**DIGITAL ASSIGNMENT – 3**

Name: Gagan Chordia

Reg.No: 19BCE0788

Slot: L9+L10

**Implement the solution for reader – writer’s problem.**

C Code:

#include<stdio.h>

#include<semaphore.h>

#include<pthread.h>

sem\_t mutex, writeBlock;

int data = 0, readCount = 0;

void \*reader(void \*read){

int x = ((int) read);

sem\_wait(&mutex);

readCount++;

if(readCount == 1){

sem\_wait(&writeBlock);

}

sem\_post(&mutex);

printf("\nData is read by reader %d is %d\n", x, data);

sleep(1);

sem\_wait(&mutex);

readCount--;

if(readCount == 0){

sem\_post(&writeBlock);

}

sem\_post(&mutex);

}

void \*writer(void \*write){

int x = ((int) write);

sem\_wait(&writeBlock);

data++;

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

sleep(1);

sem\_post(&writeBlock);

}

void main(){

pthread\_t readThread[5], writeThread[5];

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

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

for (int i = 0; i < 5; i++){

pthread\_create(&writeThread[i], NULL, writer, (void \*)i);

pthread\_create(&readThread[i], NULL, reader, (void \*)i);

}

for(int i = 0; i < 5; i++){

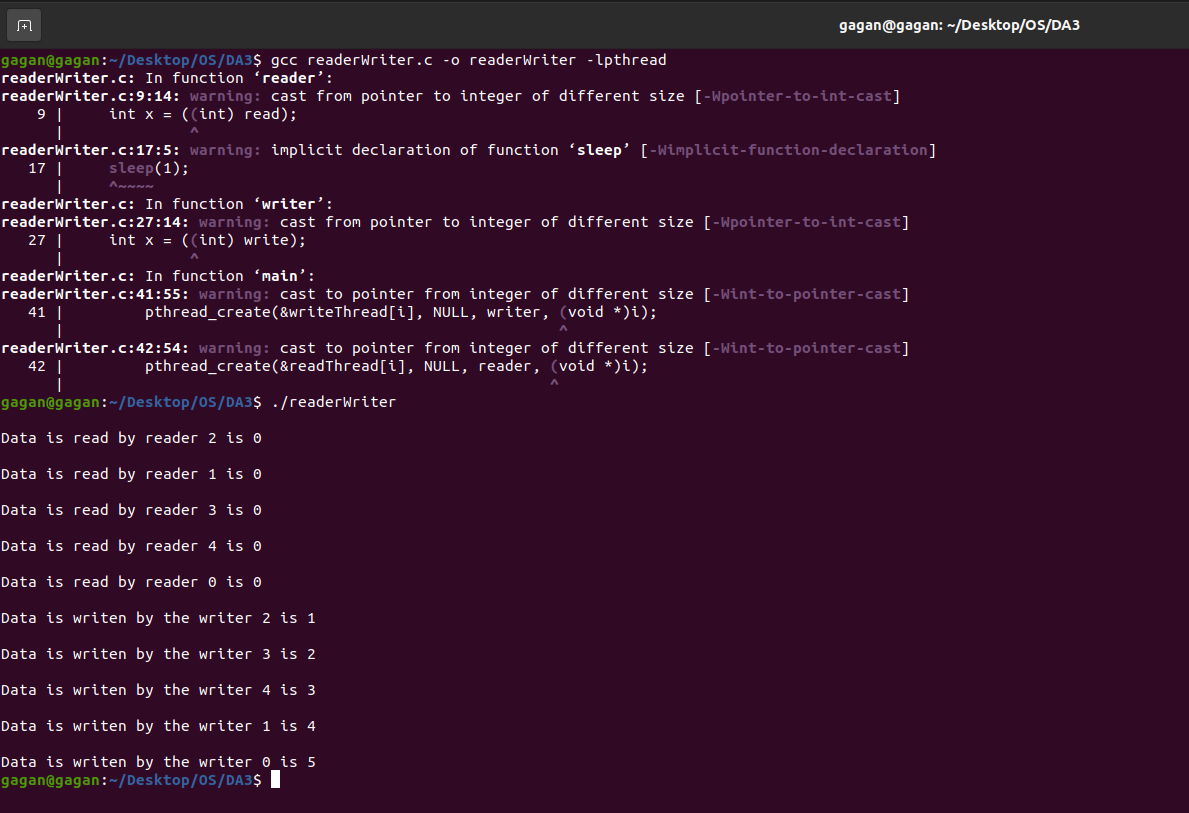
pthread\_join(writeThread[i], NULL);

