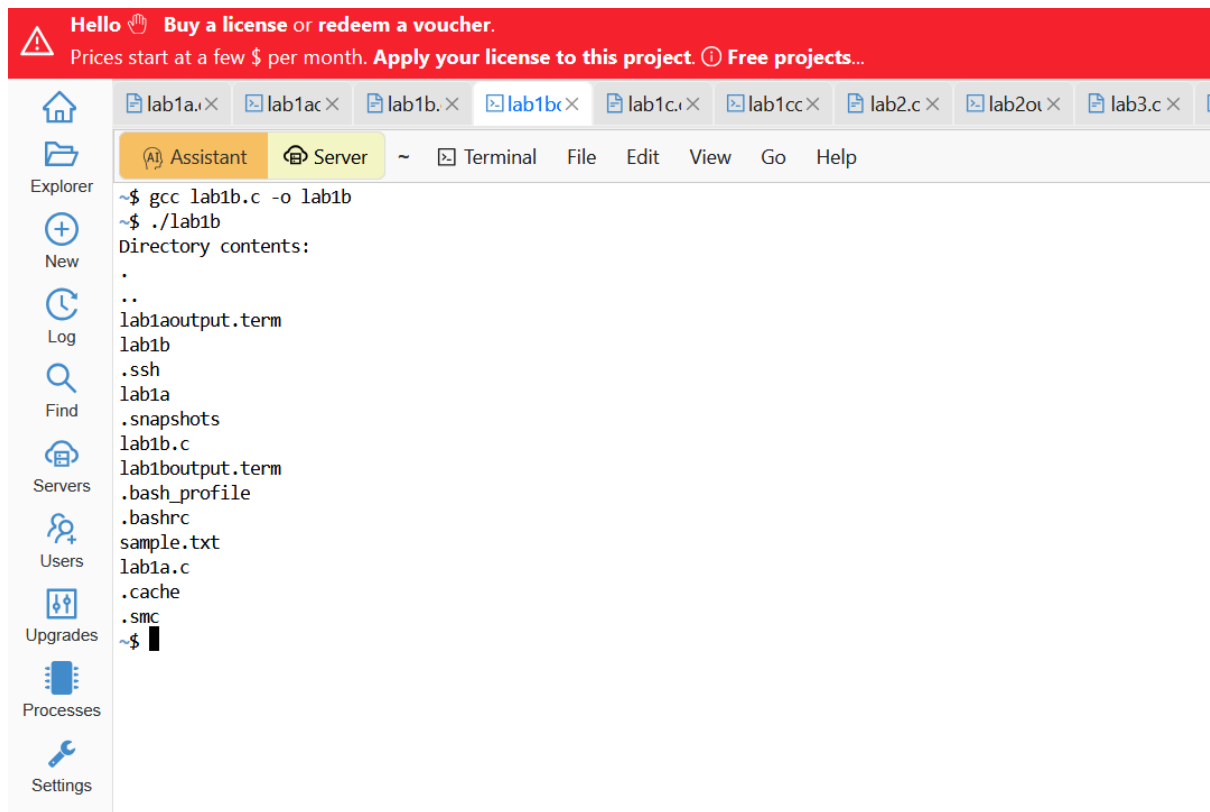
```
    /* close directory */
```

```
    closedir(d);
```

```
    return 0;
```

```
}
```

## Output:



The screenshot shows a code editor with a red header bar containing a warning icon and text: "Hello Buy a license or redeem a voucher. Prices start at a few \$ per month. Apply your license to this project. Free projects...". Below the header, there are several tabs for files: lab1a.c, lab1ac, lab1b, lab1bc, lab1c, lab1cc, lab2.c, lab2ot, and lab3.c. The active tab is lab1bc. The editor has a menu bar with options: Assistant, Server, ~, Terminal, File, Edit, View, Go, and Help. On the left side, there is a sidebar with icons for Explorer, New, Log, Find, Servers, Users, Upgrades, Processes, and Settings. The main area shows a terminal window with the following output:

```
~$ gcc lab1b.c -o lab1b
~$ ./lab1b
Directory contents:
.
..
lab1aoutput.term
lab1b
.ssh
lab1a
.ssh
lab1b.c
lab1boutput.term
.bash_profile
.bashrc
sample.txt
lab1a.c
.cache
.smc
~$
```

