**Que Code – 13**

A barbershop consists of a waiting room with n chairs and a barber room with one barber chair. If there are no customers to be served, the barber goes to sleep. If a customer enters the barbershop and all chairs are occupied, then the customer leaves the shop. If the barber is busy but chairs are available, then the customer sits in one of the free chairs. If the barber is asleep, the customer wakes up the barber. Write a program to coordinate the barber and the customers.

// shared data

semaphore customers = 0;

semaphore barbers = 0;

semaphore cutting = 0;

semaphore mutex = 1;

int customer1 = 0;

void barber()

{

while(true)

{

wait(customers); //sleep when there are no waiting customers

wait(mutex); //mutex for accessing customers1

customers1 = customers1 - 1;

signal(barbers);

signal(mutex);

cut\_hair();

}

}

void customer()

{

wait(mutex); //mutex for accessing customers1

if (customers1 < n)

{

customers1 = customers1 + 1;

signal(customers);

signal(mutex);

wait(barbers); //wait for available barbers

get\_haircut();

}

else

{

//do nothing (leave) when all chairs are used.

signal(mutex);

}

}

cut\_hair()

{

waiting(cutting);

}

get\_haircut()

{

get hair cut for some time;

signal(cutting);

}

**Que Code – 16**

1. Design a scheduling program that is capable of scheduling many processes that comes in at some time interval and are allocated the CPU not more than 10 time units. CPU must schedule processes having short execution time first. CPU is idle for 3 time units and does not entertain any process prior this time. Scheduler must maintain a queue that keeps the order of execution of all the processes. Compute average waiting and turnaround time.

|  |
| --- |
| #include<stdlib.h> |
| #include<conio.h>  #include<stdio.h> |
|  |
| int main() |
| { |
|  |
| int count,i,j,m=0,n,y=0,time,remain=0,min,flag=0; |
| int wait\_time=0,turn\_a\_time=0,a\_time[10],b\_time[10],p[10],z[10]; |
| float k=0,x=0; |
| printf("Enter number of Process:\t "); |
| scanf("%d",&n); |
| printf("\n\tArrival time should be greater than 2 as CPU remains idle for first 3 secs.\n"); |
| printf("\n\tBurst time should be less tha 10\n"); |
| for(count=0;count<n;count++) |
| { |
| printf("Enter Arrival Time and Burst Time for Process Process Number %d :",count+1); |
| scanf("%d",&a\_time[count]); |
| scanf("%d",&b\_time[count]); |
| } |
|  |
| for(i=0;i<n;i++) |
| { |
| if(a\_time[i]==0) |
| { |
| printf("\nS INVALID ARRIVAL TIME\n"); |
| getch(); |
| exit(1); |
| } |
| } |
| for(i=0;i<n;i++) |
| { |
| if(a\_time[i]<3) |
| { |
| printf("\nS INVALID Arrival Time it should be greater than 3\n"); |
| getch(); |
| exit(1); |
| } |
| } |
| printf("\n\n\tProcess\t|Turnaround Time|Waiting Time\n\n"); |
| printf("\t===========================================\n"); |
| for(i=0;i<n;i++) |
| { |
| m=m+b\_time[i]; |
| } |
| min=m; |
| time=m; |
| for(i=0;i<n;i++) |
| { |
| if(a\_time[i]<time) |
| { |
| time=a\_time[i]; |
| } |
| } |
| for(i=time;i<=m;i=i+b\_time[j]) |
| { |
| min=m; |
| remain=0; |
| flag=0; |
|  |
| for(count=0;count<n;count++) |
| { |
|  |
| if(a\_time[count]<=i) |
| { |
|  |
| if(b\_time[count]<min) |
| { |
|  |
| min = b\_time[count]; |
| j=count; |
| flag=1; |
| } |
| remain=1; |
| } |
| } |
| if(flag==1&&remain==1) |
| { |
| wait\_time=i-a\_time[j]; |
| turn\_a\_time=wait\_time+b\_time[j]; |
| printf("\tP[%d]\t|\t%d\t|\t%d\n",j+1,turn\_a\_time,wait\_time); |
| k=k+wait\_time; |
| x=x+turn\_a\_time; |
|  |
| a\_time[j]=m+1; |
| p[y]=j+1; |
| z[y]=i; |
| y++; |
| } |
| } |
| printf("\n\nAverage Waiting Time= %.2f\n",k/n); |
| printf("Avg Turnaround Time = %.2f",x/n); |
| printf("\n\n\nTotal time taken by processor to complete all the jobs : %d",m); |
| printf("\n\nQueue for order of execution:\n"); |
| printf("\n\nProcess "); |
|  |
| for(i=0;i<n;i++) |
| { |
| printf(" P[%d] ",p[i]); |
| if(i==(n-1)) |
| { |
| printf("End"); |
| } |
| } |
|  |
|  |
| return 0; |
| } |