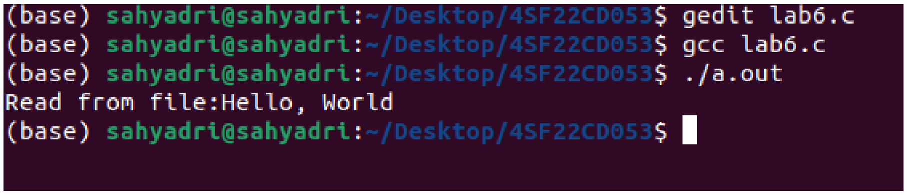
EXPERIMENT 1:- 1.Write C programs to implement basic UNIX system calls - read(), write(), open(),close(), lseek(), create().

Program:-

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>

int main() {
    int fd, ret;
    char buffer[20];

    // Create a new file
    fd = creat("example.txt", 0644);
    if (fd == -1) {
        perror("creat");
        exit(EXIT_FAILURE);
    }

    // Write to the file
    ret = write(fd, "Hello, World!", 13);
    if (ret == -1) {
        perror("write");
        exit(EXIT_FAILURE);
    }

    // Close the file
    ret = close(fd);
    if (ret == -1) {
        perror("close");
        exit(EXIT_FAILURE);
    }


    // Open the file again
    fd = open("example.txt", O_RDWR);
    if (fd == -1) {
        perror("open");
        exit(EXIT_FAILURE);
    }


    // Move the file cursor to the beginning of the file
    ret = lseek(fd, 0, SEEK_SET);
    if (ret == -1) {
        perror("lseek");
        exit(EXIT_FAILURE);
    }


    // Read from the file
    ret = read(fd, buffer, 13);
    if (ret == -1) {
        perror("read");
        exit(EXIT_FAILURE);
    }
    buffer[ret] = '\0'; // Null-terminate the string

    printf("Read from file: %s\n", buffer);

    // Close the file
    ret = close(fd);
    if (ret == -1) {
        perror("close");
        exit(EXIT_FAILURE);
    }
    return 0;
}
```

Output:-



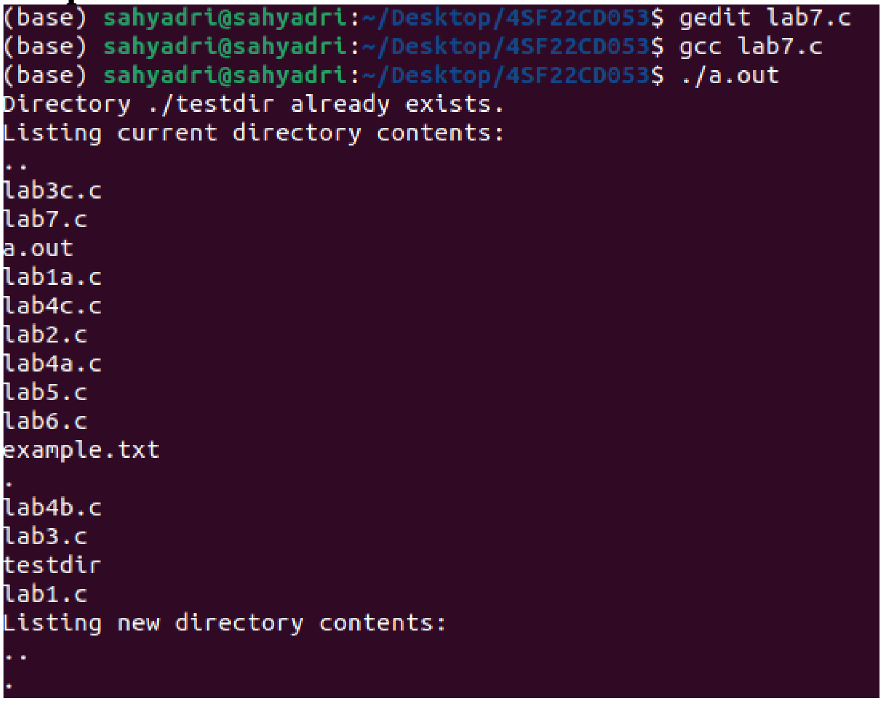
```
(base) sahyadri@sahyadri:~/Desktop/4SF22CD053$ gedit lab6.c
(base) sahyadri@sahyadri:~/Desktop/4SF22CD053$ gcc lab6.c
(base) sahyadri@sahyadri:~/Desktop/4SF22CD053$ ./a.out
Read from file:Hello, World
(base) sahyadri@sahyadri:~/Desktop/4SF22CD053$ 
```

EXPERIMENT 2:- Write C programs to implement UNIX Directory API's - opendir, closedir, readdir, mkdir.

Program:-

```
 #include <stdio.h>
#include <stdlib.h>
#include <dirent.h>
#include <sys/stat.h>
#include <errno.h>


void listDirectory(const char *path) {
    DIR *dir = opendir(path);
    if (dir == NULL) {
        perror("opendir");
        return;
    }

    struct dirent *entry;
    while ((entry = readdir(dir)) != NULL) {
        printf("%s\n", entry->d_name);
    }


    if (closedir(dir) == -1) {
        perror("closedir");
    }
}


void createDirectory(const char *path) {
    if (mkdir(path, 0755) == -1) {
        if (errno == EEXIST) {
            printf("Directory %s already exists.\n", path);
        } else {
            perror("mkdir");
        }
    } else {
        printf("Directory %s created successfully.\n", path);
    }
}


int main() {
    const char *dirPath = "./testdir";


    // Create a directory
    createDirectory(dirPath);


    // List the contents of the current directory
    printf("Listing current directory contents:\n");
    listDirectory(".");