## 1c:

### Program:

```
#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

void main()

{

printf("\nBefore declaring FORK.");

printf("\nThe value of PID is : ");

printf("\n%d",getpid());

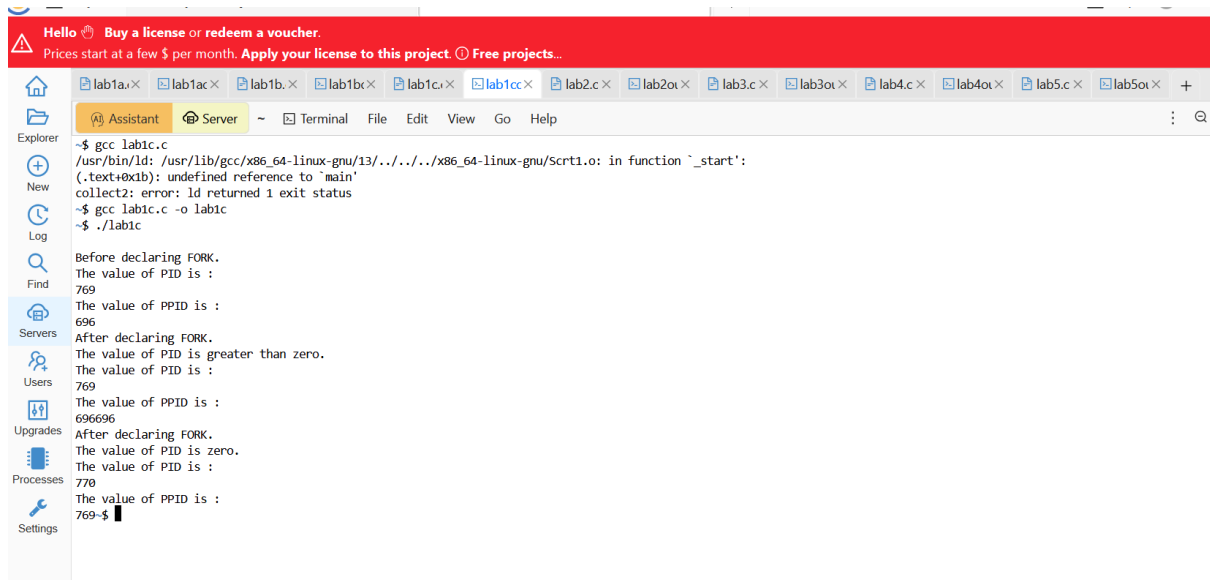
printf("\nThe value of PPID is : ");



printf("\n%d",getppid());
```

```
pid_t pid=fork();//Used to create child process
if(pid==0)//Child Process
{
printf("\nAfter declaring FORK.");
printf("\nThe value of PID is zero.");
printf("\nThe value of PID is : ");
printf("\n%d",getpid());
printf("\nThe value of PPID is : ");
printf("\n%d",getppid());
}
else if(pid>0)//Parent Process
{
printf("\nAfter declaring FORK.");
printf("\nThe value of PID is greater than zero.");
printf("\nThe value of PID is : ");
printf("\n%d",getpid());
printf("\nThe value of PPID is : ");
printf("\n%d",getppid());
}
else//Error
{
printf("\nAfter declaring FORK.");
printf("\nThe value of PID is less than zero.");
printf("\nThe value of PID is : ");
printf("\n%d",getpid());
printf("\nThe value of PPID is : ");
printf("\n%d",getppid());
}
```

```
}
```

## Output:



```
Hello  Buy a license or redeem a voucher.
Prices start at a few $ per month. Apply your license to this project.  Free projects...

lab1a.x lab1ac.x lab1b.x lab1bx.x lab1cx.x lab1cc.x lab2.c.x lab2ox.x lab3.c.x lab3ox.x lab4.c.x lab4ox.x lab5.c.x lab5ox.x +
Assistant Server ~ Terminal File Edit View Go Help

Explorer
New
Log
Find
Servers
Users
Upgrades
Processes
Settings

~$ gcc lab1c.c
/usr/bin/ld: /usr/lib/gcc/x86_64-linux-gnu/13/../../../../x86_64-linux-gnu/Scrt1.o: in function `_start':
(.text+0x1b): undefined reference to `main'
collect2: error: ld returned 1 exit status
~$ gcc lab1c.c -o lab1c
~$ ./lab1c
Before declaring FORK.
The value of PID is :
769
The value of PPID is :
696
After declaring FORK.
The value of PID is greater than zero.
The value of PID is :
769
The value of PPID is :
696696
After declaring FORK.
The value of PID is zero.
The value of PID is :
770
The value of PPID is :
769-$
```

