OPERATING SYSTEMS

TUTORIAL FILE

SUBMITTED BY-

AMAN JAIN

2017UCO1583

COE-2

INDEX

|  |  |  |
| --- | --- | --- |
| S.NO. | AIM | TEACHER’S SIGNATURE |
| 1. | Program for Process Creation and Termination using Linux operating System. |  |
| 2. | Implementing Producer-Consumer Problem using Bounded and unbounded buffer. |  |
| 3. | Program for illustrating inter-process communication |  |
| 4. | Implementing the following CPU Scheduling Algorithms:-   * FCFS * SJF * Priority Scheduling * Round Robin |  |
| 5. | Implementing following algorithms for critical section problems:-   * Peterson’s Algorithm * Baker’s Algorithm |  |
| 6. | Implementing the following Semaphore Problems:-   * Reader-Writer Problem * Dining Philosopher’s Problem |  |
| 7. | Implementing Banker’s Algorithm. |  |
| 8. | Programs to illustrate the Page Replacement Algorithms. |  |
| 9. | A Program to illustrate File Operations. |  |
| 10. | Implementing the Disk Scheduling Algorithms. |  |

**AIM-** Write a program for process creation and termination for the Linux operating system (e.g. fork (), exit(), wait() etc. )

**CODE-**

#include<stdio.h>

#include<stdlib.h>

#include<sys/wait.h>

#include<unistd.h>

void forkexample()

{

// child process because return value zero

if (fork() == 0)

printf("Hello from Child!\n");

else

printf("Hello from Parent!\n");

}

int main()

{

fork();

forkexample();

pid\_t cpid;

if (fork()== 0)

exit(0); /\* terminates child \*/

else

cpid = wait(NULL);

printf("Parent pid = %d\n", getpid());

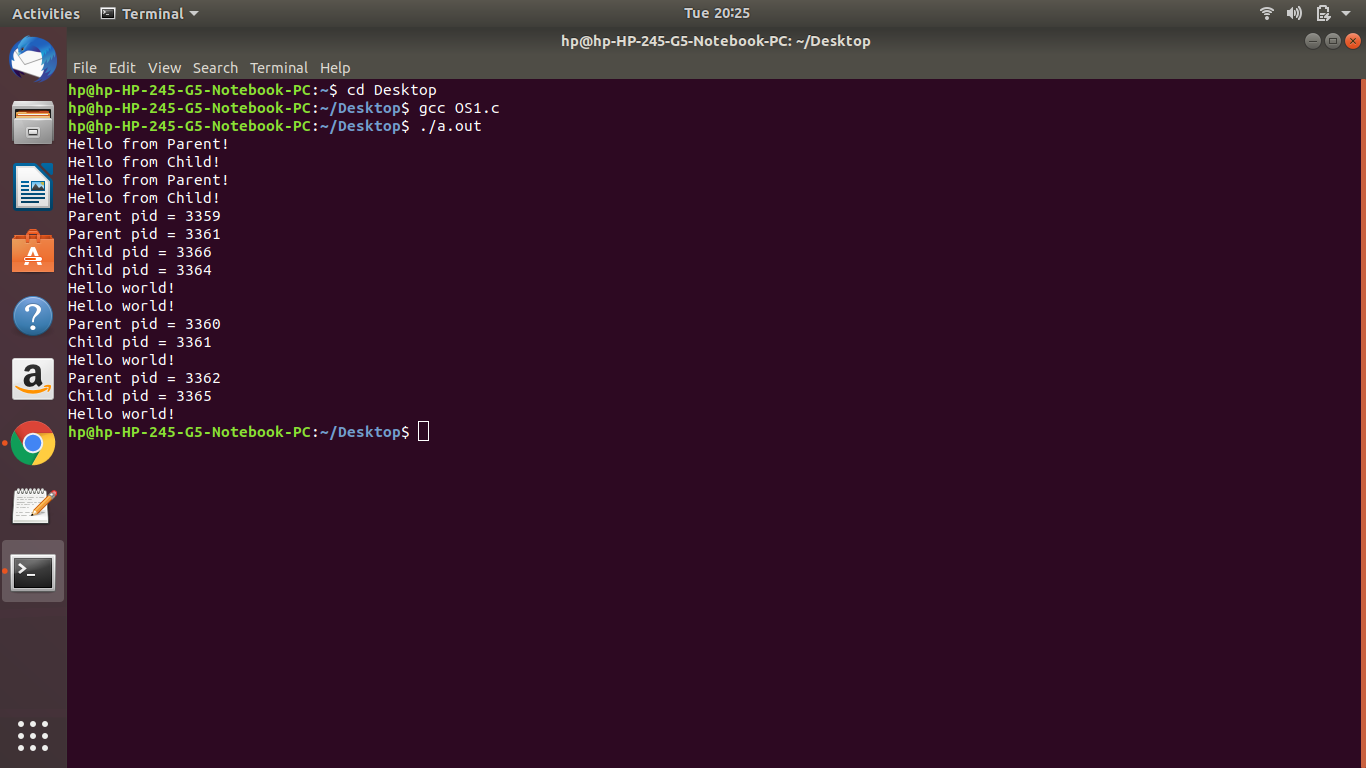
printf("Child pid = %d\n", cpid);

printf("Hello world!\n");

return 0;

}

**OUTPUT-**



**AIM**- Implement Producer-Consumer Problem using Bounded and Unbounded buffer.

**CODE-**

#include <iostream>

#include <pthread.h>

#include <semaphore.h>

#include <random>

#include <unistd.h>

using namespace std;

#define BUFFER\_SIZE 10

int buffer[BUFFER\_SIZE];

int ind=0;

sem\_t full,empty;

pthread\_mutex\_t mut;

void\* produce(void\* arg){

while(1){

sleep(1);

sem\_wait(&empty);

pthread\_mutex\_lock(&mut);

int item = rand()%100;

buffer[ind++] = item;

cout<<"Produced "<<item<<endl;

pthread\_mutex\_unlock(&mut);

sem\_post(&full);

}

}

void\* consume(void\* arg){

while(1){

sleep(1);

sem\_wait(&full);

pthread\_mutex\_lock(&mut);

int item = buffer[--ind];

cout<<"Consumed "<<item<<endl;

pthread\_mutex\_unlock(&mut);

sem\_post(&empty);

}

}

int main()

{

pthread\_t producer,consumer;

sem\_init(&empty,0,BUFFER\_SIZE);

sem\_init(&full,0,0);

pthread\_mutex\_init(&mut,NULL);