    // List the contents of the new directory
    printf("\nListing new directory contents:\n");
    listDirectory(dirPath);


    return 0;
}
```

Output:-



```
(base) sahyadri@sahyadri:~/Desktop/4SF22CD053$ gedit lab7.c
(base) sahyadri@sahyadri:~/Desktop/4SF22CD053$ gcc lab7.c
(base) sahyadri@sahyadri:~/Desktop/4SF22CD053$ ./a.out
Directory ./testdir already exists.
Listing current directory contents:
..
lab3c.c
lab7.c
a.out
lab1a.c
lab4c.c
lab2.c
lab4a.c
lab5.c
lab6.c
example.txt
.
lab4b.c
lab3.c
testdir
lab1.c
Listing new directory contents:
..
.
```

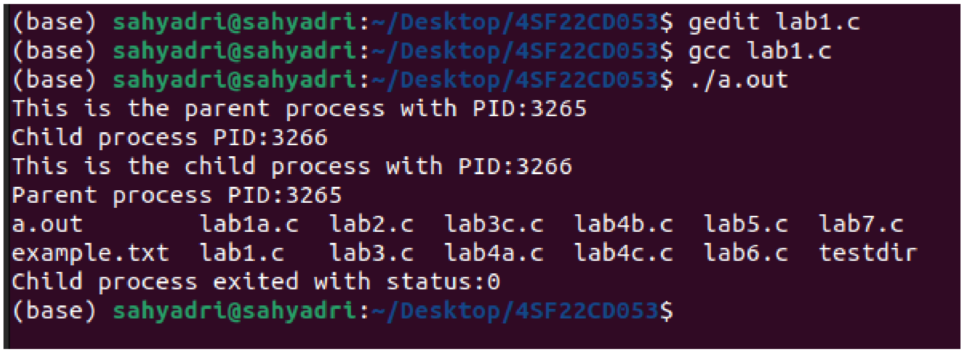
EXPERIMENT 3:-Demonstrate the Process creation and Termination using System calls –fork (), vfork (), exit (), return 0.

Program:-

a. fork():-

```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
    pid_t pid;
    int status;

    // fork() system call
    pid = fork();

    if (pid < 0) {
        printf("Error: fork() failed.\n");
        return 1;
    } else if (pid == 0) {
        // child process

        printf("This is the child process with PID: %d\n", getpid());
        printf("Parent process PID: %d\n", getppid());

        // exec() system call
        execlp("/bin/ls", "ls", NULL);

        printf("This should not be printed if exec() is successful.\n");
        return 0;
    } else {
        // parent process


        printf("This is the parent process with PID: %d\n", getpid());
        printf("Child process PID: %d\n", pid);
        // wait() system call
        wait(&status);

        printf("Child process exited with status: %d\n", status);

        return 0;
    }
}
```

Output:-

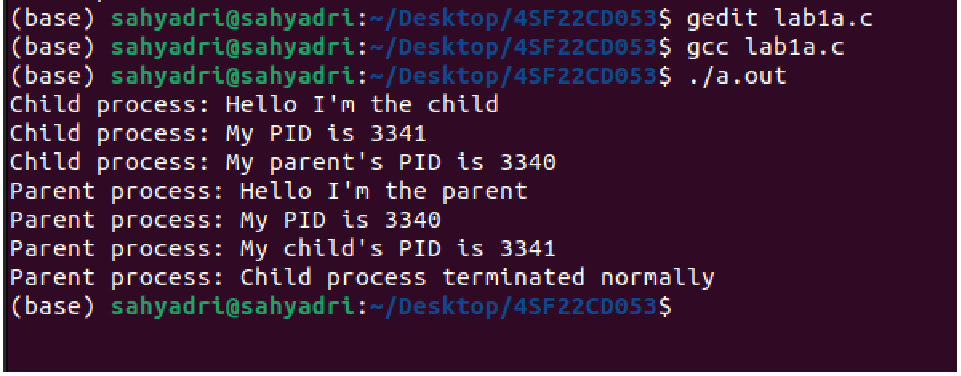


```
(base) sahyadri@sahyadri:~/Desktop/4SF22CD053$ gedit lab1.c
(base) sahyadri@sahyadri:~/Desktop/4SF22CD053$ gcc lab1.c
(base) sahyadri@sahyadri:~/Desktop/4SF22CD053$ ./a.out
This is the parent process with PID:3265
Child process PID:3266
This is the child process with PID:3266
Parent process PID:3265
a.out        lab1a.c  lab2.c  lab3c.c  lab4b.c  lab5.c  lab7.c
example.txt  lab1.c   lab3.c  lab4a.c  lab4c.c  lab6.c  testdir
Child process exited with status:0
(base) sahyadri@sahyadri:~/Desktop/4SF22CD053$
```

b. vfork ():-

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int main() {
    pid_t pid;

    // Fork a child process using vfork()
    pid = vfork();

    if (pid == -1) {
        // Forking failed
        perror("vfork");
        return 1;
    } else if (pid == 0) {
        // Child process

        printf("Child process: Hello, I'm the child!\n");
        printf("Child process: My PID is %d\n", getpid());
        printf("Child process: My parent's PID is %d\n", getppid());

        // Terminate the child process
        _exit(0);
    } else {
        // Parent process

        printf("Parent process: Hello, I'm the parent!\n");
        printf("Parent process: My PID is %d\n", getpid());
        printf("Parent process: My child's PID is %d\n", pid);

        // Wait for the child process to terminate
        int status;
        waitpid(pid, &status, 0);

        if (WIFEXITED(status)) {
            printf("Parent process: Child process terminated normally.\n");
        } else {
            printf("Parent process: Child process terminated abnormally.\n");
        }
    }
return 0;
}
```

Output:-



```
(base) sahyadri@sahyadri:~/Desktop/4SF22CD053$ gedit lab1a.c
(base) sahyadri@sahyadri:~/Desktop/4SF22CD053$ gcc lab1a.c
(base) sahyadri@sahyadri:~/Desktop/4SF22CD053$ ./a.out
Child process: Hello I'm the child
Child process: My PID is 3341
Child process: My parent's PID is 3340
Parent process: Hello I'm the parent
Parent process: My PID is 3340
Parent process: My child's PID is 3341
Parent process: Child process terminated normally
(base) sahyadri@sahyadri:~/Desktop/4SF22CD053$
```

EXPERIMENT 4:- Write C programs to simulate Inter – Process Communication (IPC) techniques: Pipes, Messages Queues, and Shared Memory.