## Lab2:

### Program:

```
#include <stdio.h>

// FCFS Scheduling

void fcfs(int n, int bt[]) {

int wt[10], tat[10];

wt[0] = 0;

for(int i = 1; i < n; i++)

wt[i] = wt[i-1] + bt[i-1];

printf("\nFCFS Scheduling:\n");

printf("Process\tBurst Time\tWaiting Time\tTurnaround Time\n");

for(int i = 0; i < n; i++) {
```

```

tat[i] = wt[i] + bt[i];
printf("P%d\tt%d\tt%d\tt%d\n", i+1, bt[i], wt[i], tat[i]);
}
}

// Round Robin Scheduling
void roundRobin(int n, int bt[], int tq) {
    int rem_bt[10], wt[10], tat[10];
    int t = 0;
    for(int i = 0; i < n; i++) {
        rem_bt[i] = bt[i];
        wt[i] = 0;
    }
    while(1) {
        int done = 1;
        for(int i = 0; i < n; i++) {
            if(rem_bt[i] > 0) {
                done = 0;
                if(rem_bt[i] > tq) {
                    t += tq;
                    rem_bt[i] -= tq;
                } else {
                    t += rem_bt[i];
                    wt[i] = t - bt[i];
                    rem_bt[i] = 0;
                }
            }
        }
    }
    if(done == 1)

```

```

        break;
    }

    printf("\nRound Robin Scheduling (Time Quantum = %d):\n", tq);
    printf("Process\tBurst Time\tWaiting Time\tTurnaround Time\n");

    for(int i = 0; i < n; i++) {
        tat[i] = bt[i] + wt[i];
        printf("P%d\t\t%d\t\t%d\t\t%d\n", i+1, bt[i], wt[i], tat[i]);
    }
}

int main() {
    int n, tq;
    int bt[10];

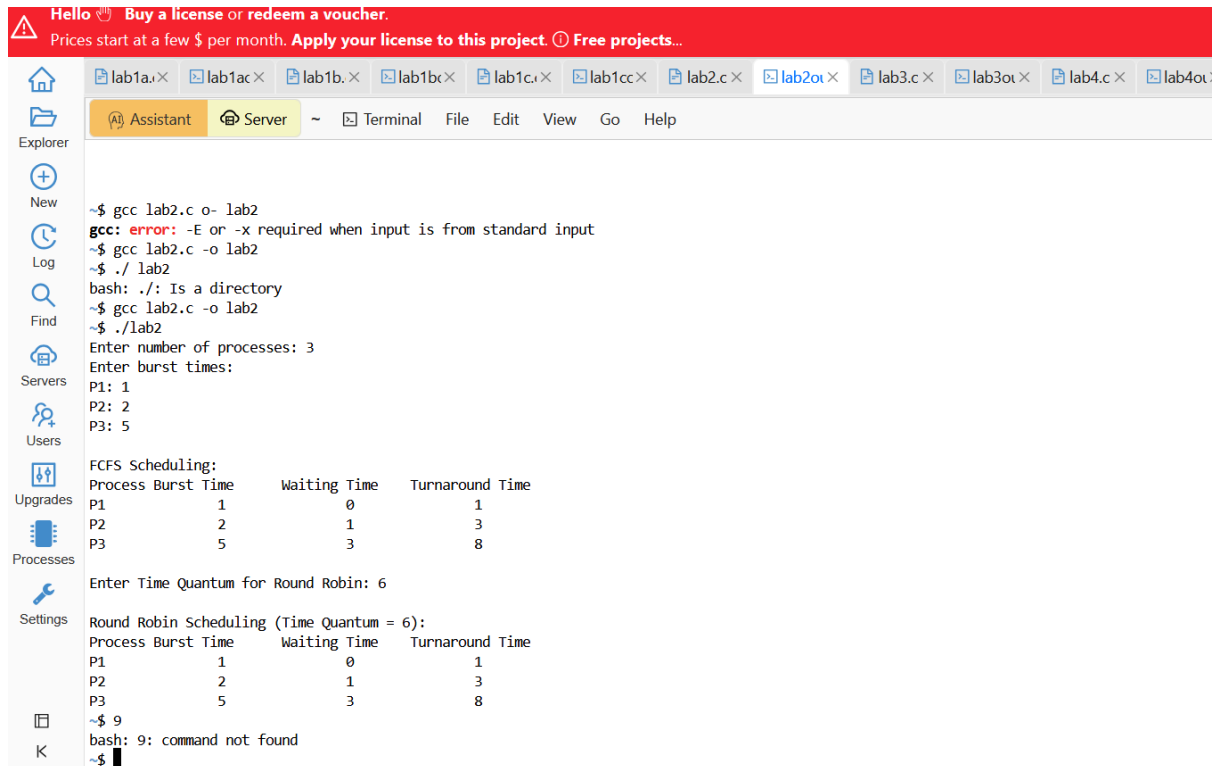
    printf("Enter number of processes: ");
    scanf("%d", &n);
    printf("Enter burst times:\n");
    for(int i = 0; i < n; i++) {
        printf("P%d: ", i+1);
        scanf("%d", &bt[i]);
    }
    fcfs(n, bt);
    printf("\nEnter Time Quantum for Round Robin: ");
    scanf("%d", &tq);
    roundRobin(n, bt, tq);
    return 0;
}