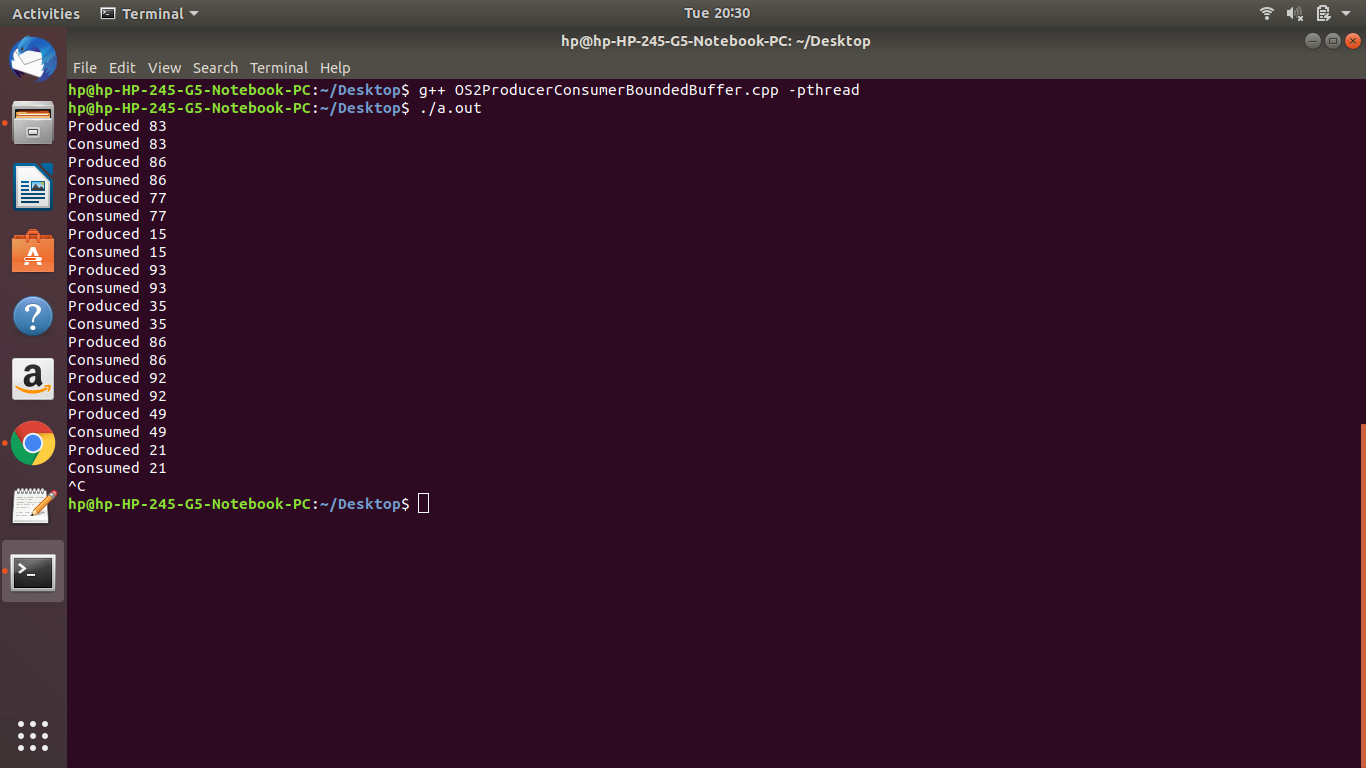
pthread\_create(&producer,NULL,produce,NULL);

pthread\_create(&consumer,NULL,consume,NULL);

pthread\_exit(NULL);

}

**OUTPUT-**



**CODE-**

#include<stdio.h>

#include<stdlib.h>

int mutex=1,full=0,empty=10,x=0;

void producer();

void consumer();

int wait(int);

int signal(int);

int main()

{

int n;

printf("\n1.Producer\n2.Consumer\n3.Exit");

while(1)

{

printf("\nEnter 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("\nProducer produces the item %d",x);

mutex=signal(mutex);

}

void consumer()

{

mutex=wait(mutex);

full=wait(full);

empty=signal(empty);

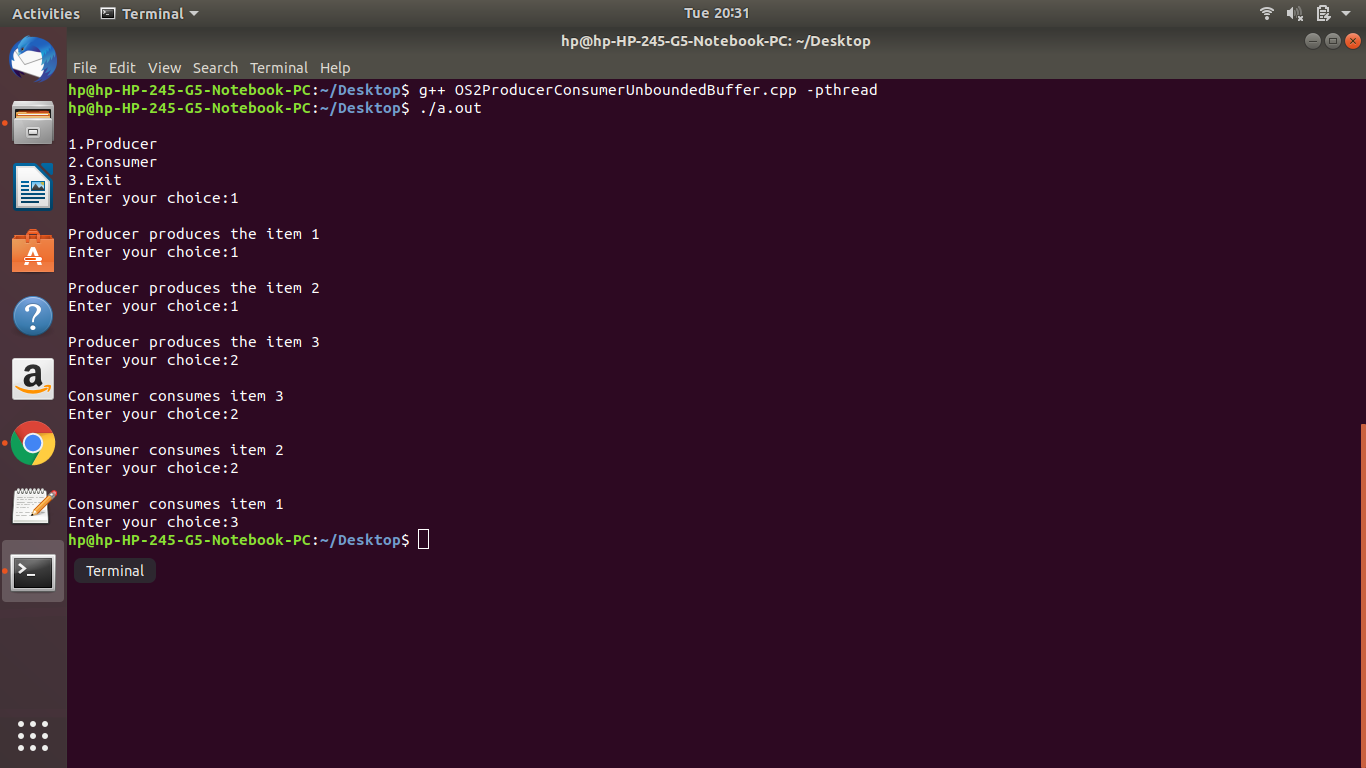
printf("\nConsumer consumes item %d",x);

x--;

mutex=signal(mutex);

}

**OUTPUT-**



**AIM-** Write a program for illustrating Inter-Process Communication.

**CODE-**

// C Program for Message Queue (Reader Process)

#include <stdio.h>

#include <sys/ipc.h>

#include <sys/msg.h>

// structure for message queue

struct mesg\_buffer

{

long mesg\_type;

char mesg\_text[100];

}message;

int main()

{

key\_t key;

int msgid;

// ftok to generate unique key

key = ftok("progfile", 65);

// msgget creates a message queue and returns identifier

msgid = msgget(key, 0666 | IPC\_CREAT);

// msgrcv to receive message

msgrcv(msgid, &message, sizeof(message), 1, 0);

// display the message

printf("Data Received is : %s \n", message.mesg\_text);

// to destroy the message queue

msgctl(msgid, IPC\_RMID, NULL);

return 0;

}

// C Program for Message Queue (Writer Process)