Program:-

```
#include <fcntl.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <unistd.h>


int main()

{

int fd;

char * myfifo = "/tmp/myfifo"; /* create the FIFO (named pipe) */

mkfifo(myfifo, 0666);


fd = open(myfifo, O_WRONLY);

write(fd,"Hi", sizeof("Hi")); /* write "Hi" to the FIFO */

close(fd);


unlink(myfifo); /* remove the FIFO */

return 0;

}
```

```
[root@localhost /]# gedit reader.c
```

Reader Process (reader.c)

```
 #include <fcntl.h>
#include <sys/stat.h>
 #include <unistd.h>

 #define MAX_BUF 1024


 int main()
 {
 int fd;
 char *myfifo = "/tmp/myfifo";
 char buf[MAX_BUF];


 /* open, read, and display the message from the FIFO */
 fd = open(myfifo, O_RDONLY);
 read(fd, buf, MAX_BUF);
 printf("Received: %s", buf);
 close(fd);
 return 0;
 }
```

Output:-

Receiver:Hi

EXPERIMENT 5:- Simulate the following CPU scheduling algorithms 1. FCFS 2. SJF 3. Priority 4. Round Robin 5. SRTF then calculate average waiting time, average Turn-around Time, Average Response time.

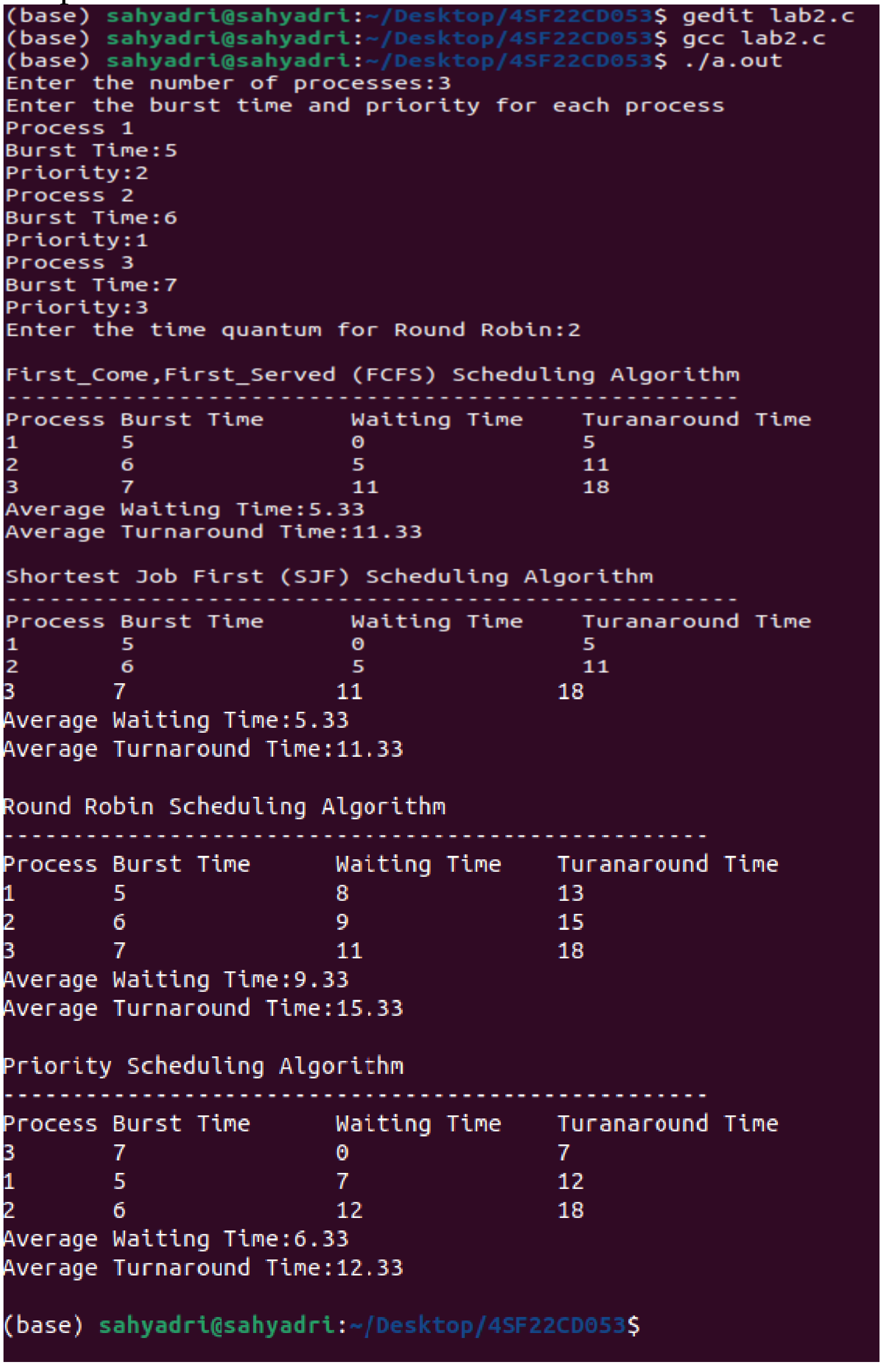
Program:-

```
#include <stdio.h>

 void fcfs(int processes[], int n, int burst_time[]) {
 int waiting_time[n], turnaround_time[n], total_waiting_time = 0,
total_turnaround_time   = 0;
    waiting_time[0] = 0; // Waiting time for first process is 0

    // Calculating waiting time for each process
    for (int i = 1; i < n; i++) {
        waiting_time[i] = burst_time[i - 1] + waiting_time[i - 1];
        total_waiting_time += waiting_time[i];
    }

    // Calculating turnaround time for each process
    for (int i = 0; i < n; i++) {
        turnaround_time[i] = burst_time[i] + waiting_time[i];
        total_turnaround_time += turnaround_time[i];
    }

    printf("First-Come, First-Served (FCFS) Scheduling Algorithm\n");
    printf("--------------------------------------------------\n");
    printf("Process\tBurst Time\tWaiting Time\tTurnaround Time\n");

    // Printing process details
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\n", processes[i], burst_time[i],
waiting_time[i],  turnaround_time[i]);
    }

    printf("Average Waiting Time: %.2f\n", (float)total_waiting_time / n);
    printf("Average Turnaround Time: %.2f\n",
(float)total_turnaround_time / n);
    printf("\n");
 }

 void sjf(int processes[], int n, int burst_time[]) {
    int waiting_time[n], turnaround_time[n], completion_time[n],
total_waiting_time = 0,  total_turnaround_time = 0;

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

        // Find the shortest job
        for (int j = i + 1; j < n; j++) {
            if (burst_time[j] < burst_time[shortest_job_index])
               shortest_job_index = j;
        }

       // Swap the shortest job with the current process
       int temp = burst_time[i];
       burst_time[i] = burst_time[shortest_job_index];
       burst_time[shortest_job_index] = temp;

       temp = processes[i];
       processes[i] = processes[shortest_job_index];
       processes[shortest_job_index] = temp;
    }

waiting_time[0] = 0; // Waiting time for first process is 0

    // Calculating waiting time for each process
    for (int i = 1; i < n; i++) {
       waiting_time[i] = burst_time[i - 1] + waiting_time[i - 1];
       total_waiting_time += waiting_time[i];
    }

    // Calculating turnaround time for each process
    for (int i = 0; i < n; i++) {
       turnaround_time[i] = burst_time[i] + waiting_time[i];
       total_turnaround_time += turnaround_time[i];
    }
    printf("Shortest Job First (SJF) Scheduling Algorithm\n");