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

}

}

Output:



**Implement the solution for dining philosopher’s problem.**

C Code:

#include<stdio.h>

#define n 5

int compltedDinner = 0,i;

struct chopSticks{

int inUse;

}CSAvailability[n];

struct philosp{

int left;

int right;

}status[n];

void toDine(int philosopherNo){

if(status[philosopherNo].left==10 && status[philosopherNo].right==10){

printf("Philosopher %d completed his dinner\n",philosopherNo+1);

}

else if(status[philosopherNo].left==1 && status[philosopherNo].right==1){

printf("Philosopher %d completed his dinner\n",philosopherNo+1);

status[philosopherNo].left = status[philosopherNo].right = 10;

int otherChopStick = philosopherNo-1;

if(otherChopStick== -1){

otherChopStick=(n-1);

}

CSAvailability[philosopherNo].inUse = CSAvailability[otherChopStick].inUse = 0;

printf("Philosopher %d has used chopsticks %d and %d\n",philosopherNo+1,philosopherNo+1,otherChopStick+1);

compltedDinner++;

}

else if(status[philosopherNo].left==1 && status[philosopherNo].right==0){

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

if(CSAvailability[philosopherNo].inUse==0){

CSAvailability[philosopherNo].inUse = status[philosopherNo].right = 1;

printf("Philosopher %d is using chopstick %d\n",philosopherNo+1,philosopherNo+1);

}else{

printf("Philosopher %d is waiting for chopstick %d\n",philosopherNo+1,philosopherNo+1);

}

}else{

int temp = philosopherNo;

philosopherNo-=1;

if(philosopherNo== -1){

philosopherNo=(n-1);

}

if(CSAvailability[philosopherNo].inUse == 0){

CSAvailability[philosopherNo].inUse = status[temp].right = 1;

printf("Philosopher %d is using chopstick %d\n",temp+1,philosopherNo+1);

}else{

printf("Philosopher %d is waiting for chopstick %d\n",temp+1,philosopherNo+1);

}

}

}

else if(status[philosopherNo].left==0){

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

if(CSAvailability[philosopherNo-1].inUse==0){

CSAvailability[philosopherNo-1].inUse = status[philosopherNo].left = 1;

printf("Philosopher %d is using chopstick %d\n",philosopherNo,philosopherNo+1);

}else{

printf("Philosopher %d is waiting for chopstick %d\n",philosopherNo+1,philosopherNo);

}

}else{ //except last philosopher case

if(CSAvailability[philosopherNo].inUse == 0){

CSAvailability[philosopherNo].inUse = status[philosopherNo].left = 1;

printf("Philosopher %d is using chopstick %d\n",philosopherNo+1,philosopherNo+1);

}else{

printf("Philosopher %d is waiting for chopstick %d\n",philosopherNo+1,philosopherNo+1);

}

}

}

}

void main(){

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

CSAvailability[i].inUse=status[i].left=status[i].right=0;

