Operating System Lab Assignment

(UCT 401)

BY:

Surabhi Mishra

(Roll No. 102018051)

&

Aniket Sahran

(Roll No. 102018065)

**TABLE OF CONTENT**

|  |  |
| --- | --- |
| S.NO | QUESTION |
| 1. | Odd/even |
| 2. | Prime number |
| 3. | Fibonacci series |
| 4. | Factorial |
| 5. | Sum of series |
| 6. | Fork system Call |
| 7. | Race Condition using semaphore |
| 8. | Producer- Consumer |
| 9. | CPU Scheduling |
| 10. | Reader- Writer |
| 11. | Deadlock |

## Q1. Shell code to determine if the given number is even or odd.

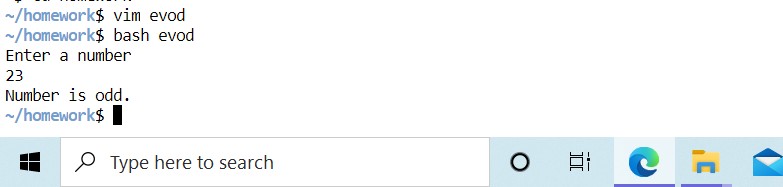
echo "Enter a number" read number

if [ $((number%2)) -eq 0 ] then

echo "Even number." else

echo "Number is odd." Fi

**OUTPUT:**



## Q2. Shell code to determine if the given number is prime or not.

echo "Enter a number" read number

i=2 f=0

while test $i -le `expr $number / 2` do

if test `expr $number % $i` -eq 0 then

f=1 fi

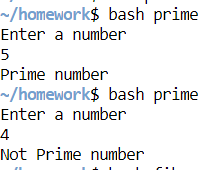
i=`expr $i + 1` done

if test $f -eq 1 then

echo "Not Prime number" else

echo "Prime number" fi

## Output:



## Q3. Shall code to print the Fibonacci series upto the given number of terms.

echo "Enter a number" read n

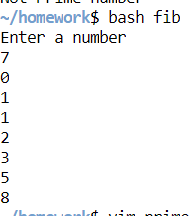
a=0 b=1

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

echo "$a" temp=$((a + b)) a=$b

b=$temp done

## Output:



pg. 4

## Q4. Shell code to find the factorial of the given number

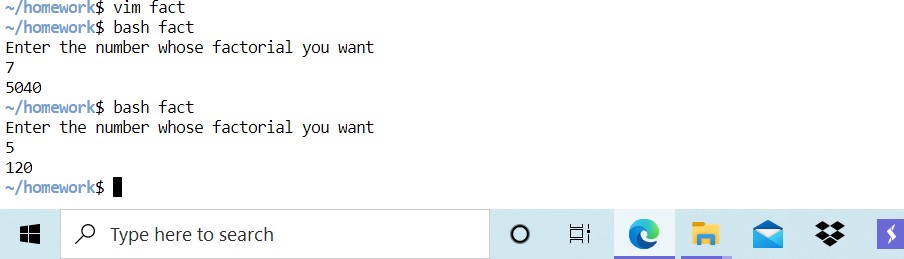
echo "Enter the number whose factorial you want" read n

fact=1 for((i=2;i<=n;i++)) do

fact=$((fact \* i)) done

echo $fact

## Output:



## Q5. Shell code for finding the sum of series.

echo "Enter a number" read n

s=0

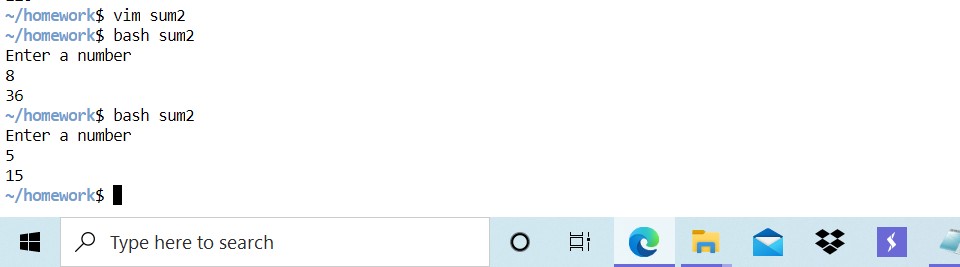
while [ $n -gt 0 ] do

s=$((s + n))