#include <stdio.h>

#include <sys/ipc.h>

#include <sys/msg.h>

// structure for message queue

struct mesg\_buffer

{

long mesg\_type;

char mesg\_text[100];

}message;

int main()

{

key\_t key;

int msgid;

// ftok to generate unique key

key = ftok("progfile", 65);

// msgget creates a message queue and returns identifier

msgid = msgget(key, 0666 | IPC\_CREAT);

message.mesg\_type = 1;

printf("Write Data : ");

fgets(message.mesg\_text,100,stdin);

// msgsnd to send message

msgsnd(msgid, &message, sizeof(message), 0);

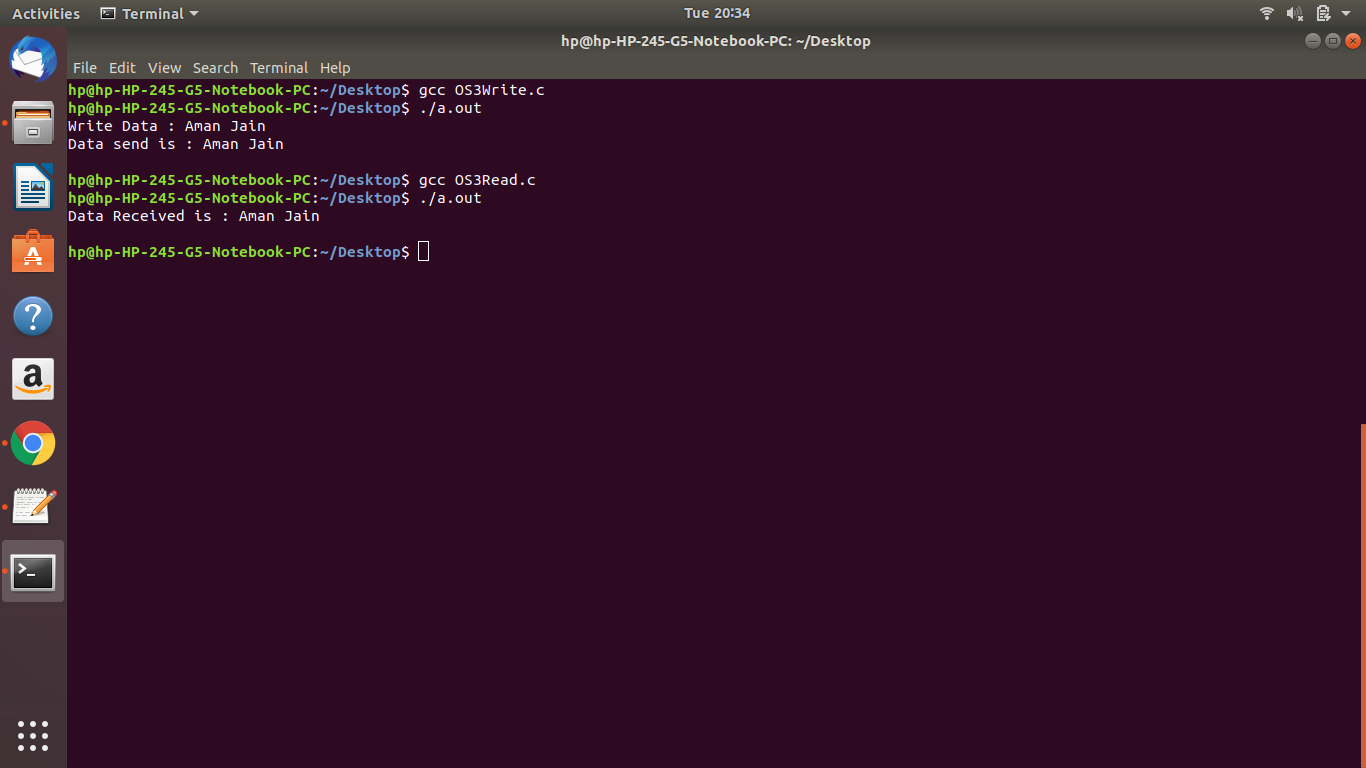
// display the message

printf("Data send is : %s \n", message.mesg\_text);

return 0;

}

**OUTPUT-**



**AIM-** Implementing all the CPU Scheduling Algorithms like:-

* First-cum-First-Serve
* Shortest-Job-First
* Priority Scheduling
* Round Robin

**CODE-**

#include<iostream>

using namespace std;

void findWaitingTime(int processes[], int n, int bt[],

int wt[], int at[])

{

int service\_time[n];

service\_time[0] = 0;

wt[0] = 0;

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

{

service\_time[i] = service\_time[i-1] + bt[i-1];

wt[i] = service\_time[i] - at[i];

if (wt[i] < 0)

wt[i] = 0;

}

}

void findTurnAroundTime(int processes[], int n, int bt[],

int wt[], int tat[])

{

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

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

}

void findavgTime(int processes[], int n, int bt[], int at[])

{

int wt[n], tat[n];

findWaitingTime(processes, n, bt, wt, at);

findTurnAroundTime(processes, n, bt, wt, tat);

cout << "Processes " << " Burst Time " << " Arrival Time "

<< " Waiting Time " << " Turn-Around Time "

<< " Completion Time \n";

int total\_wt = 0, total\_tat = 0;

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

{

total\_wt = total\_wt + wt[i];

total\_tat = total\_tat + tat[i];

int compl\_time = tat[i] + at[i];

cout << " " << i+1 << "\t\t" << bt[i] << "\t\t"

<< at[i] << "\t\t" << wt[i] << "\t\t "

<< tat[i] << "\t\t " << compl\_time << endl;

}

cout << "Average waiting time = "

<< (float)total\_wt / (float)n;

cout << "\nAverage turn around time = "

<< (float)total\_tat / (float)n;

}

int main()

{

int n;

//int burst\_time[] = {5, 9, 6};

//int arrival\_time[] = {0, 3, 6};

int processes[10] , burst\_time[10],arrival\_time[10];

cout<<"Enter no of processes "<<endl;

cin>>n;

cout<<"Enter Burst Time of all the processes \n";

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

processes[i] = i+1;

cin>>burst\_time[i];

}

cout<<"Enter Arrival Time of all the processes \n";

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

cin>>arrival\_time[i];

