

**1. Write a C program to create a new process that exec a new program using system calls fork(), execlp() & wait()**

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

void main(intargc,char \*arg[])

{

int pid;

pid=fork();

if(pid<0)

{

printf("fork failed");

exit(1);

}

else if(pid==0)

{

execlp("ls","ls",NULL);

exit(0);

}

else

{

printf("\n Process id is -%d\n",getpid());

wait(NULL);

exit(0);

}

}

**2. Write a C program to display PID and PPID using system calls getpid () & getppid ()**

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

void main()

{

int pid;

pid=fork();

if(!pid)

{

printf("Child process...");

printf("\n\nChild PID : %d",getpid());

printf("\nParent PID : %d",getppid());

printf("\n\nFinished with child\n");

}

else

{

wait(NULL);

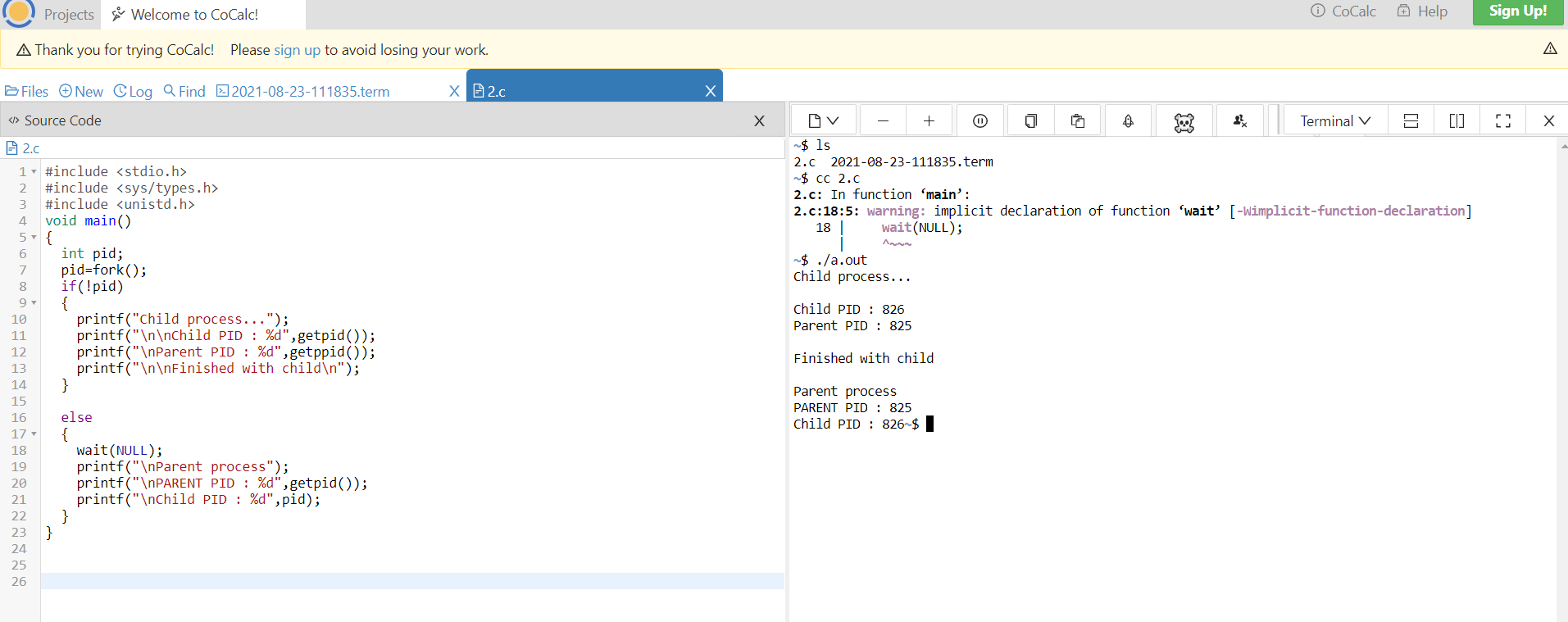
printf("\nParent process");

printf("\nPARENT PID : %d",getpid());

printf("\nChild PID : %d",pid);

}

}



**3. Write a C program using I/O system calls open(), read() & write() to copy contents of one file to another file**

#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

#include<fcntl.h>

void main()

{

char buff;

int fd,fd1;

fd=open("one.txt",O\_RDONLY);

fd1=open("two.txt",O\_WRONLY|O\_CREAT);

while(read(fd,&buff,1))

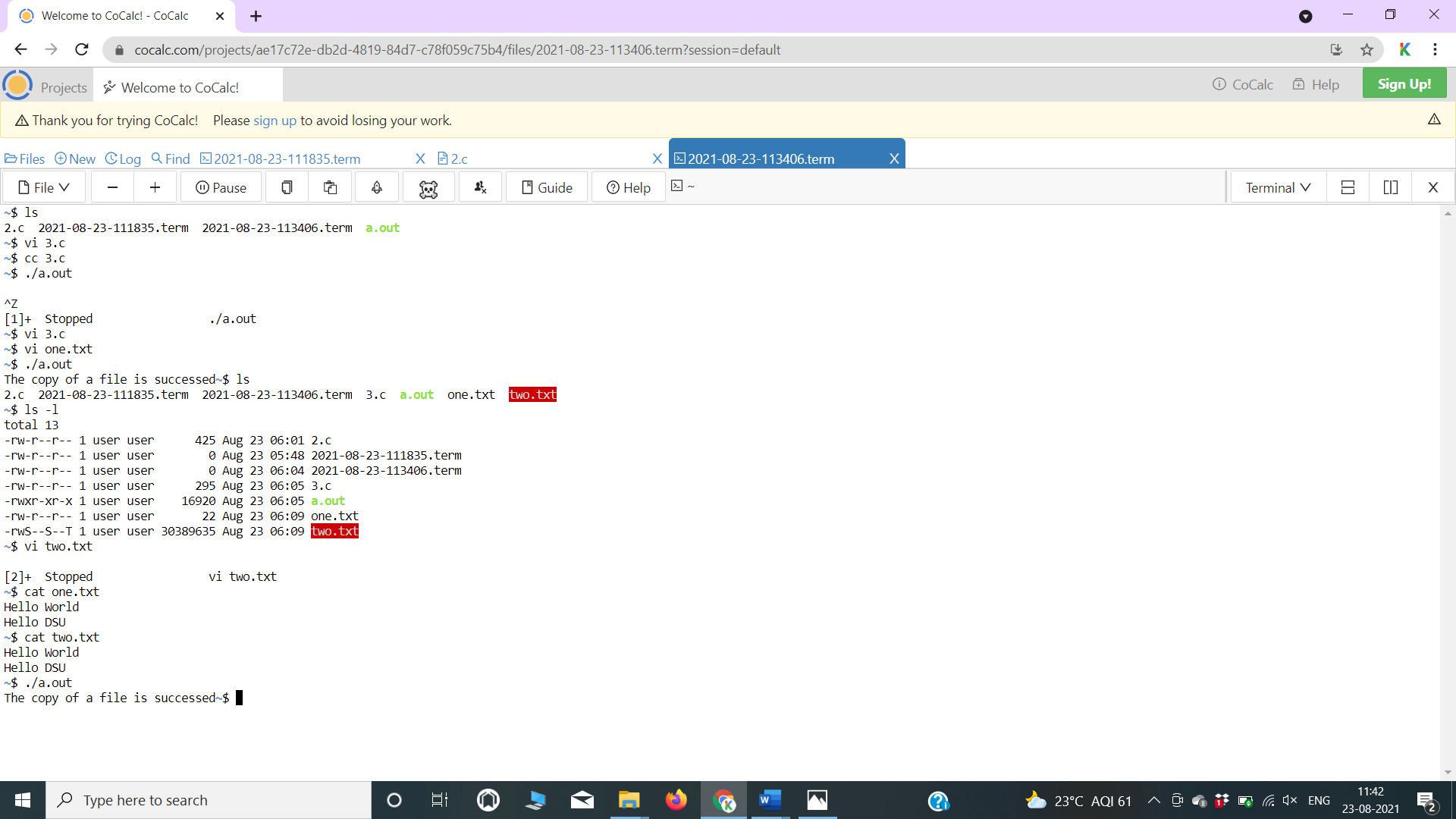
write(fd1,&buff,1);

printf("The copy of a file is successed");

close(fd);

close(fd1);

}



**4. Write a C program to implement multithreaded program using pthreads.**

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

#define NUM\_THREADS 5

void \*PrintHello(void \*threadid)

{

long tid;

tid = (long)threadid;

printf("Hello World! It's me, thread #%ld!\n", tid);

pthread\_exit(NULL);

}

int main(int argc, char \*argv[])

{

pthread\_t threads[NUM\_THREADS];

int rc;

long t;

for(t=0;t<NUM\_THREADS;t++){

printf("In main: creating thread %ld\n", t);

rc = pthread\_create(&threads[t], NULL, PrintHello, (void \*)t);

if (rc){

printf("ERROR; return code from pthread\_create() is %d\n", rc);

exit(-1);

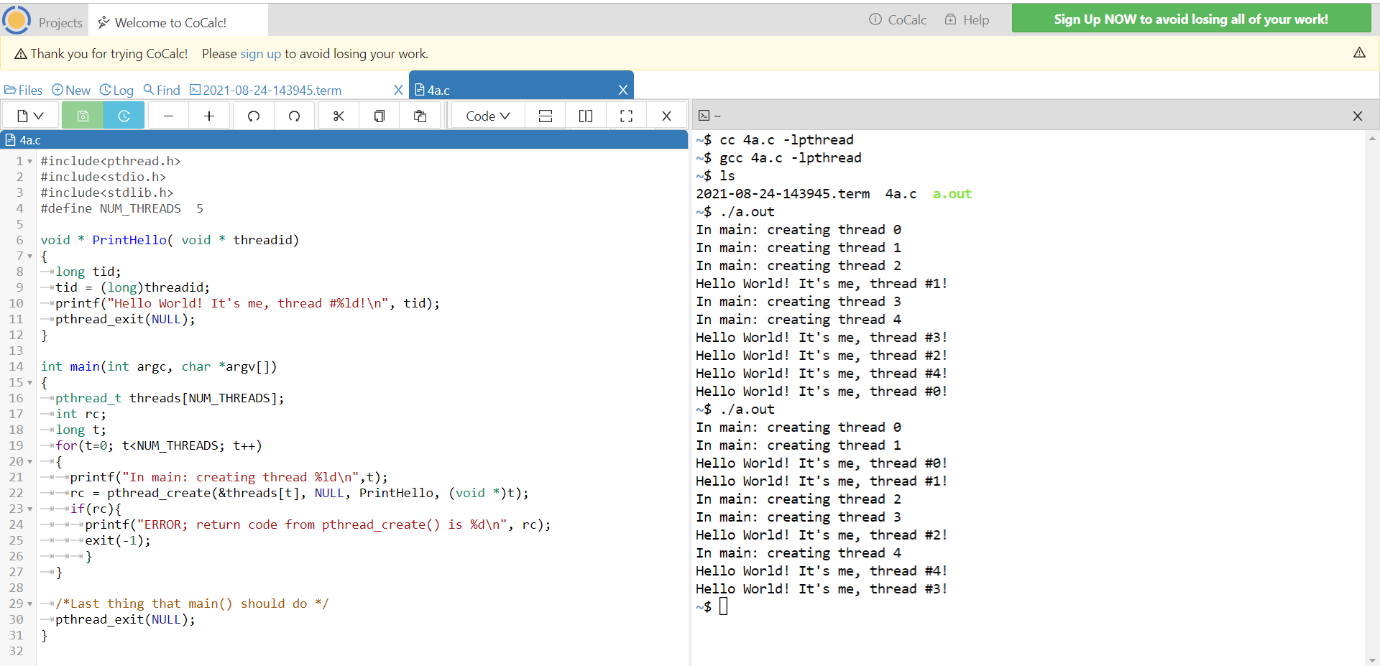
}

}

/\* Last thing that main() should do \*/

pthread\_exit(NULL);

}



**5a. Write a C program to simulate FCFS CPU scheduling ALGORITHM**

#include<stdio.h>

int main()

{

int bt[10],at[10],tat[10],wt[10],ct[10]={0};

int n, sum=0;

float avgwt=0, avgtat=0;

printf("Enter the number of processes");

scanf("%d",&n);

printf("\nEnter arrival time and burst time for each process\n");

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

{

printf("Arrival time of process %d",i+1);

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

printf("Burst time of process %d",i+1);

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

printf("\n");

}

//calculate the CT

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

{

sum=sum+bt[i];

ct[i]=ct[i]+sum;

}

//calculate WT and TAT

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

{

tat[k]=ct[k]-at[k];

avgtat=avgtat+tat[k];

}

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

{

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

avgwt=avgwt+wt[j];

}

printf("P#\tAt\tBT\tCT\tTAT\tWT\n");

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

{

printf("P%d\t%d\t%d\t%d\t%d\t%d",i+1,at[i],bt[i],ct[i],tat[i],wt[i]);

printf("\n");