while(compltedDinner<n){

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

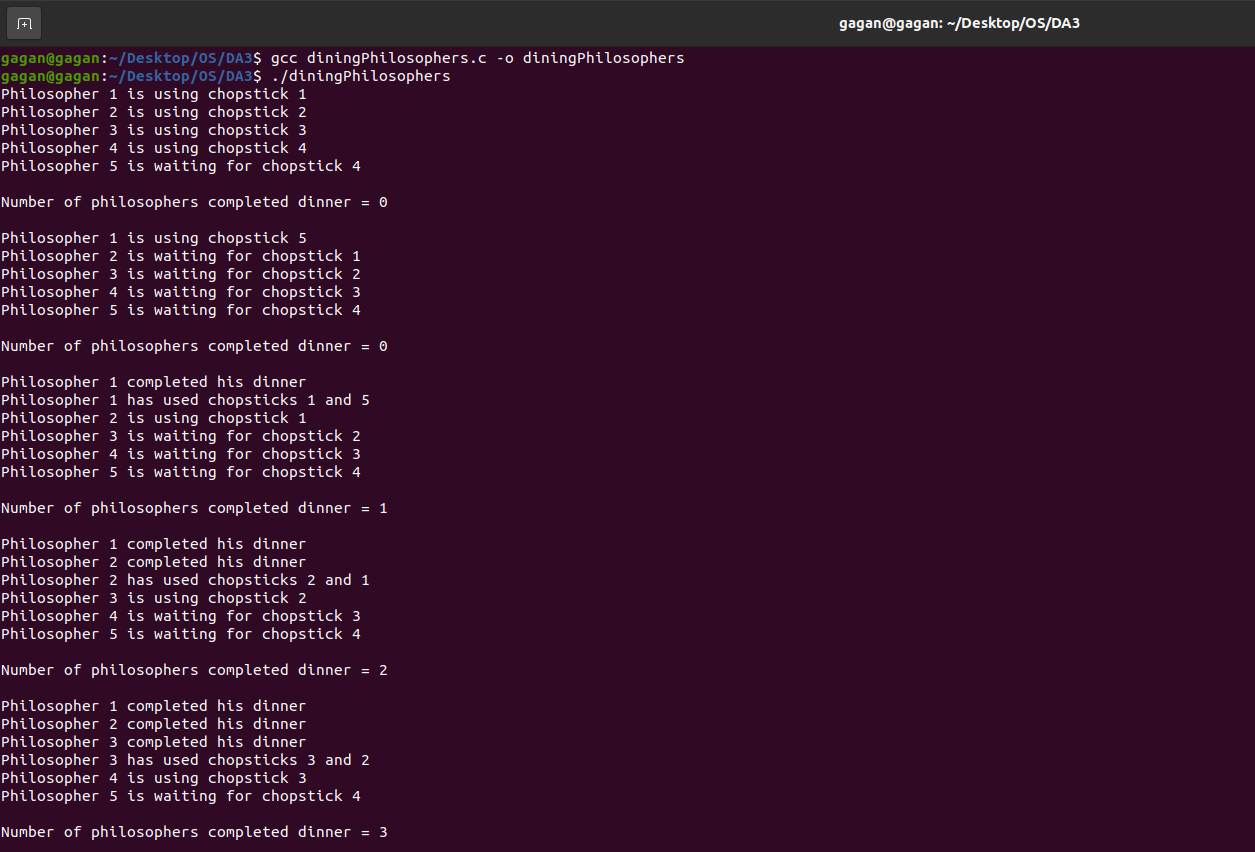
toDine(i);

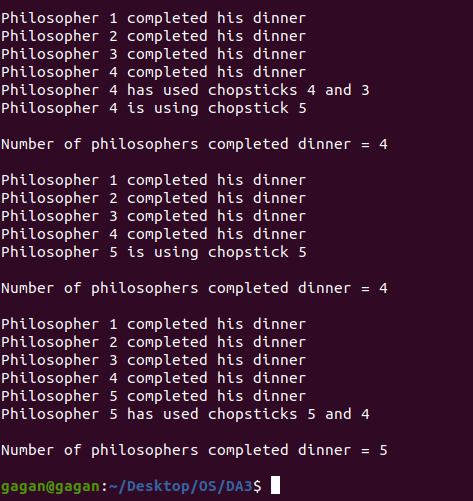
printf("\nNumber of philosophers completed dinner = %d\n\n",compltedDinner);

}

}

Output:





**Implement the solution for producer consumer problem**

C Code:

#include<stdio.h>

#include<stdlib.h>

int mutex = 1, full = 0, empty = 3, y = 0;

int wait(int x){

return --x;