```



}

## Output:



```
~$ gcc lab2.c -o lab2
gcc: error: -E or -x required when input is from standard input
~$ gcc lab2.c -o lab2
~$ ./lab2
bash: ./: Is a directory
~$ gcc lab2.c -o lab2
~$ ./lab2
Enter number of processes: 3
Enter burst times:
P1: 1
P2: 2
P3: 5

FCFS Scheduling:
Process Burst Time    Waiting Time    Turnaround Time
P1                    1              0              1
P2                    2              1              3
P3                    5              3              8

Enter Time Quantum for Round Robin: 6

Round Robin Scheduling (Time Quantum = 6):
Process Burst Time    Waiting Time    Turnaround Time
P1                    1              0              1
P2                    2              1              3
P3                    5              3              8

~$ 9
bash: 9: command not found
~$
```

## Lab3:

### Program:

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

```
int main(){
```

```
int n, i, j;
```

```
int bt[10], pr[10], p[10];
```

```
int wt[10], tat[10], temp;
```

```
printf("Enter number of processes: ");
```

```
scanf("%d", &n);
```

```
for(i = 0; i < n; i++) {
```

```
    p[i] = i + 1;
```

```
    printf\
```

```
    Enter Burst Time and Priority for P%d: ", p[i]);
```

```
    scanf("%d %d", &bt[i], &pr[i]);
```

```
}
```

```
// Sort by Priority
```

```
for(i = 0; i < n; i++) {
```

```
    for(j = i + 1; j < n; j++) {
```

```
        if(pr[i] > pr[j]) {
```

```
            temp = pr[i]; pr[i] = pr[j]; pr[j] = temp;
```

```
            temp = bt[i]; bt[i] = bt[j]; bt[j] = temp;
```

```
            temp = p[i]; p[i] = p[j]; p[j] = temp;
```

```
        }
```

```
    }
```

```
}
```

```
wt[0] = 0;
```

```
for(i = 1; i < n; i++)
```

```
    wt[i] = wt[i - 1] + bt[i - 1];
```

```
for(i = 0; i < n; i++)
```

```
    tat[i] = wt[i] + bt[i];
```

```
printf("\nProcess\tBT\tPriority\tWT\tTAT\n");
```