n=$((n - 1)) done

echo $s

## Output:



pg. 6

## Q6. Fork System Call

## #include<sys/types.h>

## #include<stdio.h>

## #include<unistd.h>

## int main(){

## pid\_t pid;

## pid = fork();

## if(pid<0){

## printf("error");

## //fprintf(stderr,"fork failed");

## return 1;

## }

## else if(pid==0){

## printf("child ");

## //execlp("/bin/ls","ls",NULL);

## }

## else{

## wait(NULL);

## printf("parent ");

## }

## return 0;

## 

## Q7. Race Condition using Semaphore.

## #include<stdio.h>

## #include<semaphore.h>

## #include<unistd.h>

## #include<pthread.h>

## void \*fun1(void \*argv);

## void \*fun2(void \*argv);

## int shared = 5;

## sem\_t s;

## int main(){

## pthread\_t thread1;

## pthread\_t thread2;

## sem\_init(&s,0,1);

## pthread\_create(&thread1,NULL,fun1,NULL);

## pthread\_create(&thread2,NULL,fun2,NULL);

## pthread\_join(thread1,NULL);

## pthread\_join(thread2,NULL);

## printf("shared = %d\n",shared);

## }

## void \*fun1(void \*argv){

## printf("Inside first function");

## sem\_wait(&s);

## int x=shared;

## x++;

## shared =x;

## printf("shread variable in function 1 = %d\n",shared);

## sem\_post(&s);

## }

## void \*fun2(void \*argv){

## printf("Inside second function");

## sem\_wait(&s);

## int y=shared;

## y--;

## //sleep(1);

## shared=y;

## printf("shared variable in function 2= %d\n",shared);

## sem\_post(&s);

}pg. 8

## 

## Q8. Producer Consumer Problem.

## #include<stdio.h>

## #include<stdlib.h>

## #include<sys/types.h>

## #include<unistd.h>

## #include<pthread.h>

## #include<stdatomic.h>

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

## \_Atomic int count= 0;

## int s = 1;

## int wait()

## {

## s =0;

## }

## int signal()

## {

## s =1;

## }

## int in = 0;

## int out = 0;

## int buffer[8] = {0,0,0,0,0,0,0,0};

## void\* producer(void \*p3)

## {

## wait();

## int itemp;

## srand(time(0));

## int j = 1;

## int k = 0;

## while(j<9)

## {

## wait();

## itemp= rand()%100;

## while(count==8)

## {

## printf("\n BUFFER FULL \n");

## break;

## }

## buffer[in] = itemp;

## in = (in+1);

## j=j+1;

## //sleep(1);

## count = count+1;

## signal();

## printf("\n produces %d , count = %d \n",itemp,count);

## }

## }

## void\* consumer(void \*p4)

## {

## int itemc;

## int i=1;

## if(s ==0)

## {printf("\n CANNOT CONSUME ITEM \n"); pthread\_exit(NULL);}

## else{

## wait();

## while(i<5)

## {

## while(count ==0)

## { printf("\n EMPTY \n");

## //pthread\_exit(NULL);

## break;

## }

## if(buffer[out] ==0)

## { out++;

## break;}

## itemc = buffer[out];

## buffer[out] = 0;

## out = (out+1);

## i = i+1;

## //sleep(1);

## count = count-1;

## signal();

## printf("\n consumes %d , count = %d \n",itemc,count);

## }

## }

## }

## int main()

## {

## pthread\_t id[2];

## void \*p1 = NULL;

## void \*p2 = NULL;

## pthread\_create(&id[0],NULL,producer,&p1);

## pthread\_create(&id[1],NULL,consumer,&p2);

## int \*pp1;

## int \*pp2;

## pthread\_join(id[0],(void\*\*)&pp1);

## pthread\_join(id[1],(void\*\*)&pp2);

## printf("\n count = %d \n",count);

## int z = 0;

## for(z=0;z<8;z++)

## {

## printf("\n %d \n ",buffer[z]);

## }

## }

## 

## 

## Q9. CPU Scheduling

## Round Robin

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

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

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

## rt[count]=bt[count];

## }

## printf("Enter the 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;

## }

## 

## Q9. CPU Scheduling

## First Come First Serve

## #include<stdio.h>

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

## int bt[], int wt[])

## {

## // waiting time for first process is 0

## wt[0] = 0;

## // calculating waiting time

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

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

## }

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

## int bt[], int wt[], int tat[])

## {

## // calculating turnaround time by adding

## // bt[i] + wt[i]

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

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

## }

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

## {

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

## 

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

## 

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

## 

## printf("Processes Burst 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];

## printf(" %d ",(i+1));

## printf(" %d ", bt[i] );

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

## printf(" %d\n",tat[i] );

## }

## int s=(float)total\_wt / (float)n;

## int t=(float)total\_tat / (float)n;

## printf("Average waiting time = %d",s);

## printf("\n");

## printf("Average turn around time = %d ",t);

## }

## int main()

## {

## //process id's

## int processes[] = { 1, 2, 3};

## int n = sizeof processes / sizeof processes[0];

## //Burst time of all processes

## int burst\_time[] = {10, 5, 8};

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

## return 0;

## }

## 

## Q9. CPU Scheduling

## Shortest Job First

## #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("\nProcess\t Burst Time \tWaiting Time\tTurnaround 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\t%d",p[i],bt[i],wt[i],tat[i]);