}

int signal(int x){

return ++x;

}

void producer(){

mutex = wait(mutex);

full = signal(full);

empty = wait(empty);

y++;

printf("\n\nProducer Produced: %d", y);

mutex = signal(mutex);

}

void consumer(){

mutex = wait(mutex);

full = wait(full);

empty = signal(empty);

printf("\n\nConsumer Consumed: %d", y);

y--;

mutex = signal(mutex);

}

void main(){

int a, flag = 1;

while(flag){

printf("\n\nPress 1 to produce.\n\nPress 2 to consume.\n\nPress 3 to exit\n");

scanf("%d",&a);

switch(a){

case 1:

if(mutex == 1 && empty != 0){

producer();

}else{

printf("\nBuffer is full");

}

break;

case 2:

if(mutex == 1 && full != 0){

consumer();

}else{

printf("\nBuffer is empty");

}

break;

case 3:

exit(0);

flag = 0;

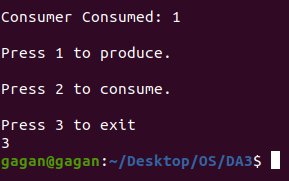
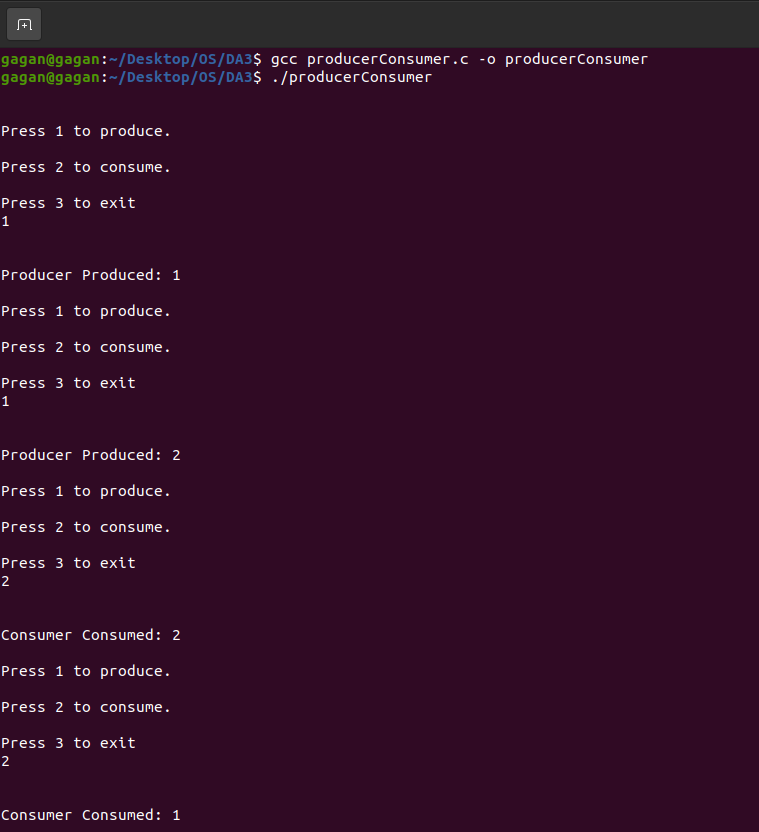
break;

}

}

}

Output:



**The analogy is based upon a hypothetical barber shop with one barber. There is a barber shop which has one barber, one barber chair, and n chairs for waiting for customers if there are any to sit on the chair.**

* **If there is no customer, then the barber sleeps in his own chair.**
* **When a customer arrives, he has to wake up the barber.**
* **If there are many customers and the barber is cutting a customer’s hair, then the remaining customers either wait if there are empty chairs in the waiting room or they leave if no chairs are empty.**

C Code:

#include <stdio.h>

#include <stdlib.h>

#include <sys/stat.h>

#include <time.h>

int seats[2];

int customers[2];

int barber[2];

int freeSeats[2];

void randomWait() {

int delay;

delay = random() % 99;

printf(" - wait: %d\n", delay);