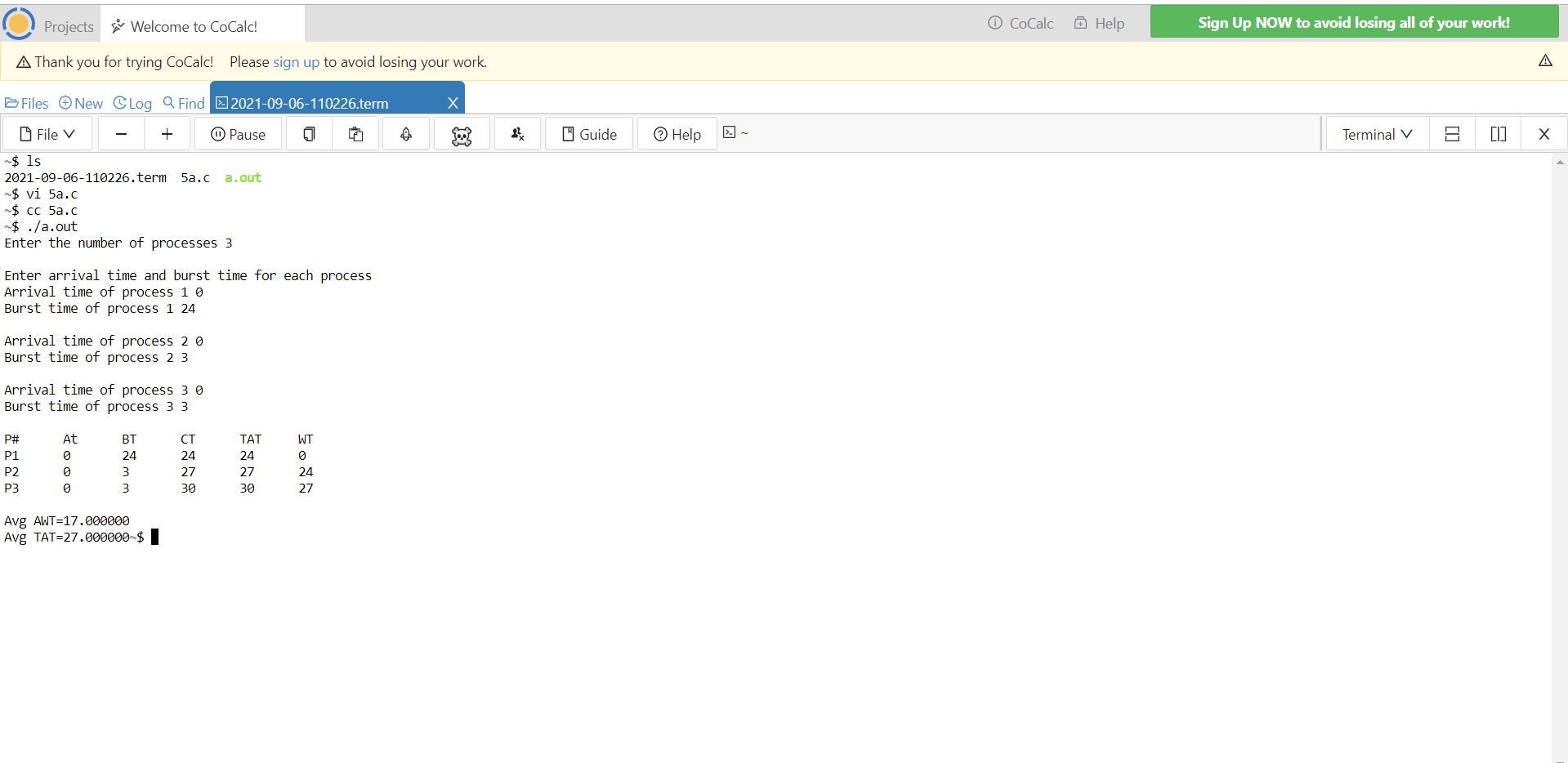
}

printf("\nAvg AWT=%f",avgwt/n);

printf("\nAvg TAT=%f",avgtat/n);

return 0;

}



**5b. Write a C program to simulate SJF CPU scheduling algorithm**

#include<stdio.h>

int main()

{

int p[20], bt[20], wt[20], tat[20], n, temp;

float wtavg=0, tatavg=0;

printf("\nEnter the number of processes -- ");

scanf("%d", &n);

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

{

p[i]=i;

printf("Enter Burst Time for Process %d -- ", i);

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

}

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

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

if(bt[i]>bt[k])

{

temp=bt[i];

bt[i]=bt[k];

bt[k]=temp;

temp=p[i];

p[i]=p[k];

p[k]=temp;

}

wt[0] = 0;

tat[0] = bt[0];

for(int 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("\n\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");

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

printf("\n\t P%d \t\t %d \t\t %d \t\t %d", p[i], bt[i], wt[i], tat[i]);

printf("\nAverage Waiting Time -- %f", wtavg/n);

printf("\nAverage Turnaround Time -- %f", tatavg/n);

return 0;}

**5c. Write a C program to simulate Priority CPU scheduling algorithm**

#include<stdio.h>

int main()

{

int p[20],bt[20],pri[20], wt[20],tat[20], n, temp;

float wtavg=0, tatavg=0;

printf("Enter the number of processes --- ");

scanf("%d",&n);

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

{

p[i] = i;

printf("Enter the Burst Time & Priority of Process %d --- ",i);

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

}

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

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

if(pri[i] > pri[k])

{

temp=p[i];

p[i]=p[k];

p[k]=temp;

temp=bt[i];

bt[i]=bt[k];

bt[k]=temp;

temp=pri[i];

pri[i]=pri[k];

pri[k]=temp;

}

wt[0] = 0;

tat[0] = bt[0];

for(int 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];

return 0;

}

printf("\nPROCESS\t\tPRIORITY\tBURST TIME\tWAITING TIME\tTURNAROUND TIME");

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

printf("\n%d \t\t %d \t\t %d \t\t %d \t\t %d ",p[i],pri[i],bt[i],wt[i],tat[i]);

printf("\nAverage Waiting Time is --- %f",wtavg/n);

printf("\nAverage Turnaround Time is --- %f",tatavg/n);

}

**5d. Write a C program to simulate Round Robin CPU scheduling algorithm**

#include<stdio.h>

int main()

{

int wt[10],tat[10],rbt[10],bt[30],ts,n,i,x=0,tot=0;

float totwt=0, totat=0;

printf("Enter the no of processes:");

scanf("%d",&n);

printf("Enter the time quantum:");

scanf("%d",&ts);

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

{

printf("enter burst time for process %d:",i+1);

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

rbt[i]=bt[i]; // Make a copy of burst times bt[] to store remaining burst times.

}

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

tot=tot+bt[i];

while(x!=tot) // Keep traversing processes in round robin manner until all of them are not done.

{

for(i=0;i<n;i++) // Traverse all processes one by one repeatedly

{

if(rbt[i]>ts) // If burst time of a process is greater than ts

{

x=x+ts; // Increase the value of x i.e. shows how much time a process has been processed

rbt[i]=rbt[i]-ts; // Decrease the burst\_time of current process by quantum

}

else

if((rbt[i]<=ts)&&rbt[i]!=0) //If burst time is smaller than or equal to quantum. Last cycle for this process

{

x=x+rbt[i]; // Increase the value of x i.e. shows how much time a process has been processed

wt[i] = x - bt[i]; // Waiting time is current time minus time used by this process

tat[i] = bt[i] + wt[i]; // calculating turnaround time by adding bt[i] + wt[i]

rbt[i]=0;} // As the process gets fully executed make its remaining burst time = 0

}

}

printf("Process\_no\t Burst time\tWait time\tTurn around time\n");

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

printf("%d\t%d\t%d\t%d\n",i+1,bt[i],wt[i], tat[i]);

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

totat=totat+tat[i];

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

totwt = totwt + wt[i];

printf("average waiting time=%f",totwt/n);

printf("\naverage turnaround time=%f",totat/n);

return 0;

}

**6. Write a C program to simulate producer-consumer problem using Semaphores.**

#include<stdio.h>

#include<stdlib.h>

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

void producer();

void consumer();

int wait(int);

int signal(int);

int main()

{

int n;

do

{

printf("\n1.producer\n2.consumer\n3.exit\n");

printf("\nenter ur choice");

scanf("%d",&n);

switch(n)

{

case 1:

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

producer();

else

printf("buffer is full\n");

break;

case 2:

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

consumer();

else

printf("buffer is empty");

break;

case 3:

exit(0);

break;

}

}while(n!=3);

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 items%d",x);

mutex=signal(mutex);

}

void consumer()

{

mutex=wait(mutex);

full=wait(full);

empty=signal(empty);

printf("\nconsumerconsumes the item %d",x);

x--;

mutex=signal(mutex);

}

**7. Write a C program to simulate Bankers algorithm for the purpose of deadlock avoidance.**

#include<stdio.h>

int main()

{

  int n, m, i, j, instanc, k=0,count1=0,count2=0; //count, k variables are taken for

                                                                            // counting purpose

 printf("\n\t Enter No. of Process:-\n");

  printf("\t\t");

  scanf("%d",&n);                            //Entering No. of Processes

  printf("\n\tEnter No. of Resources:-\n");

  printf("\t\t");

  scanf("%d",&m);                       //No. of Resources

  int   avail[m], max[n][m], allot[n] [m],   need[n][m],completed[n];

  for(i=0;i<n;i++)

  completed[i]=0;                             //Setting Flag for uncompleted Process

  printf("\n\tEnter No. of Available Instances\n");

  for(i=0;i<m;i++)