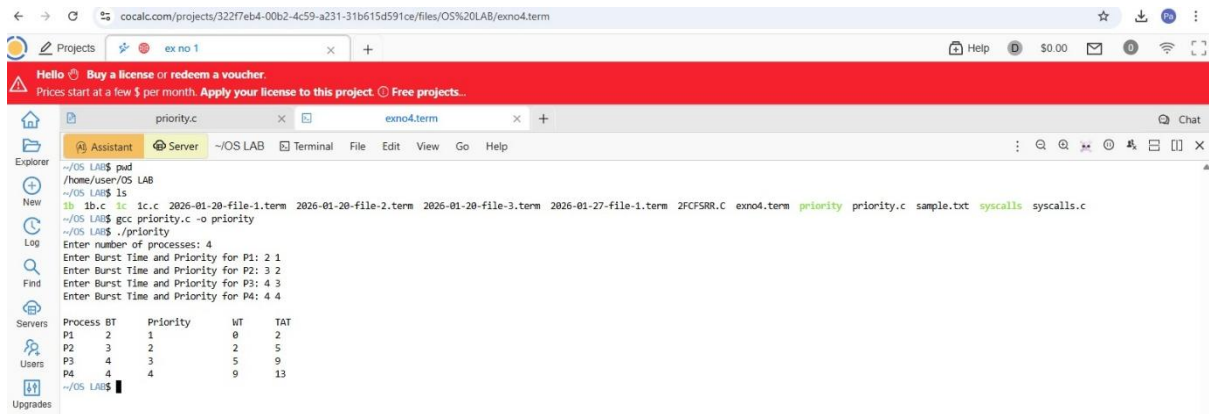
```
for(i = 0; i < n; i++)
```

```
    printf("P%d\t%d\t%d\t%d\t%d\n", p[i], bt[i], pr[i], wt[i], tat[i]);
```

return 0;

}

## **Output:**



```
~/OS LAB$ pad
~/home/user/OS LAB
~/OS LAB$ ls
1b 1b.c 1c 1c.c 2026-01-20-file-1.term 2026-01-20-file-2.term 2026-01-20-file-3.term 2026-01-27-file-1.term 2FCF5RR.C exno4.term priority priority.c sample.txt syscalls syscalls.c
~/OS LAB$ gcc priority.c -o priority
~/OS LAB$ ./priority
Enter number of processes: 4
Enter Burst Time and Priority for P1: 2 1
Enter Burst Time and Priority for P2: 3 2
Enter Burst Time and Priority for P3: 4 3
Enter Burst Time and Priority for P4: 4 4

Process BT Priority WT TAT
P1 2 1 0 2
P2 3 2 2 5
P3 4 3 5 9
P4 4 4 9 13
~/OS LAB$
```

## **Lab4:**

### **Program:**

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

#define N 5

sem\_t forks[N];

pthread\_t philosophers[N];

void\* philosopher(void\* num){

int id = \*(int\*)num;

printf("Philosopher %d is thinking\n", id);

```

sleep(1);

sem_wait(&forks[id]);

sem_wait(&forks[(id + 1) % N]);

printf("Philosopher %d is eating\n", id);

sleep(1);

sem_post(&forks[id]);

sem_post(&forks[(id + 1) % N]);

printf("Philosopher %d finished eating\n", id);

return NULL;
}

int main() {
    int i, id[N];

    for (i = 0; i < N; i++)
        sem_init(&forks[i], 0, 1);

    for (i = 0; i < N; i++) {
        id[i] = i;

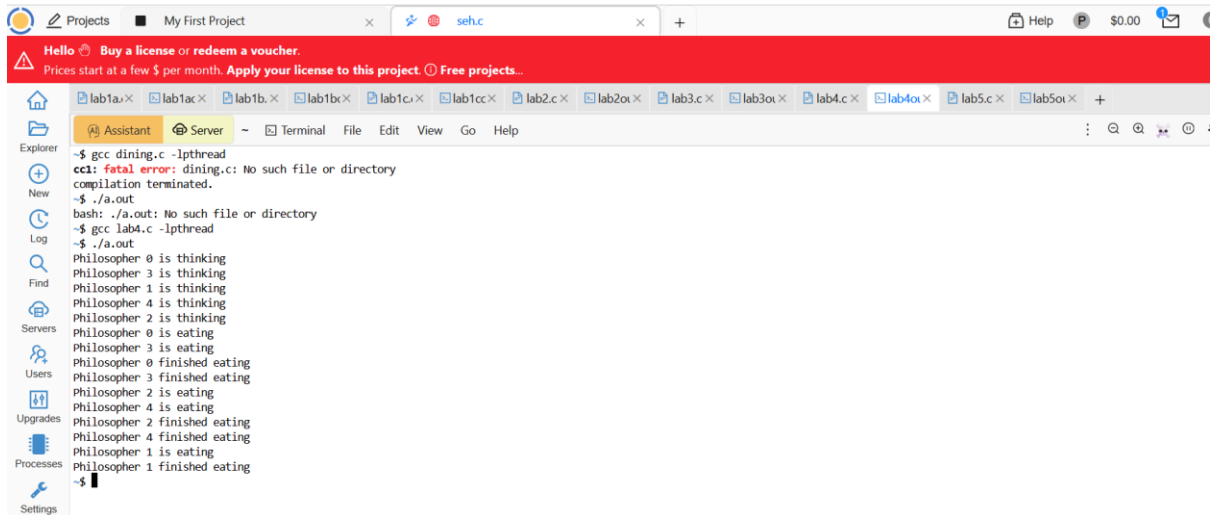
        pthread_create(&philosophers[i], NULL, philosopher, &id[i]);
    }

    for (i = 0; i < N; i++)
        pthread_join(philosophers[i], NULL);

    return 0;
}