}

void barberProcess() {

int i;

int num;

for (i = 1; i <= 10; ++i) {

printf("Barber %d is trying to get a customer\n",i);

P(customers);

printf("Barber %d is waiting for the seat to become free\n",i);

P(seats);

read(freeSeats[0],&num,sizeof(int));

num++;

write(freeSeats[1],&num,sizeof(int));

printf("Barber %d is increasing the number of free accessSeats to %d\n",i,num);

V(barber);

V(seats);

printf("Barber is now cutting hair %d\n",i);

randomWait();

}

}

void customerProcess() {

int i;

int num;

for (i = 1; i <= 2; ++i) {

printf("New customer trying to find a seat\n");

P(seats);

read(freeSeats[0],&num,sizeof(int));

if (num > 0) {

num--;

write(freeSeats[1],&num,sizeof(int));

printf("Customer left seating in waiting room. The total free accessSeats are now: %d\n",num);

V(customers);

V(seats);

printf("Customer is now waiting for the barber\n");

P(barber);

printf("Customer is now getting a hair cut\n");

}else{

write(freeSeats[1],&num,sizeof(int));

V(seats);

printf("No free chairs available\n");

}

randomWait();

}

}

void V(int arr[]) {

int a=1;

write(arr[1],&a,sizeof(int));

}

void P(int arr[]) {

int a;

read(arr[0],&a,sizeof(int));

}

void main() {

int i;

pipe(seats);

pipe(customers);

pipe(barber);

pipe(freeSeats);

V(seats);

int num = 3;

write(freeSeats[1],&num,sizeof(int));

if (fork() == 0) {

srand(time(0)+1);

barberProcess();

return;

}

for (i = 1; i <= 5; i++) {

if (fork() == 0) {

srand(time(0)+2\*i);

customerProcess();

return;