    printf("-----------------------------------------------\n");
    printf("Process\tBurst Time\tWaiting Time\tTurnaround Time\n");

    // Printing process details
        for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\n", processes[i], burst_time[i],
waiting_time[i], turnaround_time[i]);
    }

    printf("Average Waiting Time: %.2f\n", (float)total_waiting_time / n);
    printf("Average Turnaround Time: %.2f\n", (float)total_turnaround_time / n);
    printf("\n");
}

 void roundRobin(int processes[], int n, int burst_time[], int quantum) {
     int remaining_time[n], waiting_time[n], turnaround_time[n],
total_waiting_time = 0,  total_turnaround_time = 0;

    // Copying burst time into remaining time array
    for (int i = 0; i < n; i++) {
        remaining_time[i] = burst_time[i];
    }

    int time = 0; // Current time

    // Run the round robin algorithm
    while (1) {
        int all_processes_completed = 1;

        // Traverse all processes

   for (int i = 0; i < n; i++) {
            if (remaining_time[i] > 0) {
                all_processes_completed = 0; // There is still a pending process

                if (remaining_time[i] > quantum) {
                    time += quantum;
                    remaining_time[i] -= quantum;
                } else {
                    time += remaining_time[i];
                    waiting_time[i] = time - burst_time[i];
                    remaining_time[i] = 0;
                }
            }
        }

        if (all_processes_completed) {
            break;
        }
    }

    // Calculating turnaround time for each process
    for (int i = 0; i < n; i++) {
        turnaround_time[i] = burst_time[i] + waiting_time[i];
        total_waiting_time += waiting_time[i];
        total_turnaround_time += turnaround_time[i];
    }

    printf("Round Robin Scheduling Algorithm\n");
    printf("--------------------------------\n");
    printf("Process\tBurst Time\tWaiting Time\tTurnaround Time\n");

    // Printing process details
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\n", processes[i], burst_time[i], 
waiting_time[i], turnaround_time[i]);
    }

    printf("Average Waiting Time: %.2f\n", (float)total_waiting_time / n);
    printf("Average Turnaround Time: %.2f\n", (float)total_turnaround_time / n);
    printf("\n");
}


void priorityy(int processes[], int n, int burst_time[], int priority[]) {
    int waiting_time[n], turnaround_time[n], total_waiting_time = 0, 
total_turnaround_time = 0;
```

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

        // Find the highest priority job
        for (int j = i + 1; j < n; j++) {
            if (priority[j] < priority[highest_priority_index])
                highest_priority_index = j;
        }

        // Swap the highest priority job with the current process
        int temp = burst_time[i];
        burst_time[i] = burst_time[highest_priority_index];
        burst_time[highest_priority_index] = temp;
temp = processes[i];
        processes[i] = processes[highest_priority_index];
        processes[highest_priority_index] = temp;

        temp = priority[i];
        priority[i] = priority[highest_priority_index];
        priority[highest_priority_index] = temp;
    }

    waiting_time[0] = 0; // Waiting time for first process is 0

    // Calculating waiting time for each process
    for (int i = 1; i < n; i++) {
        waiting_time[i] = burst_time[i - 1] + waiting_time[i - 1];
        total_waiting_time += waiting_time[i];
    }

    // Calculating turnaround time for each process
    for (int i = 0; i < n; i++) {
        turnaround_time[i] = burst_time[i] + waiting_time[i];
        total_turnaround_time += turnaround_time[i];
    }

    printf("Priority Scheduling Algorithm\n");
    printf("----------------------------------------------\n");
    printf("Process\tBurst Time\tWaiting Time\tTurnaround Time\n");
    // Printing process details
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\n", processes[i], burst_time[i],
waiting_time[i], turnaround_time[i]);
    }

    printf("Average Waiting Time: %.2f\n", (float)total_waiting_time / n);
    printf("Average Turnaround Time: %.2f\n", (float)total_turnaround_time / n);
    printf("\n");
}

int main() {
    int n;

    printf("Enter the number of processes: ");
    scanf("%d", &n);

    int processes[n], burst_time[n], priority[n];

    printf("Enter the burst time and priority for each process:\n");
    for (int i = 0; i < n; i++) {
        printf("Process %d\n", i + 1);
        printf("Burst Time: ");
        scanf("%d", &burst_time[i]);
        printf("Priority: ");
        scanf("%d", &priority[i]);
        processes[i] = i + 1;
    }

    int quantum;
    printf("Enter the time quantum for Round Robin: ");
    scanf("%d", &quantum);

    printf("\n");
    fcfs(processes, n, burst_time);
    sjf(processes, n, burst_time);
    roundRobin(processes, n, burst_time, quantum);
    priorityy(processes, n, burst_time, priority);

    return 0;
```

}

Output:-



```
(base) sahyadri@sahyadri:~/Desktop/4SF22CD053$ gedit lab2.c
(base) sahyadri@sahyadri:~/Desktop/4SF22CD053$ gcc lab2.c
(base) sahyadri@sahyadri:~/Desktop/4SF22CD053$ ./a.out
Enter the number of processes:3
Enter the burst time and priority for each process
Process 1
Burst Time:5
Priority:2
Process 2
Burst Time:6
Priority:1
Process 3
Burst Time:7
Priority:3
Enter the time quantum for Round Robin:2

First_Come,First_Served (FCFS) Scheduling Algorithm
---------------------------------------------------
Process Burst Time      Waiting Time    Turanaround Time
1       5               0               5
2       6               5               11
3       7               11              18
Average Waiting Time:5.33
Average Turnaround Time:11.33

Shortest Job First (SJF) Scheduling Algorithm
---------------------------------------------------
Process Burst Time      Waiting Time    Turanaround Time
1       5               0               5
2       6               5               11
3       7               11              18
Average Waiting Time:5.33
Average Turnaround Time:11.33

Round Robin Scheduling Algorithm
---------------------------------------------------
Process Burst Time      Waiting Time    Turanaround Time
1       5               8               13
2       6               9               15
3       7               11              18
Average Waiting Time:9.33
Average Turnaround Time:15.33

Priority Scheduling Algorithm
---------------------------------------------------
Process Burst Time      Waiting Time    Turanaround Time
3       7               0               7
1       5               7               12
2       6               12              18
Average Waiting Time:6.33
Average Turnaround Time:12.33

(base) sahyadri@sahyadri:~/Desktop/4SF22CD053$
```

Experiment 6:- Demonstrate the following Classical problems of synchronization using semaphores.

a. Producer-Consumer
b. Dining Philosopher

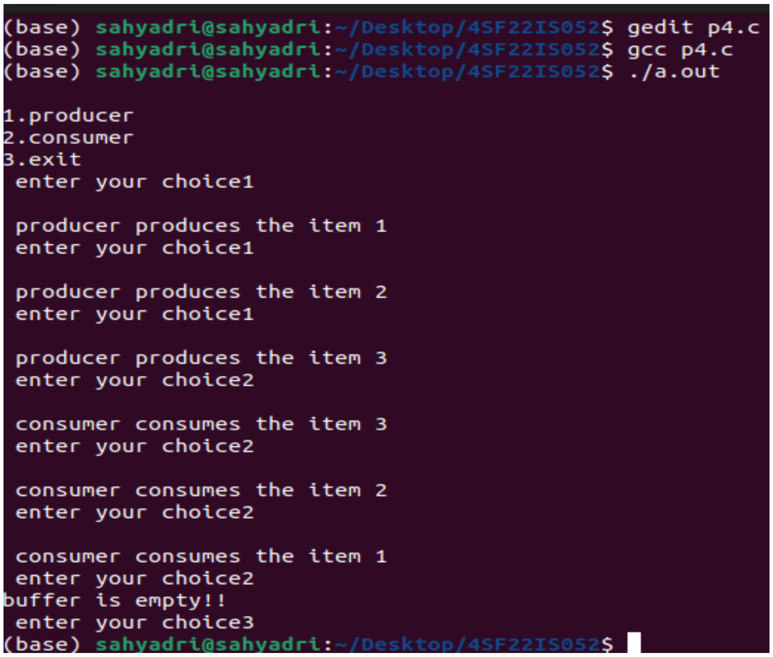
Program:-

```
#include<stdio.h>
#include<stdlib.h>
int mutex=1,full=0,empty=3,x=0;
int main()
{
int n;
void producer();
void consumer();
int wait(int);
int signal(int);
printf("\n1.producer\n2.consumer\n3.exit");
while(1)
{
printf("\n enter your choice");
scanf("%d",&n);
switch(n)
{
case 1:if((mutex==1)&&(empty!=0))
producer();
else
printf("buffer is full!!");
break;
case 2:if((mutex==1)&&(full!=0))
consumer();
else
printf("buffer is empty!!");
break;
case 3:
exit(0);
break;
}
}
return 0;
}
int wait(int s)
{
return(--s);
}
```