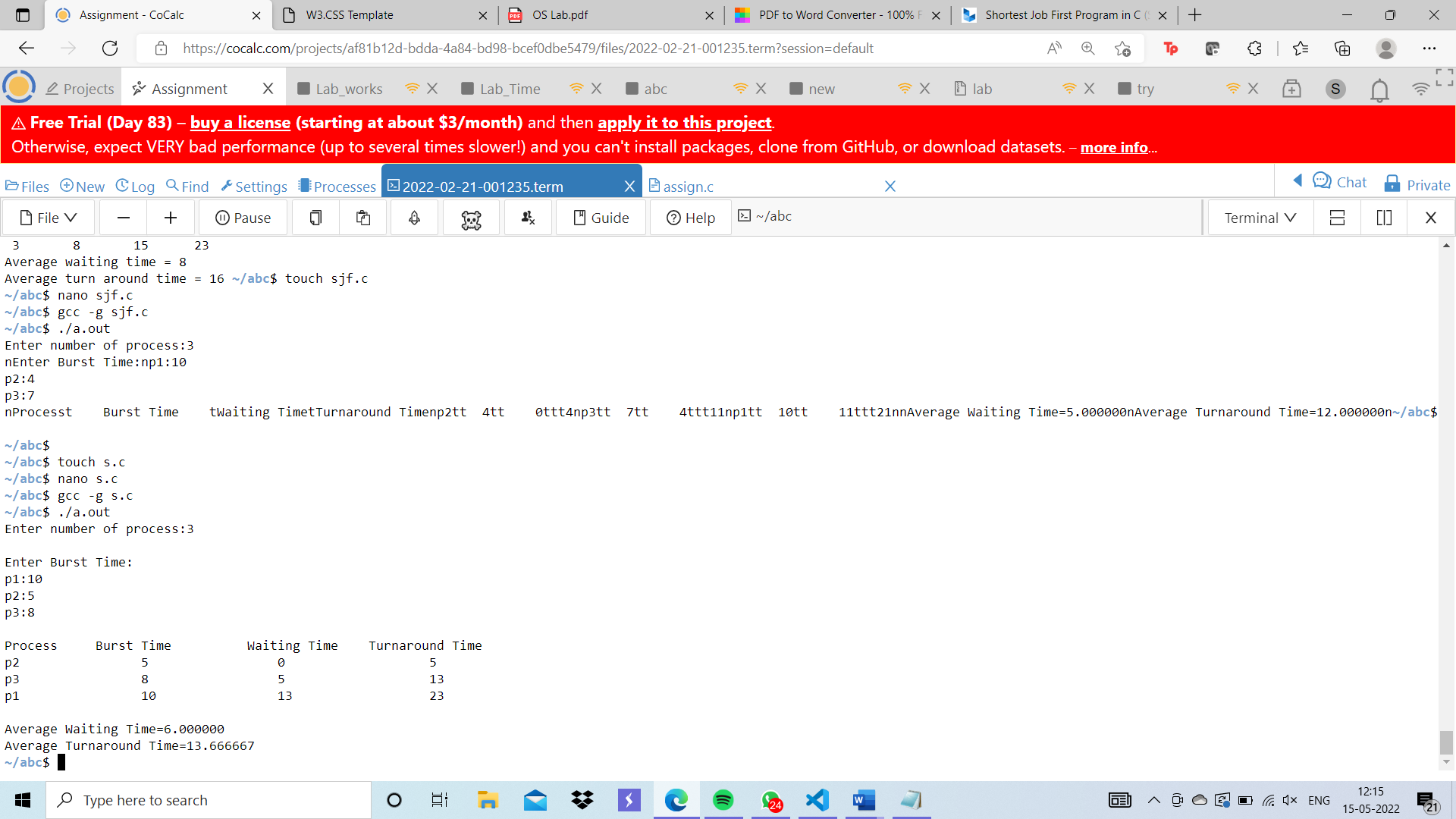
## }

## avg\_tat=(float)total/n;

## printf("\n\nAverage Waiting Time=%f",avg\_wt);

## printf("\nAverage Turnaround Time=%f\n",avg\_tat);

## }



## Q10. Reader Writer Problem

## #include<stdio.h>

## #include<stdlib.h>

## #include<unistd.h>

## #include<pthread.h>

## #include<sys/types.h>

## #include<stdatomic.h>

## \_Atomic int n;

## \_Atomic int m;

## int rw\_mutex = 1;

## int mutex = 1;

## \_Atomic int read\_count = 0;

## int wait(int a)

## {

## a = 0;

## return a;

## }

## int signal(int b)

## {

## b = 1;

## return b;

## }

## void\* writer()

## {

## int i = 1;

## rw\_mutex = wait(rw\_mutex);

## printf("\n WRITER IN CS \n");

## i++;

## sleep(1);

## rw\_mutex = signal(rw\_mutex);

## }

## void \* reader()

## { //sleep(5);

## int j =1;

## if(rw\_mutex ==0)

## {

## printf("\n READER CANNOT ENTER IN CS \n");

## pthread\_exit(NULL);

## }

## else{

## while(j<=m){

## mutex = wait(mutex);

## read\_count++;

## if(read\_count ==1)

## {

## rw\_mutex = wait(rw\_mutex);

## }

## mutex = signal(mutex);

## printf("\n READER IN CS \n");

## mutex = wait(mutex);

## read\_count--;

## j++;

## if(read\_count ==0)

## {

## rw\_mutex = signal(rw\_mutex);

## }

## mutex = signal(mutex);

## }

## }

## }

## int main()

## {

## //printf("\n enter n for writer");

## //scanf("%d",&n);