```

**Output:**



## Lab5:

### Program:

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

```
void firstFit(int blockSize[], int m, int processSize[], int n)
```

```
{
```

```
int allocation[n];
```

```
for(int i=0;i<n;i++)
```

```
allocation[i]=-1;
```

```
for(int i=0;i<n;i++)
```

```
{
```

```
for(int j=0;j<m;j++)
```

```
{
```

```
if(blockSize[j] >= processSize[i])
```

```
{
```

```
allocation[i] = j;
```

```
blockSize[j] -= processSize[i];
```

```
break;
```

```
____}  
____}  
____}
```

```
____prin ("\nFIRST FIT\nProcess No.\tProcess Size\tBlock No.\n");  
____for(int i=0;i<n;i++)  
____{  
____    if(allocation[i]!=-1)  
____        prin ("%d\t\t%d\t\t%d\n",i+1,processSize[i],allocation[i]+1);  
____    else  
____        prin ("%d\t\t%d\t\t\tNot Allocated\n",i+1,processSize[i]);  
____}  
____}
```

```
void bestFit(int blockSize[], int m, int processSize[], int n)  
{  
____    int allocation[n];  
____    for(int i=0;i<n;i++)  
____        allocation[i]=-1;  
  
____    for(int i=0;i<n;i++)  
____    {  
____        int bestIdx=-1;  
____        for(int j=0;j<m;j++)  
____        {  
____            if(blockSize[j]>=processSize[i])  
____            {  
____                if(bestIdx==-1 || blockSize[j]<blockSize[bestIdx])
```

```

        bestIdx=j;
    }
}

if(bestIdx!=-1)
{
    allocation[i]=bestIdx;
    blockSize[bestIdx]-=processSize[i];
}
}

prin ("\nBEST FIT\nProcess No.\tProcess Size\tBlock No.\n");
for(int i=0;i<n;i++)
{
    if(allocation[i]!=-1)
        prin ("%d\t\t%d\t\t%d\n",i+1,processSize[i],allocation[i]+1);
    else
        prin ("%d\t\t%d\t\tNot Allocated\n",i+1,processSize[i]);
}
}

int main()
{
    int blockSize[] = {100, 500, 200, 300, 600};
    int processSize[] = {212, 417, 112, 426};

    int m = 5;
    int n = 4;

```

```

    int block1[5], block2[5];

    for(int i=0;i<m;i++)
    {
        block1[i]=blockSize[i];
        block2[i]=blockSize[i];
    }

    firstFit(block1,m,processSize,n);
    bestFit(block2,m,processSize,n);

    return 0;
}

```

## Output:

The screenshot shows a web-based IDE with a terminal window. The terminal output is as follows:

```

~/OS LAB$ ls
1b.c  1c.c  2026-01-20-file-1.term  2026-01-20-file-2.term  2026-01-20-file-3.term  2026-01-27-file-1.term  2026-01-27-file-2.term  2FCFSRR.C  'EX NO 5.term'  exno4.term  priority  sample.txt  syscalls.c
~/OS LAB$ cd
~/OS LAB$ gcc memoryalloc.c -o memoryalloc
~/OS LAB$ ./memoryalloc

```

The output then displays the results of the 'FIRST FIT' and 'BEST FIT' algorithms:

**FIRST FIT**

Process No.	Process Size	Block No.
1	212	2
2	417	5
3	112	2
4	426	Not Allocated

**BEST FIT**

Process No.	Process Size	Block No.
1	212	4
2	417	2
3	112	3
4	426	5