

**Term work**

**on**

**Operating Systems**

**(PCS 506)**

**2021-22**

**Submitted to: Submitted by:**

Ms. Himadri vaidya Name:Naman Joshi

University Roll.No:2018502

# ClassRoll-NO:28 Section: I

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**GRAPHIC ERA HILL UNIVERSITY, DEHRADUN**

**ACKNOWLEDGMENT**

I would like to particularly thank my Operating Systems Lab Faculty Ms. Himadri vaidya for his patience, support and encouragement throughout the completion of this Term work.

At last but not the least I greatly indebted to all other persons who directly or indirectly helped me during this course.

**Name: Naman Joshi**

**University. Roll No: 20181502**

**B.Tech CSE-I-V Sem**

**Session: 2021-22 GEHU, Dehradun**

**Program 1**

**Q1. Write a C program to demonstrate the use of fork() System call**

## SOURCE CODE

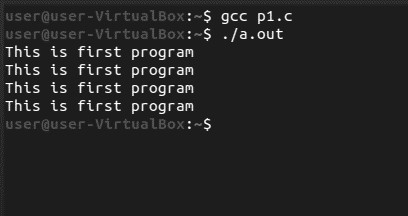
#include <stdio.h> #include<unistd.h> int main(){ fork(); fork();

printf("This is first program\n");

return 0;

}

## OUTPUT:



## Program 2

**Q2. C Program in which Parent Process Computes the SUM OF EVEN NUMBERS and Child Process Computes the sum of ODD NUMBERS**

**stored in array using fork () . First the child process should print its answer i.e sum of odd numbers, then parent should print its answer, i.e sum of even numbers.**

### SOURCE CODE

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h> #include<sys/wait.h>

int main(){

int p,n,i,sum=0; printf("Enter size:"); scanf("%d",&n);

int arr[n];

printf("Enter elements: "); for(i=0;i<n;i++) scanf("%d",&arr[i]);

p=fork();

if(p<0){

printf("failed to create child\n"); exit(1);

}

else if(p==0){

printf("\nchild process :: "); for(i = 0;i<n;i++) if(arr[i]%2!=0)

{

sum+=arr[i];

}

printf("\n\tsum: %d",sum);

} else if(p>0){

wait(NULL);

printf("\nParent process :: "); for(i = 0;i<n;i++) if(arr[i]%2==0){

sum+=arr[i];

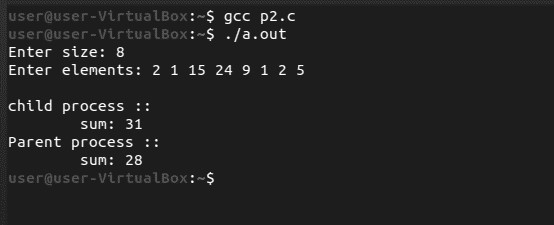
} printf("\n\tsum: %d\n",sum);

}

return 0;

}

### OUTPUT:



### 

### Program-3

**Q3. C program to Implement the Orphan Process and Zombie Process.**

#### SOURCE CODE

#include<stdio.h> #include<unistd.> int main(){

pid\_t pid; pid=fork(); if(pid==0){ sleep(6);

printf("\n I m Child. My PID = %d And PPID = %d", getpid(),getppid());

} else

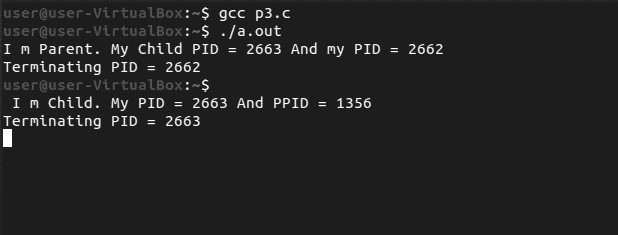
{

printf("I m Parent. My Child PID = %d And my PID = %d",pid,getpid());

}

printf("\nTerminating PID = %d\n",getpid()); return 0;

#### OUTPUT:



### Program-4

**Q. C program to Implement FCFS CPU Scheduling Algorithm**

#### SOURCE CODE

#include<stdio.h> int main() {

int n;

printf("Enter the size of array: \n");

scanf("%d",&n);

int at[n];

printf("enter the arrival time: \n"); for(int i=0;i<n;i++) { scanf("%d",&at[i]); }

int bt[n];

printf("enter the burst time: \n"); for(int i=0;i<n;i++)

{

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

}

int ct[n],i=0;

printf("the complition time: \n"); for(i=0;i<n;i++)

{

ct[i] += bt[i];

}

printf("%d ",ct[i]);

printf("\nthe turn around time: \n"); int tat[n],sum=0;

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

{

tat[i]=ct[i]-at[i]; sum += tat[i];

}

printf("%d ",tat[i]); printf("\nthe waiting time: \n"); int wt[n],sum1=0; for(i=0;i<n;i++)

{

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

sum1 += wt[i];

}

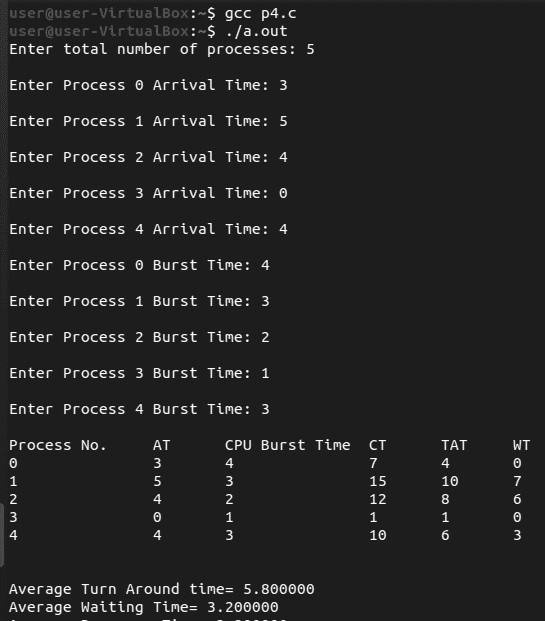
printf("%d ",wt[i]);

int avgtat= sum/n;

int avgwt = sum1/n;

printf("avg of tat: %d\n",avgtat); printf("avg of wt: %d\n",avgwt); }

#### Output:



### Program-5

**Q. C program to Implement Sortest job first non-premptive Scheduling Algorithm**

#### SOURCE CODE

#include<stdio.h> int main()

{ int

bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,

temp;

float avg\_wt,avg\_tat;

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

printf("\nEnter Burst Time:\n"); for(i=0;i<n;i++)

{

printf("p%d:",i+1);

scanf("%d",&bt[i]); p[i]=i+1;

}

//sorting of burst times for(i=0;i<n;i++)

{ pos=i;

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

{

if(bt[j]<bt[pos])

pos=j;

}

temp=bt[i]; bt[i]=bt[pos];

bt[pos]=temp;

temp=p[i]; p[i]=p[pos]; p[pos]=temp;

}

wt[0]=0;

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

{ wt[i]=0; for(j=0;j<i;j++)

wt[i]+=bt[j];

total+=wt[i];

}

avg\_wt=(float)total/n; total=0;

printf("\nProcesst Burst Time tWaiting

Timet Turnaround Time");

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

{

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

total+=tat[i];

printf("\np%d\t\t %d\t\t %d\t\t\

%d",p[i],bt[i],wt[i],tat[i]);

}

|  |  |
| --- | --- |
| avg\_tat=(float)total/n; |  |
| printf("\n\nAverage Time=%f",avg\_wt); | Waiting |
| printf("\nAverage  Time=%f\n",avg\_tat);  } | Turnaround |

**Output:**



**Program-6**

**Q. C program to Implement Sortest job first premptive Scheduling Algorithm**

#### SOURCE CODE

#include <stdio.h>

int main()

{

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

int i, smallest, count = 0, time, limit; double wait\_time = 0,

turnaround\_time = 0, end;

float average\_waiting\_time, average\_turnaround\_time;

printf("\nEnter the Total Number of Processes:\t");

scanf("%d", &limit);

printf("\nEnter Details of %d Processes\n", limit);

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

{

printf("\nEnter Arrival Time:\t"); scanf("%d", &arrival\_time[i]); printf("Enter Burst Time:\t"); scanf("%d", &burst\_time[i]);

temp[i] = burst\_time[i];

}

burst\_time[9] = 9999;

for(time = 0; count != limit; time++)

{

smallest = 9;

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

{

if(arrival\_time[i] <= time && burst\_time[i] < burst\_time[smallest] &&

burst\_time[i] > 0)

{

smallest = i;

}

}

burst\_time[smallest]--;

if(burst\_time[smallest] == 0)

{

count++;

end = time + 1;

wait\_time = wait\_time + end - arrival\_time[smallest] - temp[smallest];

turnaround\_time = turnaround\_time + end - arrival\_time[smallest];

}

}

average\_waiting\_time = wait\_time /

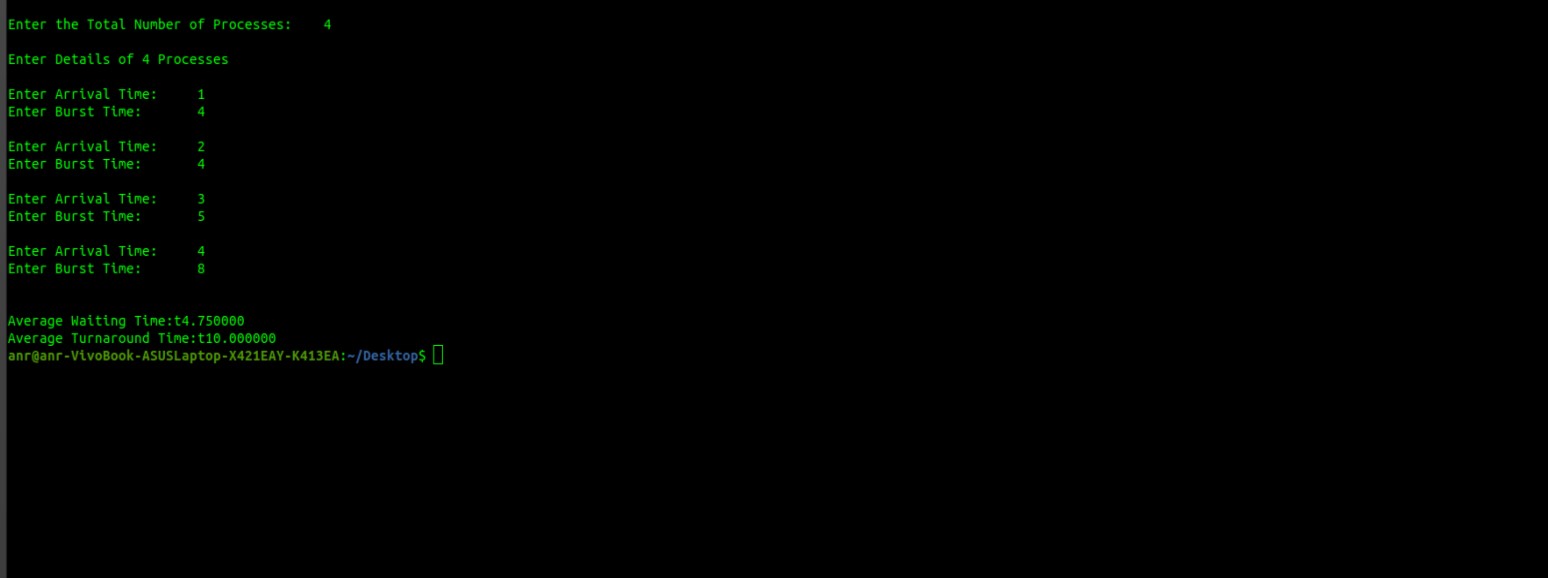
limit;

average\_turnaround\_time = turnaround\_time / limit; printf("\n\nAverage Waiting Time:t%lf\n", average\_waiting\_time); printf("Average Turnaround Time:t%lf\n", average\_turnaround\_time);

return 0;

}

#### Output:



### Program-7

**Q. C program to Implement Priority Scheduling Algorithm**

#### SOURCE CODE

#include<stdio.h>

int main() {

int bt[20],p[20],wt[20],

tat[20],pr[20],i,j,n,

total=0,pos,temp,avg\_wt,avg\_tat;

printf("Enter Total Number of Process:"); scanf("%d",&n);

printf("\nEnter Burst Time and Priority\n");

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

{

printf("\nP[%d]\n",i+1); printf("Burst Time:"); scanf("%d",&bt[i]); printf("Priority:"); scanf("%d",&pr[i]);

p[i]=i+1; //contains process number

}

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

{ pos=i;

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

{

if(pr[j]<pr[pos])

pos=j;

}

temp=pr[i];

pr[i]=pr[pos];

pr[pos]=temp;

temp=bt[i];

bt[i]=bt[pos];

bt[pos]=temp;

temp=p[i];

p[i]=p[pos];

p[pos]=temp;

}

wt[0]=0; //waiting time for first process is0

//calculate waiting time for(i=1;i<n;i++)

{

wt[i]=0; for(j=0;j<i;j++)

wt[i]+=bt[j];

total+=wt[i];

}

avg\_wt=total/n; //average waiting time

total=0;

printf("\nProcess\t Burst Time \tWaiting

Time\tTurnaround Time");

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

{

tat[i]=bt[i]+wt[i]; //calculate turnaround time

total+=tat[i]; printf("\nP[%d]\t\t %d\t\t %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);

}

avg\_tat=total/n;

printf("\n\nAverageWaiting

Time=%d",avg\_wt);

printf("\nAverage Turnaround

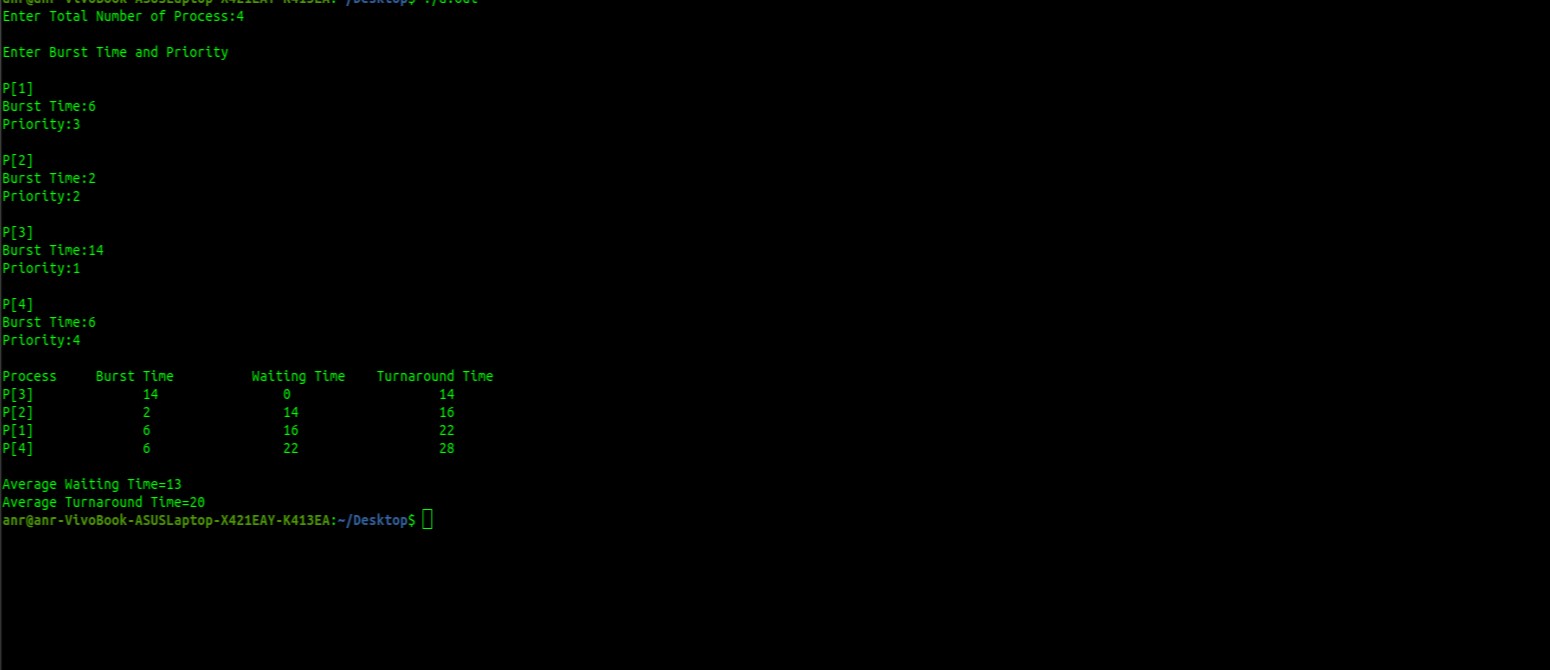
Time=%d\n",avg\_tat);

return 0;

}

#### Output:

#### 



### Program-8

**Q. C program to Implement Round robin Scheduling Algorithm**

#### SOURCE CODE

#include<stdio.h>

void main()

{

int i, NOP, sum=0,count=0, y, quant, wt=0, tat=0, at[10], bt[10], temp[10]; float avg\_wt, avg\_tat;

printf(" Total number of process in the system: ");

scanf("%d", &NOP);

y = NOP;

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

{

printf("\n Enter the Arrival and Burst time of the Process[%d]\n", i+1);

printf(" Arrival time is: \t"); // Accept arrival time

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

printf(" \nBurst time is: \t"); // Accept the Burst time scanf("%d", &bt[i]);

temp[i] = bt[i]; // store the burst time in temp array

}

printf("Enter the Time Quantum for the

process: \t");

scanf("%d", &quant); printf("\n Process No \t\t Burst Time \t\t TAT

\t\t Waiting Time ");

for(sum=0, i = 0; y!=0; )

{

if(temp[i] <= quant && temp[i] > 0) // define

the conditions

{

sum = sum + temp[i]; temp[i] = 0;

count=1;

}

else if(temp[i] > 0)

{

temp[i] = temp[i] - quant; sum = sum + quant;

}

if(temp[i]==0 && count==1)

{

y--; //decrement the process no. printf("\nProcess No[%d] \t\t %d\t\t\t\t %d\t\t\t %d", i+1, bt[i], sum-at[i], sum-at[i]bt[i]); wt = wt+sum-at[i]-bt[i]; tat = tat+sum-at[i]; count =0;

}

if(i==NOP-1)

{ i=0;

}

else if(at[i+1]<=sum)

{ i++; }

else { i=0;

}

}

Turn Around time avg\_wt = wt \* 1.0/NOP;

avg\_tat = tat \* 1.0/NOP; printf("\n Average Turn Around Time: \t%f", avg\_wt);

printf("\n Average Waiting Time: \t%f",

avg\_tat);

}

#### Output:

#### 



## Program-9

**Q. C program to Implement Multilevel queue Scheduling Algorithm**

### SOURCE CODE

#include<stdio.h> int main() {

int p[20],bt[20], su[20], wt[20],tat[20],i, k, n, temp; float wtavg, tatavg;

printf("Enter the number of

processes:");

scanf("%d",&n);

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

{

p[i] = i;

printf("Enter the Burst Time of

Process%d:", i);

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

printf("System/User Process

(0/1) ? ");

scanf("%d", &su[i]);

}

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

for(k=i+1;k<n;k++) if(su[i] > su[k])

{

temp=p[i]; p[i]=p[k]; p[k]=temp; temp=bt[i]; bt[i]=bt[k]; bt[k]=temp; temp=su[i]; su[i]=su[k];

su[k]=temp;

}

wtavg = wt[0] = 0; tatavg = tat[0] = bt[0];

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

{

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

tat[i] = tat[i-1] + bt[i]; wtavg = wtavg + wt[i]; tatavg = tatavg + tat[i];

}

printf("\nPROCESS\t\t SYSTEM/USER

PROCESS \tBURST TIME\tWAITING TIME\tTURNAROUND TIME"); for(i=0;i<n;i++) { printf("\n%d \t\t %d \t\t %d \t\t

%d \t\t %d ",p[i],su[i],bt[i],wt[i],tat[i]); printf("\nAverage Waiting Time is ---

%f",wtavg/n);

printf("\nAverage Turnaround Time is -

-- %f",tatavg/n);

return 0;

}

}

### Output:

### 