```
int signal(int s)
{
return(++s);
}
void producer()
{
mutex=wait(mutex);
full=signal(full);
empty=wait(empty);
x++;
printf("\n producer produces the item %d",x);
mutex=signal(mutex);
}
void consumer()
{
mutex=wait(mutex);
full=wait(full);
empty=signal(empty);
printf("\n consumer consumes the item %d",x);
x--;
mutex=signal(mutex);
}
```

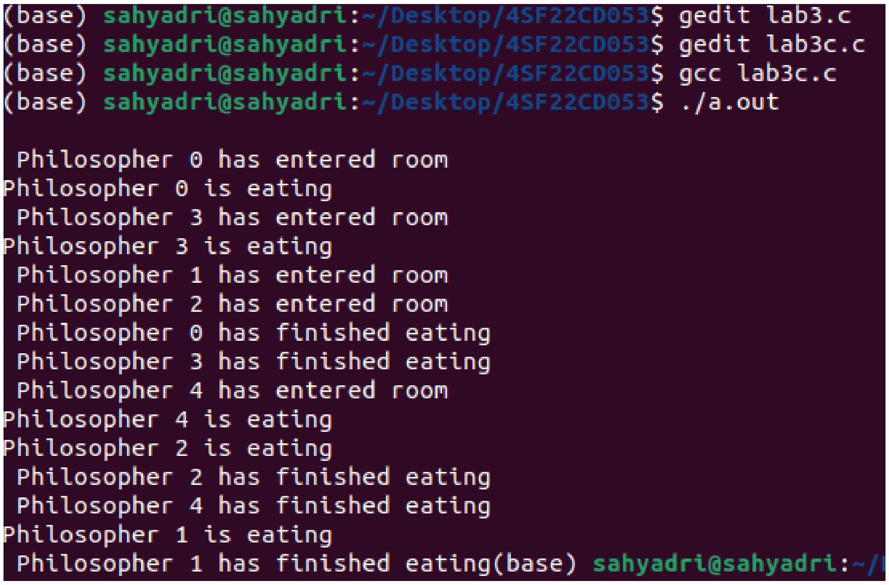
Output:-

Producer-Consumer



```
(base) sahyadri@sahyadri:~/Desktop/4SF22IS052$ gedit p4.c
(base) sahyadri@sahyadri:~/Desktop/4SF22IS052$ gcc p4.c
(base) sahyadri@sahyadri:~/Desktop/4SF22IS052$ ./a.out

1.producer
2.consumer
3.exit
 enter your choice1

 producer produces the item 1
 enter your choice1

 producer produces the item 2
 enter your choice1

 producer produces the item 3
 enter your choice2

 consumer consumes the item 3
 enter your choice2

 consumer consumes the item 2
 enter your choice2

 consumer consumes the item 1
 enter your choice2
buffer is empty!!
 enter your choice3
(base) sahyadri@sahyadri:~/Desktop/4SF22IS052$ 
```

Dining Philosopher:-



```
(base) sahyadri@sahyadri:~/Desktop/4SF22CD053$ gedit lab3.c
(base) sahyadri@sahyadri:~/Desktop/4SF22CD053$ gedit lab3c.c
(base) sahyadri@sahyadri:~/Desktop/4SF22CD053$ gcc lab3c.c
(base) sahyadri@sahyadri:~/Desktop/4SF22CD053$ ./a.out

 Philosopher 0 has entered room
Philosopher 0 is eating
 Philosopher 3 has entered room
Philosopher 3 is eating
 Philosopher 1 has entered room
 Philosopher 2 has entered room
 Philosopher 0 has finished eating
 Philosopher 3 has finished eating
 Philosopher 4 has entered room
Philosopher 4 is eating
Philosopher 2 is eating
 Philosopher 2 has finished eating
 Philosopher 4 has finished eating
Philosopher 1 is eating
 Philosopher 1 has finished eating(base) sahyadri@sahyadri:~/
```

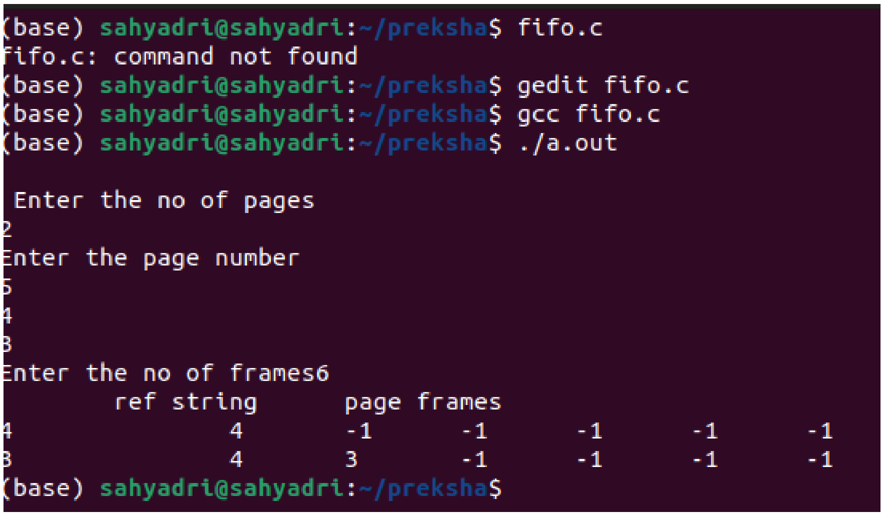
Experiment 7:- Demonstrate following page replacement algorithms:

a. FIFO, b. LRU, c. OPTIMAL.

a. FIFO:-

```
#include<stdio.h>
int main()
{
int i,j,n,a[50],frame[10],no,k,avail,count=0;
        printf("\n ENTER THE NUMBER OF PAGES:\n");
scanf("%d",&n);
        printf("\n ENTER THE PAGE NUMBER :\n");
        for(i=1;i<=n;i++)
        scanf("%d",&a[i]);
        printf("\n ENTER THE NUMBER OF FRAMES :");
        scanf("%d",&no);
for(i=0;i<no;i++)
        frame[i]= -1;
                j=0;
                printf("\tref string\t page frames\n");
for(i=1;i<=n;i++)
                {
                        printf("%d\t\t",a[i]);
                        avail=0;
                        for(k=0;k<no;k++)
if(frame[k]==a[i])
                                avail=1;
                        if (avail==0)
                        {
                                frame[j]=a[i];
                                j=(j+1)%no;
                                count++;
                                for(k=0;k<no;k++)
                                printf("%d\t",frame[k]);
}
                        printf("\n");
}
                printf("Page Fault Is %d",count);
                return 0;
}
```

Output:-