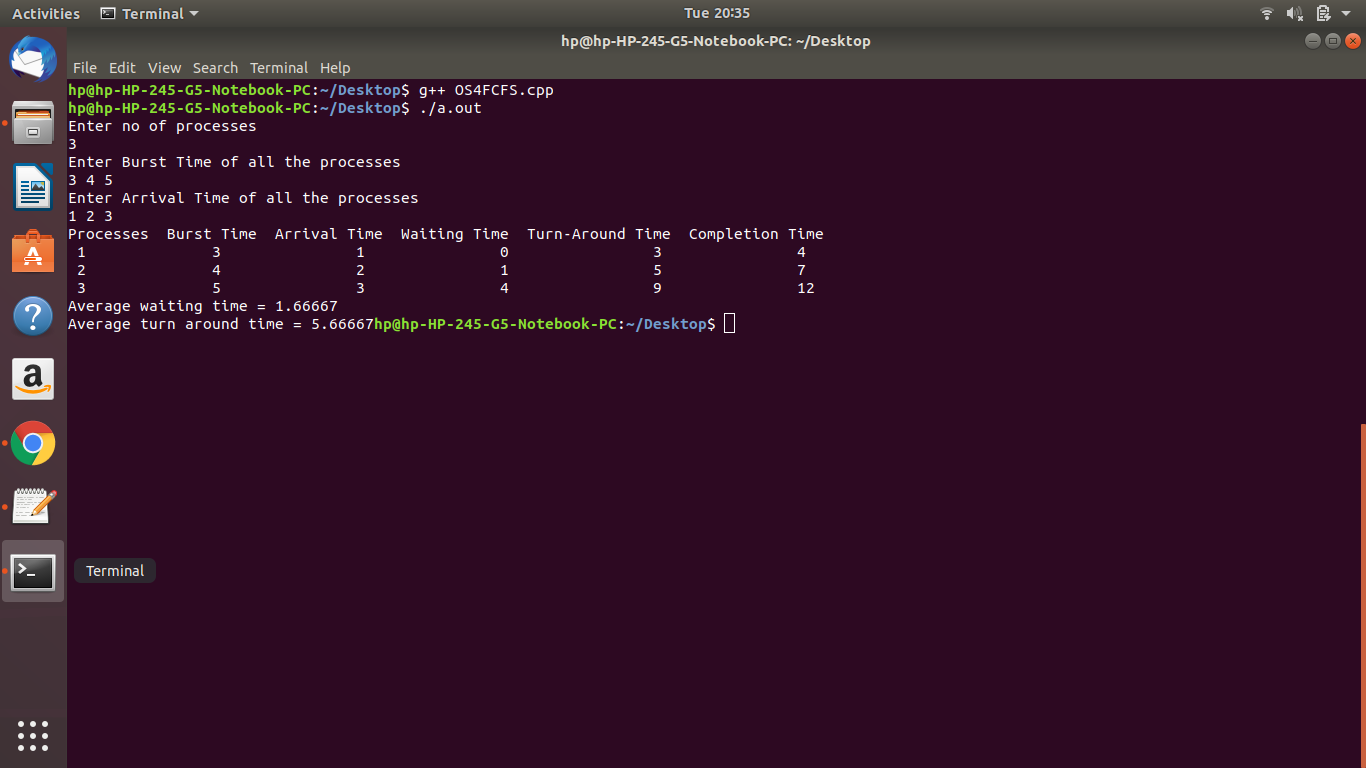
}

findavgTime(processes, n, burst\_time, arrival\_time);

return 0;

}

**OUTPUT-**



**CODE-**

#include <iostream>

#include<bits/stdc++.h>

using namespace std;

struct Process {

int pid; // Process ID

int bt; // Burst Time

int art; // Arrival Time

};

void findWaitingTime(Process proc[], int n,int wt[])

{

int rt[n];

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

rt[i] = proc[i].bt;

int complete = 0, t = 0, minm = INT\_MAX;

int shortest = 0, finish\_time;

bool check = false;

while (complete != n)

{

for (int j = 0; j < n; j++) {

if ((proc[j].art <= t) &&

(rt[j] < minm) && rt[j] > 0) {

minm = rt[j];

shortest = j;

check = true;

}

}

if (check == false) {

t++;

continue;

}

rt[shortest]--;

minm = rt[shortest];

if (minm == 0)

minm = INT\_MAX;

if (rt[shortest] == 0) {

complete++;

check = false;

finish\_time = t + 1;

wt[shortest] = finish\_time -

proc[shortest].bt -

proc[shortest].art;

if (wt[shortest] < 0)

wt[shortest] = 0;

}

t++;

}

}

void findTurnAroundTime(Process proc[], int n,

int wt[], int tat[])

{

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

tat[i] = proc[i].bt + wt[i];

}

void findavgTime(Process proc[], int n)

{

int wt[n], tat[n], total\_wt = 0,

total\_tat = 0;

findWaitingTime(proc, n, wt);

findTurnAroundTime(proc, n, wt, tat);

cout << "Processes "

<< " Burst time "

<< " Arrival time "

<< " Waiting time "

<< " Turn around time\n";

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

total\_wt = total\_wt + wt[i];

total\_tat = total\_tat + tat[i];

cout << " " << proc[i].pid << "\t\t"

<< proc[i].bt << "\t\t "

<< proc[i].art << "\t\t "<< wt[i]

<< "\t\t " << tat[i] << endl;

}

cout << "\nAverage waiting time = "

<< (float)total\_wt / (float)n;

cout << "\nAverage turn around time = "

<< (float)total\_tat / (float)n;

}

int main()

{

int n;

Process proc[10];

cout<<"Enter no of processes "<<endl;

cin>>n;

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

{

proc[i].pid = i+1;

cout<<"Process "<<i+1<<endl;

cout<<"Enter Burst Time\n";

cin>>proc[i].bt;

cout<<"Enter Arrival Time\n";

cin>>proc[i].art;

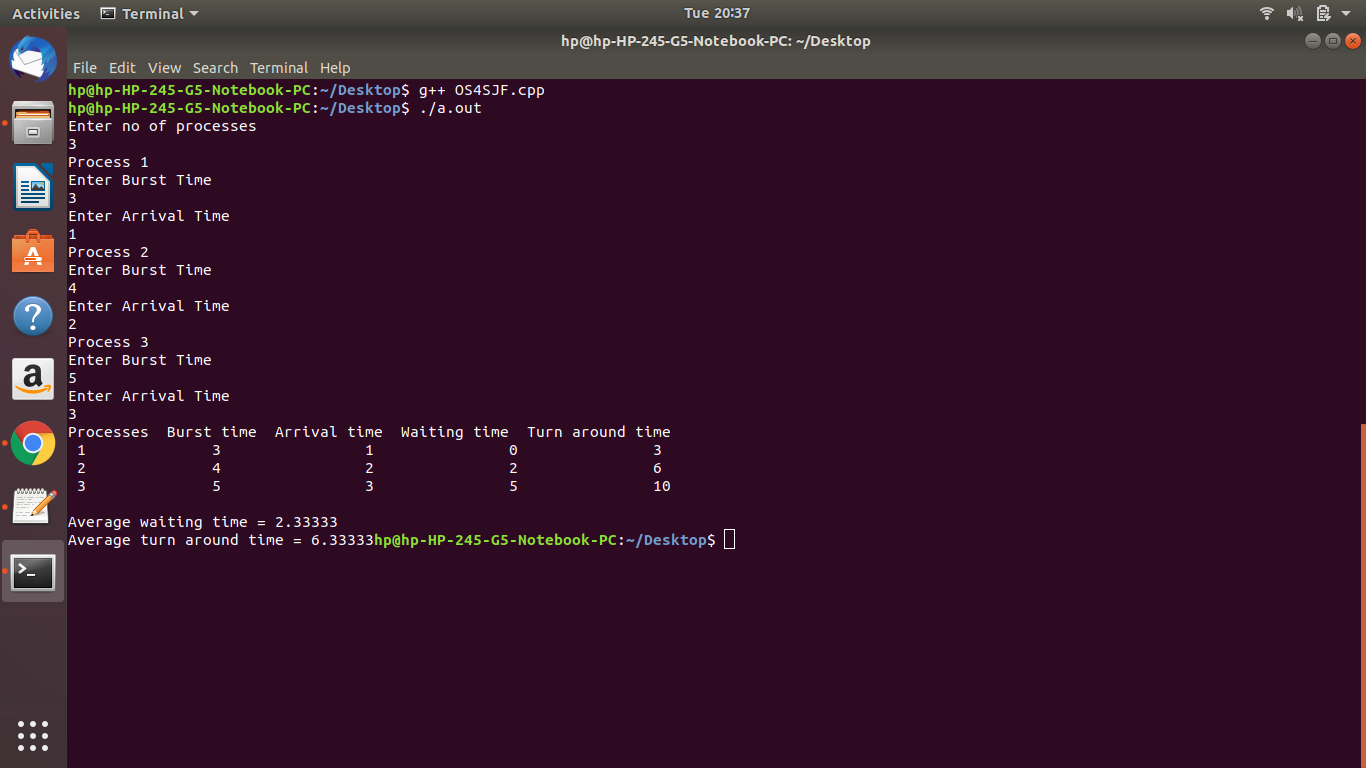
}

findavgTime(proc, n);

return 0;

}

**OUTPUT-**



**CODE-**

#include<iostream>

#include<algorithm>

#include<string.h>

using namespace std;

typedef struct proccess

{

int at,bt,ct,ta,wt,btt,pr;

string pro\_id;

}schedule;

bool compare(schedule a,schedule b)

{

return a.at<b.at;

}

bool compare2(schedule a,schedule b)

{

return a.pr>b.pr;

}

int main()

{

schedule pro[10];

int n,i,j,pcom;

cout<<"Enter the number of process::";

cin>>n;

cout<<"Enter the Process id arrival time burst time and priority :::";

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

{

cin>>pro[i].pro\_id;

cin>>pro[i].at;

cin>>pro[i].bt;

pro[i].btt=pro[i].bt;

cin>>pro[i].pr;

}

sort(pro,pro+n,compare);

i=0;

pcom=0;

while(pcom<n)

{

for(j=0;j<n;j++)

{

if(pro[j].at>i)

break;

}

sort(pro,pro+j,compare2);

if(j>0)

{

for(j=0;j<n;j++)

{

if(pro[j].bt!=0)

break;

}

if(pro[j].at>i)

i=pro[j].at;

pro[j].ct=i+1;

pro[j].bt--;

}

i++;

pcom=0;

for(j=0;j<n;j++)

{

if(pro[j].bt==0)

pcom++;

}

}

cout<<"Process Id "<<" Arrival Time "<<" Burst Time "<<" Completion Time "<<" Turn Around Time "<<" Waiting Time "<<"Priority "<<endl;

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

{

pro[i].ta=pro[i].ct-pro[i].at;

pro[i].wt=pro[i].ta-pro[i].btt;

cout<<" "<<pro[i].pro\_id<<"\t\t"

<<pro[i].at<<"\t\t"

<<pro[i].btt<<"\t\t"

<<pro[i].ct<<"\t\t"

<<pro[i].ta<<"\t\t"

<<pro[i].wt<<"\t\t"

<<pro[i].pr;

cout<<endl;

}

int total\_wt=0,total\_tat=0;

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

total\_wt = total\_wt + pro[i].wt;

total\_tat = total\_tat + pro[i].ta;

}

cout << "\nAverage waiting time = "

<< (float)total\_wt / (float)n;

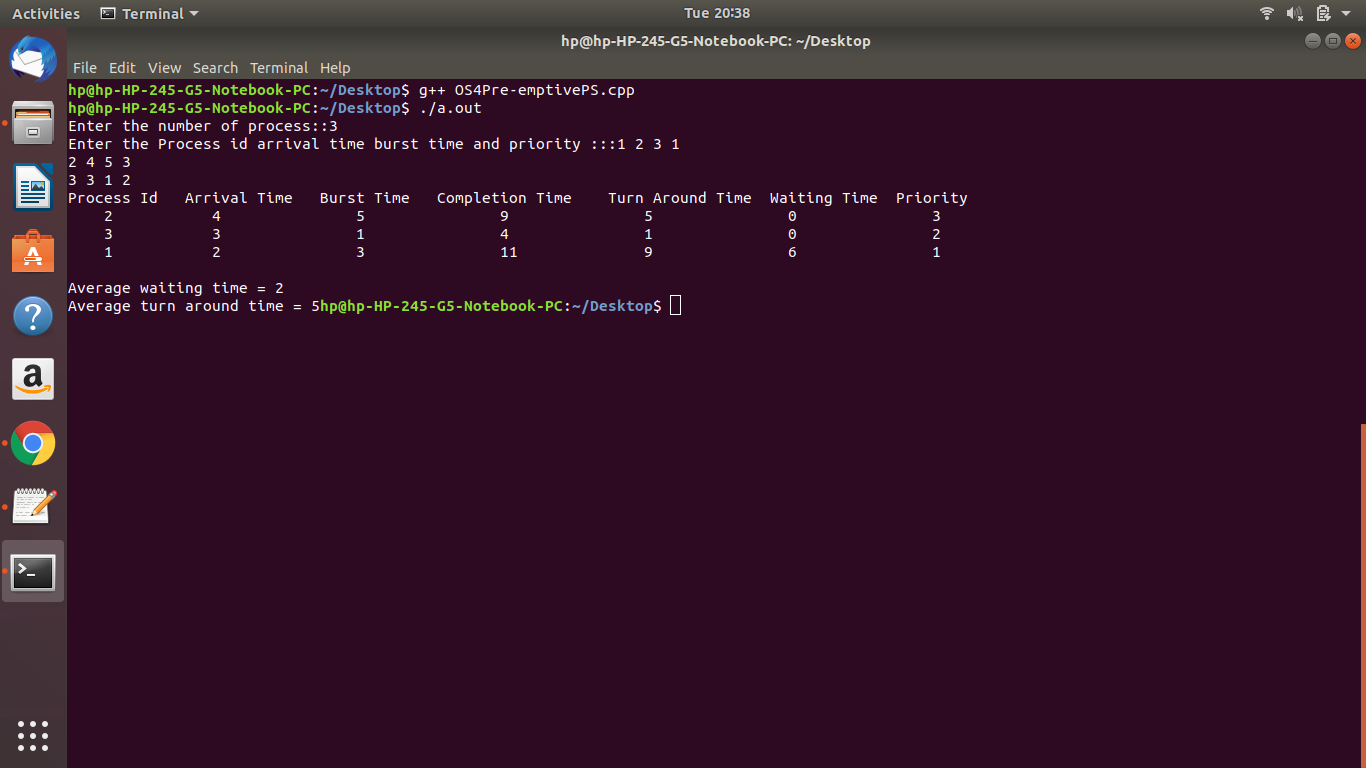
cout << "\nAverage turn around time = "

<< (float)total\_tat / (float)n;

return 0;

}

**OUTPUT-**



**CODE-**

#include<stdio.h>

int main()

{

int count,j,n,time,remain,flag=0,time\_quantum;

int wait\_time=0,turnaround\_time=0,at[10],bt[10],rt[10];

printf("Enter Total Process:\t ");

scanf("%d",&n);

remain=n;

for(count=0;count<n;count++)

{

printf("Enter Arrival Time and Burst Time for Process Process Number %d :",count+1);

scanf("%d",&at[count]);

scanf("%d",&bt[count]);

rt[count]=bt[count];

}

printf("Enter Time Quantum:\t");

scanf("%d",&time\_quantum);

printf("\n\nProcess\t|Turnaround Time|Waiting Time\n\n");

for(time=0,count=0;remain!=0;)

{

if(rt[count]<=time\_quantum && rt[count]>0)

{

time+=rt[count];

rt[count]=0;

flag=1;

}

else if(rt[count]>0)

{

rt[count]-=time\_quantum;

time+=time\_quantum;

}

if(rt[count]==0 && flag==1)

{

remain--;

printf("P[%d]\t|\t%d\t|\t%d\n",count+1,time-at[count],time-at[count]-bt[count]);

wait\_time+=time-at[count]-bt[count];

turnaround\_time+=time-at[count];

flag=0;

}

if(count==n-1)

count=0;

else if(at[count+1]<=time)

count++;

else

count=0;

}

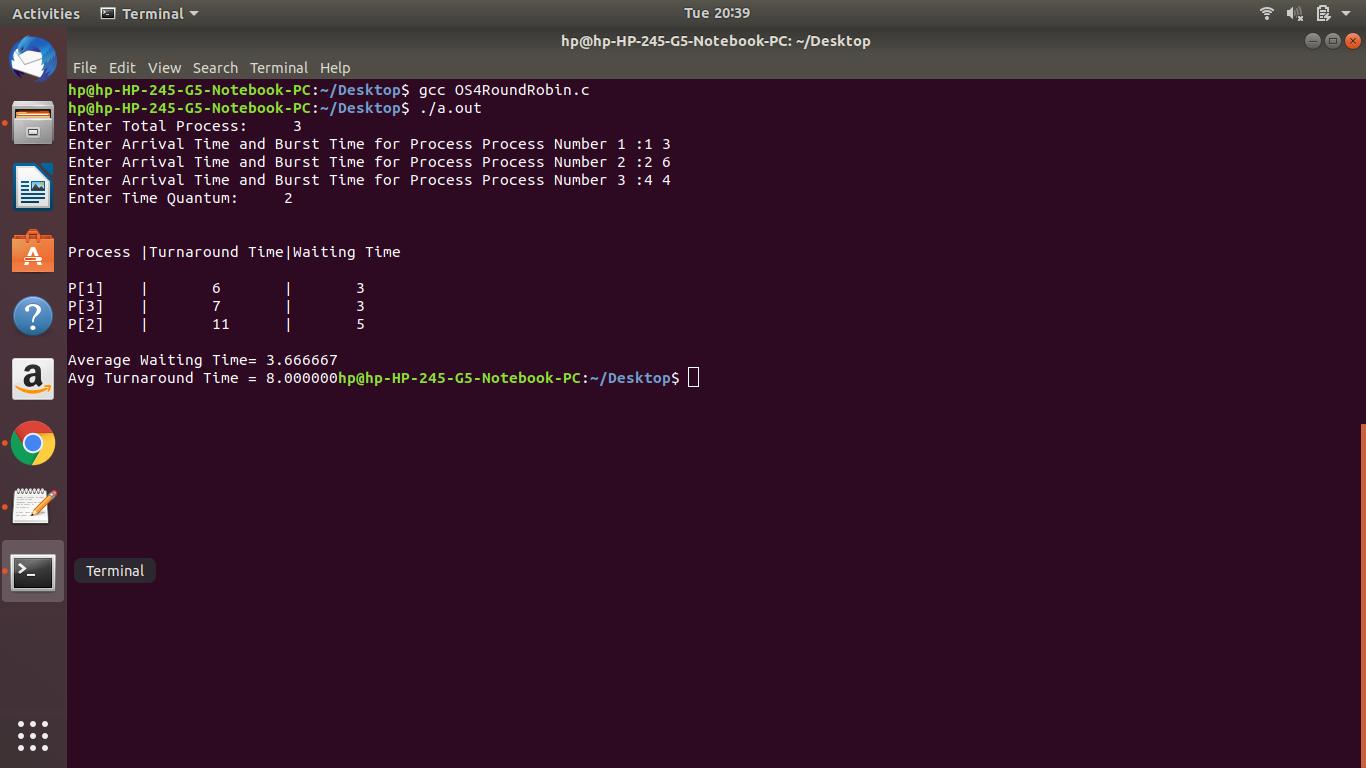
printf("\nAverage Waiting Time= %f\n",wait\_time\*1.0/n);

printf("Avg Turnaround Time = %f",turnaround\_time\*1.0/n);

return 0;

}

**OUTPUT-**



**AIM**- Write a program to illustrate following algorithms of the Critical Section Problem.

**CODE-**

#include<pthread.h>

#include<stdio.h>

void \*func1(void \*);

void \*func2(void \*);

int flag[2];

int turn=0;

int global=100;

int main()

{

pthread\_t tid1,tid2;

pthread\_create(&tid1,NULL,func1,NULL);

pthread\_create(&tid2,NULL,func2,NULL);

pthread\_join(tid1,NULL);

pthread\_join(tid2,NULL);

}

void \*func1(void \*param)

{

int i=0;

while(i<2)

{

flag[0]=1;

turn=1;

while(flag[1]==1 && turn==1);

global+=100;

printf("FT: g: %d\t",global);

flag[0]=0;

i++;

}

}

void \*func2(void \*param)

{

int i=0;

while(i<2)

{

flag[1]=1;

turn=0;

while(flag[0]==1 && turn==0);

global-=75;

printf("SP: g: %d\n",global);

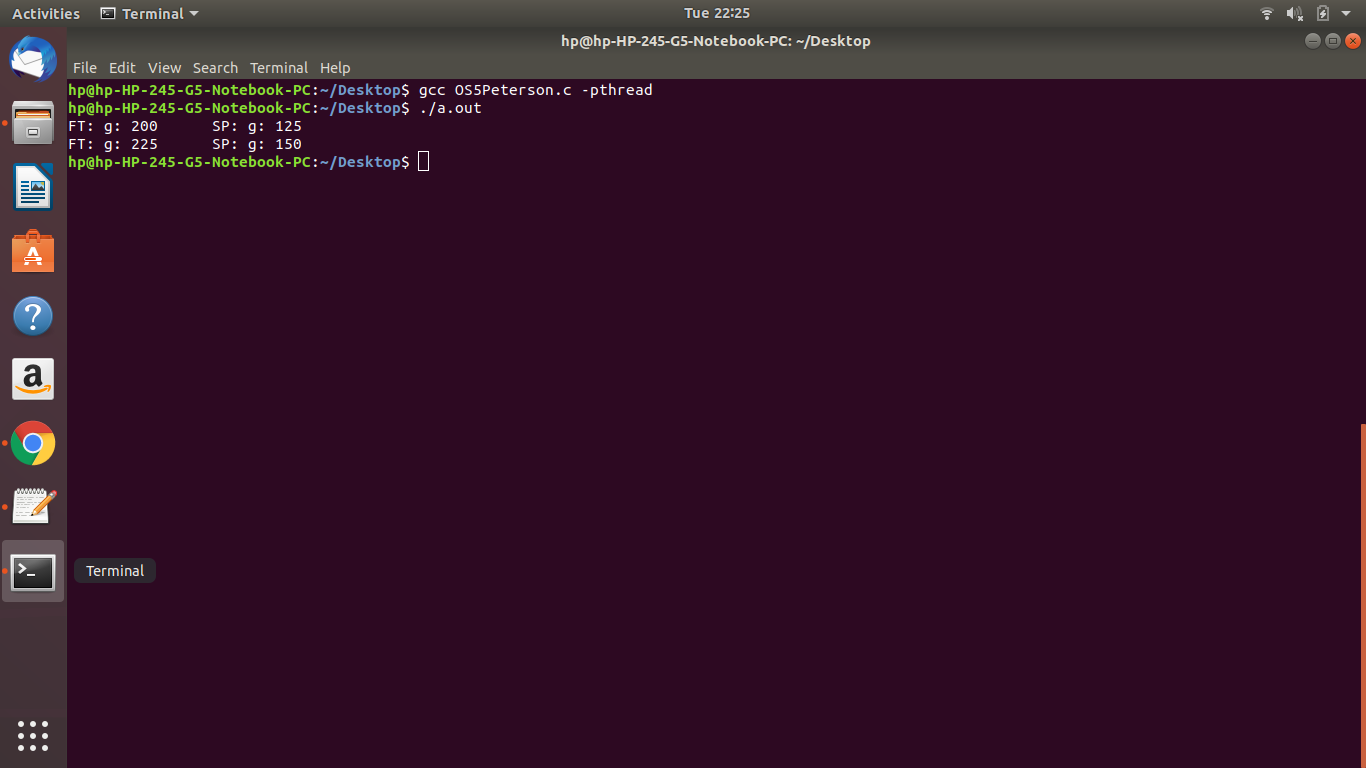
flag[1]=0;

i++;