  {

    printf("\t\t");

    scanf("%d",&instanc);

    avail[i]=instanc;                        // Storing Available instances

  }

  printf("\n\tEnter Maximum No. of instances of resources that a Process need:\n");

  for(i=0;i<n;i++)

  {

    printf("\n\t For P[%d]",i);

    for(j=0;j<m;j++)

     {

        printf("\t");

        scanf("%d",&instanc);

        max[i][j]=instanc;

     }

  }

  printf("\n\t Enter no. of instances already allocated to process of a resource:\n");

  for(i=0;i<n;i++)

  {

    printf("\n\t For P[%d]\t",i);

    for(j=0;j<m;j++)

     {

        printf("\t\t");

        scanf("%d",&instanc);

        allot[i][j]=instanc;

        need[i][j]=max[i][j]-allot[i][j];       //calculating Need of each process

     }

 }

printf("\n\t Safe Sequence is:- \t");

 while(count1!=n)

 {

  count2=count1;

  for(i=0;i<n;i++)

  {

    for(j=0;j<m;j++)

    {

        if(need[i][j]<=avail[j])

          {

            k++;

          }

    }

    if(k==m && completed[i]==0 )

     {

       printf("P[%d]\t",i);

       completed[i]=1;

       for(j=0;j<m;j++)

         {

           avail[j]=avail[j]+allot[i][j];

          }

         count1++;

     }

     k=0;

   }

     if(count1==count2)

     {

     printf("\t\t Stop ..After this.....Deadlock \n");

     break;

   }

 }

 return 0;

}

**8. Write a C program to simulate deadlock detection.**

#include<stdio.h>  
static int mark[20];  
int i,j,np,nr;  
  
int main()  
{  
int alloc[10][10],request[10][10],avail[10],r[10],w[10];  
  
printf("\nEnter the no of process: ");  
scanf("%d",&np);  
printf("\nEnter the no of resources: ");  
scanf("%d",&nr);  
for(i=0;i<nr;i++)  
{  
printf("\nTotal Amount of the Resource R%d: ",i+1);  
scanf("%d",&r[i]);  
}  
  
printf("\nEnter the request matrix:");  
  
for(i=0;i<np;i++)  
for(j=0;j<nr;j++)  
scanf("%d",&request[i][j]);  
  
printf("\nEnter the allocation matrix:");  
for(i=0;i<np;i++)  
for(j=0;j<nr;j++)  
scanf("%d",&alloc[i][j]);

/\*Available Resource calculation\*/  
for(j=0;j<nr;j++)  
{  
avail[j]=r[j];  
for(i=0;i<np;i++)  
{  
avail[j]-=alloc[i][j];  
  
}  
}  
  
//marking processes with zero allocation  
  
for(i=0;i<np;i++)  
{  
int count=0;  
 for(j=0;j<nr;j++)  
   {  
      if(alloc[i][j]==0)  
        count++;  
      else  
        break;  
    }  
 if(count==nr)  
 mark[i]=1;  
}  
// initialize W with avail  
  
for(j=0;j<nr;j++)  
    w[j]=avail[j];  
  
//mark processes with request less than or equal to W  
for(i=0;i<np;i++)  
{  
int canbeprocessed=0;  
 if(mark[i]!=1)  
{  
   for(j=0;j<nr;j++)  
    {  
      if(request[i][j]<=w[j])  
        canbeprocessed=1;  
      else  
         {  
         canbeprocessed=0;  
         break;  
          }  
     }  
if(canbeprocessed)  
{  
mark[i]=1;  
  
for(j=0;j<nr;j++)  
w[j]+=alloc[i][j];  
}  
}  
}  
  
//checking for unmarked processes  
int deadlock=0;  
for(i=0;i<np;i++)  
if(mark[i]!=1)  
deadlock=1;  
  
  
if(deadlock)  
printf("\n Deadlock detected");  
else  
printf("\n No Deadlock possible");  
}

**OUTPUT:**