```
(base) sahyadri@sahyadri:~/preksha$ fifo.c
fifo.c: command not found
(base) sahyadri@sahyadri:~/preksha$ gedit fifo.c
(base) sahyadri@sahyadri:~/preksha$ gcc fifo.c
(base) sahyadri@sahyadri:~/preksha$ ./a.out

 Enter the no of pages
2
Enter the page number
5
4
3
Enter the no of frames6
        ref string       page frames
4               4        -1       -1       -1       -1       -1
3               4        3        -1       -1       -1       -1
(base) sahyadri@sahyadri:~/preksha$
```

LRU:-

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

int checkHit(int incomingPage, int queue[], int occupied){

    for(int i = 0; i < occupied; i++){
        if(incomingPage == queue[i])
            return 1;
    }

    return 0;
}

void printFrame(int queue[], int occupied)
{
    for(int i = 0; i < occupied; i++)
        printf("%d\t\t\t",queue[i]);
```

```
}

int main()
{

//    int incomingStream[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1};
//    int incomingStream[] = {1, 2, 3, 2, 1, 5, 2, 1, 6, 2, 5, 6, 3, 1, 3, 6, 1, 2, 4, 3};
    int incomingStream[] = {1, 2, 3, 2, 1, 5, 2, 1, 6, 2, 5, 6, 3, 1, 3};

    int n = sizeof(incomingStream)/sizeof(incomingStream[0]);
    int frames = 3;
    int queue[n];
    int distance[n];
    int occupied = 0;
    int pagefault = 0;

    printf("Page\t Frame1 \t Frame2 \t Frame3\n");

    for(int i = 0;i < n; i++)
    {
        printf("%d:  \t\t",incomingStream[i]);
        // what if currently in frame 7
        // next item that appears also 7
        // didnt write condition for HIT

        if(checkHit(incomingStream[i], queue, occupied)){
            printFrame(queue, occupied);
        }

        // filling when frame(s) is/are empty
        else if(occupied < frames){
            queue[occupied] = incomingStream[i];
            pagefault++;
            occupied++;

            printFrame(queue, occupied);
        }
        else{

            int max = INT_MIN;
            int index;
            // get LRU distance for each item in frame
            for (int j = 0; j < frames; j++)
            {
                distance[j] = 0;
                // traverse in reverse direction to find
                // at what distance  frame item occurred last
                for(int k = i - 1; k >= 0; k--)
                {
                    ++distance[j];

                    if(queue[j] == incomingStream[k])
                        break;
                }

                // find frame item with max distance for LRU
                // also notes the index of frame item in queue
                // which appears furthest(max distance)
                if(distance[j] > max){
                    max = distance[j];
                    index = j;
                }
            }
            queue[index] = incomingStream[i];
            printFrame(queue, occupied);
            pagefault++;
        }

        printf("\n");
    }

    printf("Page Fault: %d",pagefault);

    return 0;
}
```

Output:-

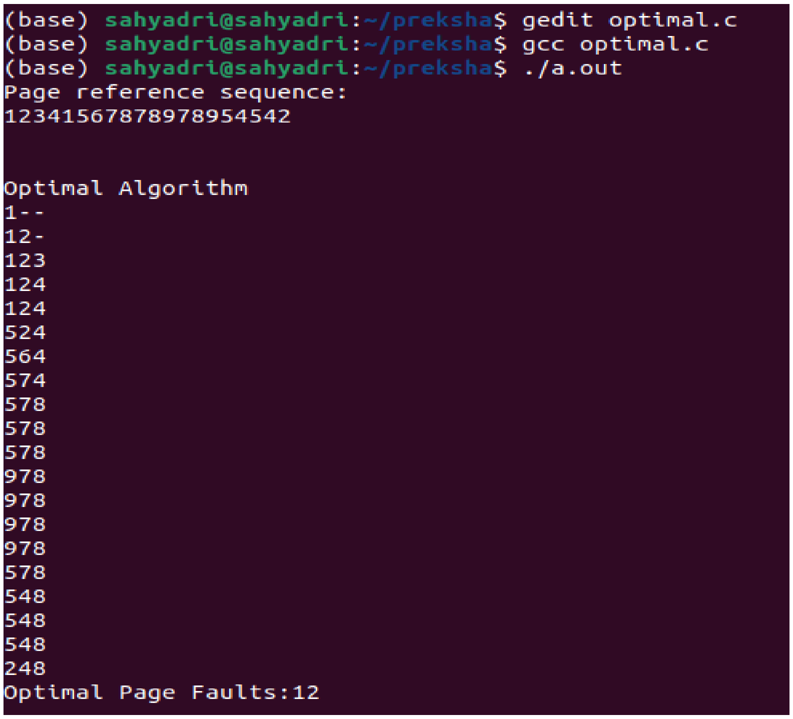
```
Page     Frame1          Frame2          Frame3
1:              1
2:              1                       2
3:              1                       2                       3
2:              1                       2                       3
1:              1                       2                       3
5:              1                       2                       5
2:              1                       2                       5
1:              1                       2                       5
6:              1                       2                       6
2:              1                       2                       6
5:              5                       2                       6
6:              5                       2                       6
3:              5                       3                       6
1:              1                       3                       6
3:              1                       3                       6
Page Fault: 8


=== Code Execution Successful ===
```