}

}

**OUTPUT-**



**CODE-**

#include "pthread.h"

#include "stdio.h"

#include "unistd.h"

#include "string.h"

#define MEMBAR \_\_sync\_synchronize()

#define THREAD\_COUNT 8

volatile int tickets[THREAD\_COUNT];

volatile int choosing[THREAD\_COUNT];

volatile int resource;

void lock(int thread) {

choosing[thread] = 1;

MEMBAR;

int max\_ticket = 0;

for (int i = 0; i < THREAD\_COUNT; ++i) {

int ticket = tickets[i];

max\_ticket = ticket > max\_ticket ? ticket : max\_ticket;

}

tickets[thread] = max\_ticket + 1;

MEMBAR;

choosing[thread] = 0;

MEMBAR;

for (int other = 0; other < THREAD\_COUNT; ++other) {

while (choosing[other]) { }

MEMBAR;

while (tickets[other] != 0 &&

(tickets[other] < tickets[thread] ||

(tickets[other] == tickets[thread] && other < thread))) { }

}

}

void unlock(int thread) {

MEMBAR;

tickets[thread] = 0;

}

void use\_resource(int thread) {

if (resource != 0) {

printf("Resource was acquired by %d, but is still in-use by %d!\n",

thread, resource);

}

resource = thread;

printf("%d using resource...\n", thread);

MEMBAR;

sleep(2);

resource = 0;

}

void \*thread\_body(void \*arg) {

long thread = (long)arg;

lock(thread);

use\_resource(thread);

unlock(thread);

return NULL;

}

int main(int argc, char \*\*argv) {

memset((void\*)tickets, 0, sizeof(tickets));

memset((void\*)choosing, 0, sizeof(choosing));

resource = 0;

pthread\_t threads[THREAD\_COUNT];

for (int i = 0; i < THREAD\_COUNT; ++i) {

pthread\_create(&threads[i], NULL, &thread\_body, (void\*)((long)i));

}

for (int i = 0; i < THREAD\_COUNT; ++i) {

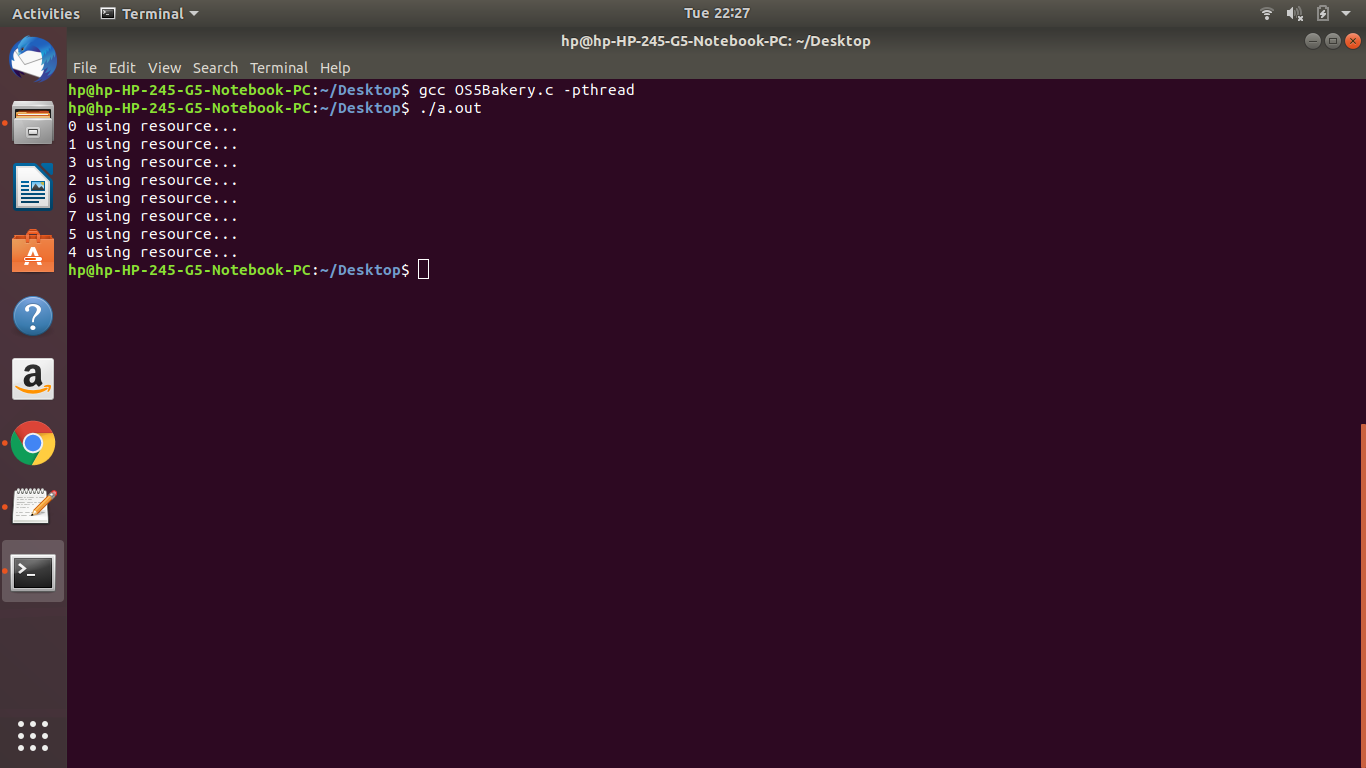
pthread\_join(threads[i], NULL);

}

return 0;

}

**OUTPUT-**



**AIM-** Write a program to illustrate the use of Semaphores with the following problems:-

* Reader-Writer Problem
* Dining Philosopher’s Problem

**CODE-**

#include<iostream>

#include<bits/stdc++.h>

#define n 4

using namespace std;

int compltedPhilo = 0,i;

struct fork

{

int taken;

}ForkAvil[n];

struct philosp

{

int left;

int right;

}Philostatus[n];

void goForDinner(int philID)

{ //same like threads concept here cases implemented

if(Philostatus[philID].left==10 && Philostatus[philID].right==10)

cout<<"Philosopher "<<philID+1<<" completed his dinner\n";

//if already completed dinner

else if(Philostatus[philID].left==1 && Philostatus[philID].right==1){

//if just taken two forks

cout<<"Philosopher "<<philID+1<<" completed his dinner\n";

Philostatus[philID].left = Philostatus[philID].right = 10; //remembering that he completed dinner by assigning value 10

int otherFork = philID-1;

if(otherFork== -1)

otherFork=(n-1);

ForkAvil[philID].taken = ForkAvil[otherFork].taken = 0; //releasing forks

cout<<"Philosopher "<<philID+1<<" released fork "<<philID+1<<" and fork "<<otherFork+1<<"\n";

compltedPhilo++;