## printf("\n enter n for reader \n");

## scanf("%d",&m);

## pthread\_t id[2];

## pthread\_create(&id[0],NULL,writer,NULL);

## pthread\_create(&id[1],NULL,reader,NULL);

## int \*pp1;

## int \*pp2;

## pthread\_join(id[0],(void\*\*)&pp1);

## pthread\_join(id[1],(void\*\*)&pp2);

## }

## 

## Q10. Deadlock Condition (Banker Algorithm)

Banker Algorithm

#include <stdio.h>

int main()

{

int n, r, i, j, k;

n = 5;

r = 3;

int alloc[5][3] = { { 0, 1, 0},

{ 2, 0, 0 },

{ 3, 0, 2 },

{ 2, 1, 1 },

{ 0, 0, 2 } };

int max[5][3] = { { 7, 5, 3 },

{ 3, 2, 2 },

{ 9, 0, 2 },

{ 4, 2, 2 },

{ 5, 3, 3 } };

int avail[3] = { 3, 3, 2 };

int f[n], ans[n], ind = 0;

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

f[k] = 0;

}

int need[n][r];

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

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

need[i][j] = max[i][j] - alloc[i][j];

}

int y = 0;

for (k = 0; k < 5; k++) {

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

if (f[i] == 0) {

int flag = 0;

for (j = 0; j < r; j++) {

if (need[i][j] > avail[j]){

flag = 1;

break;

}

}

if (flag == 0) {

ans[ind++] = i;

for (y = 0; y < r; y++)

avail[y] += alloc[i][y];

f[i] = 1;

}

}

}

}

printf("Th SAFE Sequence is as follows\n");

for (i = 0; i < n - 1; i++)

printf(" P%d ->", ans[i]);

printf(" P%d", ans[n - 1]);

return (0);