OPTIMAL:-

```
#include <stdio.h>
#include <stdbool.h>
// Function to find the index of the page in the frames
int findIndex(int frames[], int n, int page)
{
for (int i = 0; i < n; i++)
{
if (frames[i] == page)
return i;
}
return -1;
}
// Function to print the contents of the frames
void printFrames(int frames[], int n)
{
for (int i = 0; i < n; i++)
{
if (frames[i] == -1)
printf("- ");
else
printf("%d ", frames[i]);
}
printf("\n");
}
// OPTIMAL page replacement algorithm
void optimal(int pages[], int n, int capacity)
{
int frames[capacity];
int pageFaults = 0;
int index, farthest, futureIndex;
for (int i = 0; i < capacity; i++)
frames[i] = -1;
for (int i = 0; i < n; i++)
```

```
{
int page = pages[i];
index = findIndex(frames, capacity, page);
if (index == -1)
{
int emptyIndex = findIndex(frames, capacity, -1);
if (emptyIndex != -1)
{
frames[emptyIndex] = page;
}
else
{
farthest = i + 1;
futureIndex = -1;
for (int j = 0; j < capacity; j++)
{
int currentPage = frames[j];
int k;
for (k = i + 1; k < n; k++)

{
if (currentPage == pages[k])
{
if (k > farthest)
{
farthest = k;
futureIndex = j;
}
break;
}
}
if (k == n)
{
futureIndex = j;
break;
}
}
frames[futureIndex] = page;
}
pageFaults++;
}
printFrames(frames, capacity);
}
printf("Optimal Page Faults: %d\n", pageFaults);
}
int main()
{
int pages[] = {1, 2, 3, 4, 1, 5, 6, 7, 8, 7, 8, 9, 7, 8, 9, 5, 4, 5, 4, 2};
int capacity = 3;
int n = sizeof(pages) / sizeof(pages[0]);
printf("Page reference sequence:\n");
for (int i = 0; i < n; i++)
{
printf("%d ", pages[i]);
}
printf("\n\n");

printf("\n");
printf("Optimal Algorithm:\n");
optimal(pages, n, capacity);
printf("\n");

return 0;
}
```

Output:-



```
(base) sahyadri@sahyadri:~/preksha$ gedit optimal.c
(base) sahyadri@sahyadri:~/preksha$ gcc optimal.c
(base) sahyadri@sahyadri:~/preksha$ ./a.out
Page reference sequence:
12341567878978954542


Optimal Algorithm
1--
12-
123
124
124
524
564
574
578
578
578
978
978
978
978
578
548
548
548
248
Optimal Page Faults:12
```

Experiment 8:- Analyze the seek time for the following Disk scheduling algorithms –

1. FCFS 2. SCAN 3. LOOK

   Program:-

```
#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>



// Function to calculate absolute difference between two numbers

int absDiff(int a, int b)

{

return abs(a - b);

}
```

```
// FCFS Disk Scheduling Algorithm
int FCFS(int *requests, int numRequests)
{
    int totalSeekTime = 0;

    for (int i = 1; i < numRequests; i++)
{
    totalSeekTime += absDiff(requests[i], requests[i - 1]);
    }

    return totalSeekTime;
}

// SCAN Disk Scheduling Algorithm
 int SCAN(int *requests, int numRequests, int start, int end)
 {
 int totalSeekTime = 0;
    int currentTrack = start;
    bool movingUp = true;
 // Moving towards the end of the disk

    while (numRequests > 0)
```

```
{

for (int i = 0; i < numRequests; i++)

 {

            if (requests[i] == currentTrack)

{

      totalSeekTime += absDiff(currentTrack, start);

      start = currentTrack;

      requests[i] = -1;

// Mark this request as processed

            }

        }


        if (movingUp)

 {

           currentTrack++;

        if (currentTrack > end)

 {                 movingUp = false;

currentTrack = end;

            }

        }

 else

{
 currentTrack--;
          if (currentTrack < 0)
 {
   movingUp = true;
           currentTrack = 0;
          }
       }

       // Remove processed requests
       int newNumRequests = 0;
   for (int i = 0; i < numRequests; i++)
{             if (requests[i] != -1)
{
              requests[newNumRequests++] = requests[i];
          }
     }
       numRequests = newNumRequests;
   }

   return totalSeekTime;
}

// LOOK Disk Scheduling Algorithm
```

```
int LOOK(int *requests, int numRequests, int start, int end)
{
    int totalSeekTime = 0;
 int currentTrack = start;
   bool movingUp = true;
// Moving towards the end of the disk


   while (numRequests > 0)
{
  for (int i = 0; i < numRequests; i++)
{
if (requests[i] == currentTrack)
{
     totalSeekTime += absDiff(currentTrack, start);
              start = currentTrack;
           requests[i] = -1;
// Mark this request as processed
           }
       }


       if (movingUp)
{
currentTrack++;
```

```
        if (currentTrack > end)
 {
 movingUp = false;
               currentTrack = end;
          }
     }
     else
     {
          currentTrack--;
          if (currentTrack < 0)
 {
   movingUp = true;
              currentTrack = 0;
          }
       }

       // Remove processed requests
        int newNumRequests = 0;
       for (int i = 0; i < numRequests; i++)
 {
          if (requests[i] != -1)
{
              requests[newNumRequests++] = requests[i];
```

```
            }
        }
        numRequests = newNumRequests;
    }

    return totalSeekTime;
}

int main()
 {
    int numRequests, start, end;

    printf("Enter the number of requests: ");
    scanf("%d", &numRequests);

    int *requests = (int *)malloc(numRequests * sizeof(int));
    printf("Enter the requests: ");
    for (int i = 0; i < numRequests; i++)
 {
         scanf("%d", &requests[i]);
    }

    printf("Enter the start and end of the disk: ");
    scanf("%d %d", &start, &end);

    int fcfsSeekTime = FCFS(requests, numRequests);
    int scanSeekTime = SCAN(requests, numRequests, start, end);
    int lookSeekTime = LOOK(requests, numRequests, start, end);

    printf("FCFS Seek Time: %d\n", fcfsSeekTime);
    printf("SCAN Seek Time: %d\n", scanSeekTime);
    printf("LOOK Seek Time: %d\n", lookSeekTime);

    free(requests);

    return 0;
}
```

Output:-

```
Enter the number of requests: 5
Enter the requests:
11
22
33
44
55
Enter the start and end of the disk: 10 100
```

```
FCFS Seek Time: 44

SCAN Seek Time: 45

LOOK Seek Time: 45
```