}

else if(Philostatus[philID].left==1 && Philostatus[philID].right==0){ //left already taken, trying for right fork

if(philID==(n-1)){

if(ForkAvil[philID].taken==0){ //KEY POINT OF THIS PROBLEM, THAT LAST PHILOSOPHER TRYING IN reverse DIRECTION

ForkAvil[philID].taken = Philostatus[philID].right = 1;

cout<<"Fork "<<philID+1<<" taken by philosopher "<<philID+1<<"\n";

}else{

cout<<"Philosopher "<<philID+1<<" is waiting for fork "<<philID+1<<"\n";

}

}else{ //except last philosopher case

int dupphilID = philID;

philID-=1;

if(philID== -1)

philID=(n-1);

if(ForkAvil[philID].taken == 0){

ForkAvil[philID].taken = Philostatus[dupphilID].right = 1;

cout<<"Fork "<<philID+1<<" taken by Philosopher "<<dupphilID+1<<"\n";

}else{

cout<<"Philosopher "<<dupphilID+1<<" is waiting for Fork "<<philID+1<<"\n";

}

}

}

else if(Philostatus[philID].left==0){ //nothing taken yet

if(philID==(n-1)){

if(ForkAvil[philID-1].taken==0){ //KEY POINT OF THIS PROBLEM, THAT LAST PHILOSOPHER TRYING IN reverse DIRECTION

ForkAvil[philID-1].taken = Philostatus[philID].left = 1;

cout<<"Fork "<<philID<<" taken by philosopher "<<philID+1<<"\n";

}else{

cout<<"Philosopher "<<philID+1<<" is waiting for fork "<<philID<<"\n";

}

}else{ //except last philosopher case

if(ForkAvil[philID].taken == 0){

ForkAvil[philID].taken = Philostatus[philID].left = 1;

cout<<"Fork "<<philID+1<<" taken by Philosopher "<<philID+1<<"\n";

}else{

cout<<"Philosopher "<<philID+1<<" is waiting for Fork "<<philID+1<<"\n";

}

}

}else{}

}

int main(){

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

ForkAvil[i].taken=Philostatus[i].left=Philostatus[i].right=0;

while(compltedPhilo<n){

/\* Observe here carefully, while loop will run until all philosophers complete dinner

Actually problem of deadlock occur only thy try to take at same time

This for loop will say that they are trying at same time. And remaining status will print by go for dinner function

\*/

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

goForDinner(i);

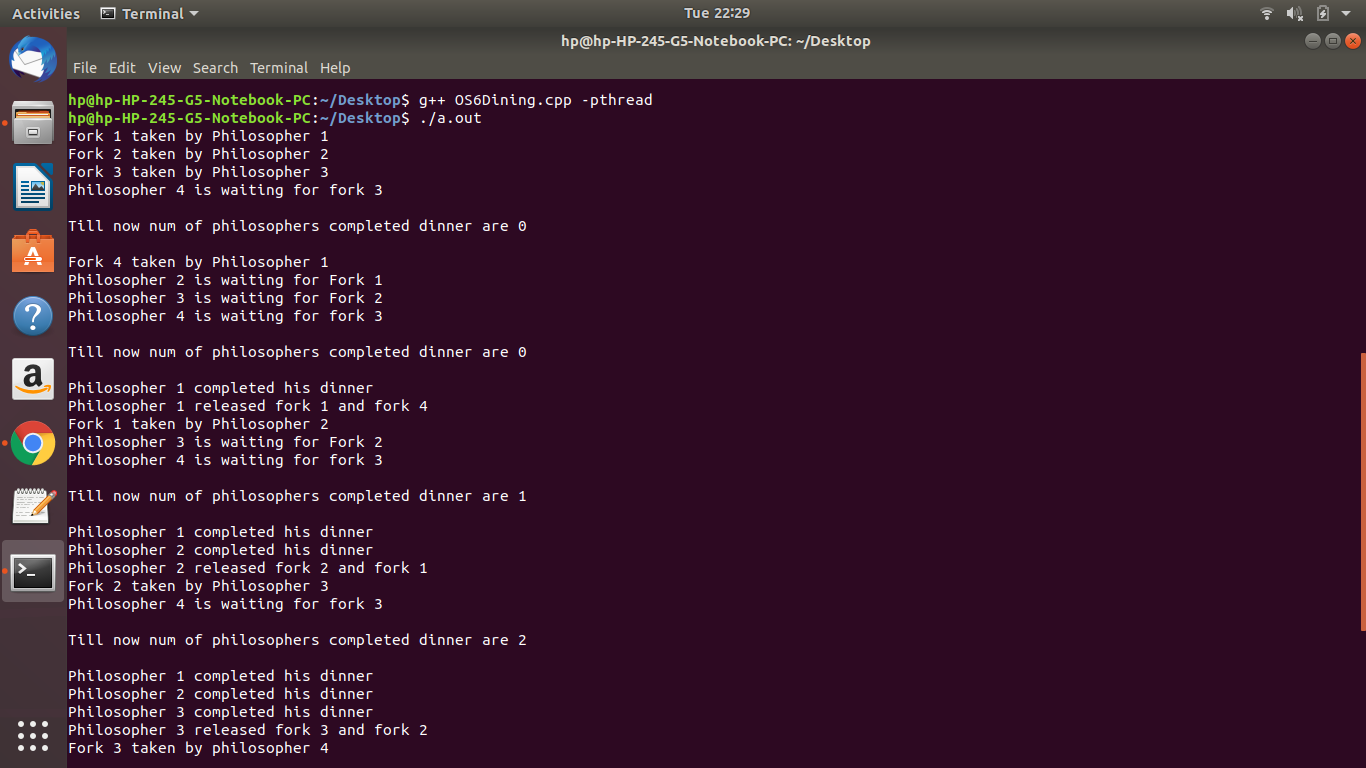
cout<<"\nTill now num of philosophers completed dinner are "<<compltedPhilo<<"\n\n";

}

return 0;

}

**OUTPUT-**





**CODE-**

#include<stdio.h>

#include<pthread.h>

#include<semaphore.h>

#include<stdint.h>

#include<unistd.h>

sem\_t mutex,writeblock;

int data = 0,rcount = 0;

void \*reader(void \*arg)

{

int f;

f = ((intptr\_t)arg);

sem\_wait(&mutex);

rcount+=1;

if(rcount==1)

sem\_wait(&writeblock);

sem\_post(&mutex);

printf("Data read by the reader %d is %d\n",f,data);

sleep(1);

sem\_wait(&mutex);

rcount-=1;

if(rcount==0)

sem\_post(&writeblock);

sem\_post(&mutex);

}

void \*writer(void \*arg)

{

int f;

f = ((intptr\_t)arg);

sem\_wait(&writeblock);

data++;

printf("Data writen by the writer %d is %d\n",f,data);

sleep(1);

sem\_post(&writeblock);

}

int main()

{

int i,b;

pthread\_t rtid[5],wtid[5];

sem\_init(&mutex,0,1);

sem\_init(&writeblock,0,1);

for(i=0;i<=2;i++)

{

pthread\_create( &wtid[i],NULL,writer,(void \*)(intptr\_t)i );

pthread\_create( &rtid[i],NULL,reader,(void \*)(intptr\_t)i );

}

for(i=0;i<=2;i++)

{

pthread\_join(wtid[i],NULL);

pthread\_join(rtid[i],NULL);

}

return 0;

}

**OUTPUT-**