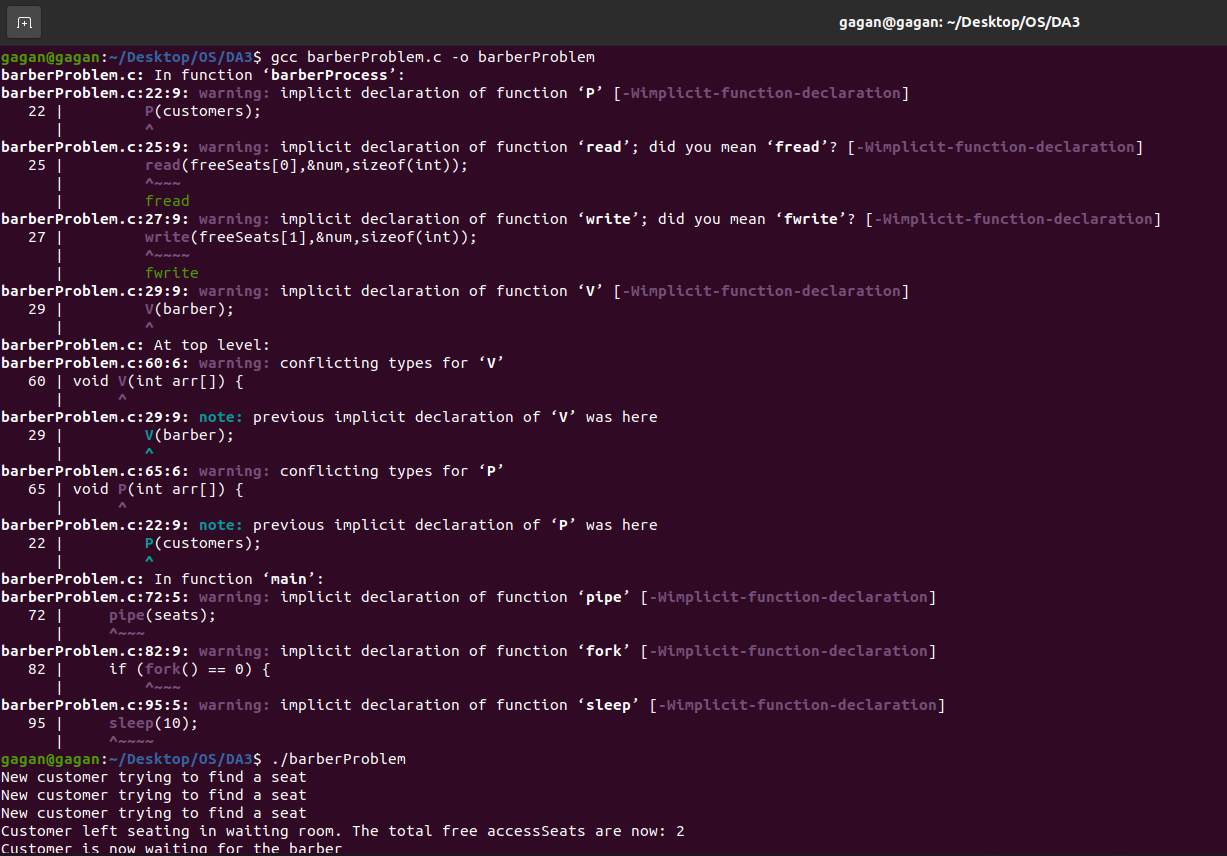
}

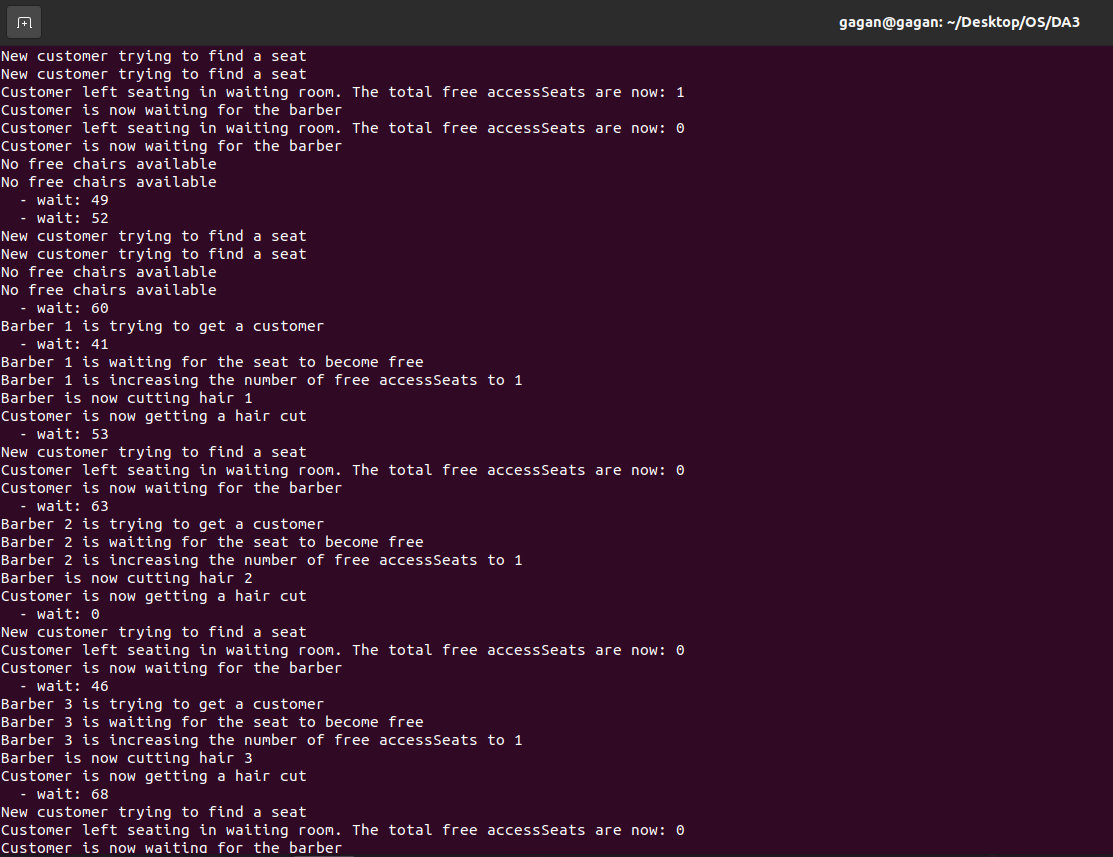
}

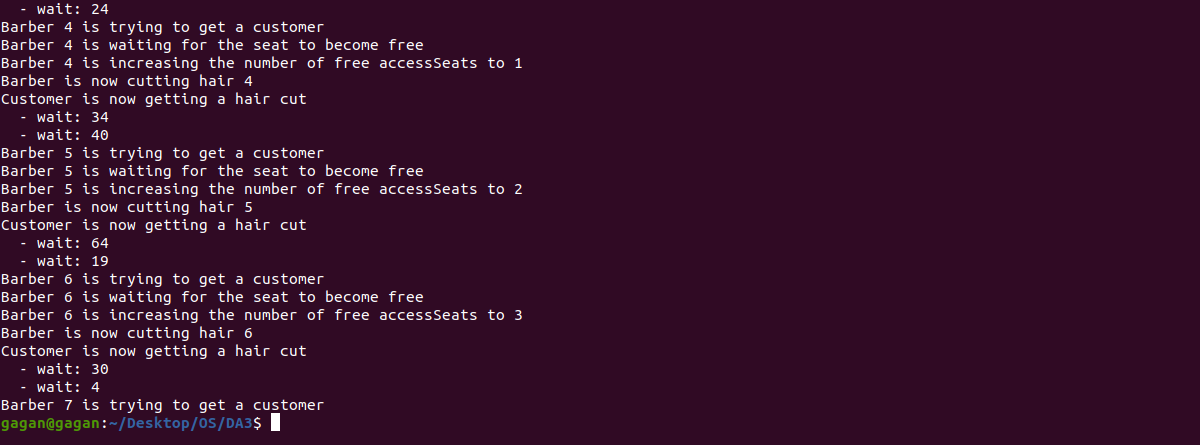
sleep(10);

}

Output:







**A pair of processes involved in exchanging a sequence of integers. The number of integers that can be produced and consumed at a time is limited to 100. Write a Program to implement the producer and consumer problem using POSIX semaphore for the above scenario.**

C Code:

#include<stdio.h>

#include<semaphore.h>

#include<pthread.h>

#include<stdlib.h>

#define buffersize 100

pthread\_mutex\_t mutex;

pthread\_t tidP[100],tidC[100];

sem\_t full,empty;

int counter;

int buffer[buffersize];

void initialize(){

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

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

sem\_init(&empty,1,buffersize);

counter=0;

}

void write(int item){

buffer[counter++]=item;

}

int read(){

return(buffer[--counter]);

}

void \*producer (void \*produce){

int waittime,item,i;

item=rand()%5;

waittime=rand()%5;

sem\_wait(&empty);

pthread\_mutex\_lock(&mutex);

printf("\nProducer produced: %d\n",item);

write(item);

pthread\_mutex\_unlock(&mutex);

sem\_post(&full);

}

void \*consumer(void \*consume){

int waittime,item;

waittime=rand()%5;

sem\_wait(&full);

pthread\_mutex\_lock(&mutex);

item=read();

printf("\nConsumer consumed: %d\n",item);

pthread\_mutex\_unlock(&mutex);

sem\_post(&empty);

}

void main(){

int producers, consumers, i;

initialize();

printf("\nEnter the number of producers: ");

scanf("%d",&producers);

printf("\nEnter the number of consumers: ");

scanf("%d",&consumers);

for(i = 0;i < producers;i++){

pthread\_create(&tidP[i],NULL,producer,NULL);

}

for(i=0;i<consumers;i++){

pthread\_create(&tidC[i],NULL,consumer,NULL);

}

for(i=0;i<producers;i++){

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

}

for(i=0;i<consumers;i++){

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

}

}

Output:

