**OS Lab End Sem Examination**

**Codes**

**Name: Vraj Parikh**

**Enrolment No: AU1940185**

**Ques 1:**

Print the following Pattern

A 1 a B 2 b C 3 c ... Y 25 y Z 26 z

Using any one of the following concepts

a. Multiprocesses (Hint: using 3 child processes)

b. Multithreads (Hint: using 3 Threads)

(PS: Process Synchronization or Thread Synchronization is key thing for

pattern printing)

// Vraj Parikh

// AU1940185

#include <iostream>

#include <thread>

using namespace std;

void foo(int Z)

{

    for (int i = 0; i < Z; i++) {

       printf("%c" , Alpha);

       Alpha++;

    }

}

class thread\_obj {

public:

    void operator()(int x)

    {

        for (int i = 0; i < x; i++)

    }

};

int Num = 0

int Alpha = 65

int lowerAlpha = 97

int main()

{

    printf("%c" , lowerAlpha);

       lowerAlpha++;

    thread th1(foo, 3);

    thread th2(thread\_obj(), 3);

    auto f = [](int x) {

        for (int i = 0; i < x; i++)

        Num ++

        printf("%d" , Num);

    };

    thread th3(f, 3);

    th1.join();

    th2.join();

    th3.join();

    return 0;

}

**Ques 2(a)**

Describe the RoundRobin (RR) and Modified RoundRobin (MRR)

Algorithm. Also mention the difference between the results of both

the algorithms and implement them both in C.

**Round Robin Scheduling Algorithm**

// Vraj Parikh

// AU1940185

#include<stdio.h>

#include<conio.h>

void main()

{

    // initlialize the variable name

    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; // Assign the number of process to variable y

// Use for loop to enter the details of the process like Arrival time and the Burst Time

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

}

// Accept the Time qunat

printf("Enter the Time Quantum for the process: \t");

scanf("%d", &quant);

// Display the process No, burst time, Turn Around Time and the waiting time

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;

    }

}

// represents the average waiting time and 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);

getch();

}

**Modified Round Robin Algorithm**

// Vraj Parikh

// AU1940185

#include <stdio.h>

 #include <conio.h>

 #include <stdio.h>

 #include <conio.h>

 #include<math.h>

 #include<string.h>

 int wt[100],bt[100],at[100],tat[100],n,p[100];

 float awt[5],atat[5];

 int temp1,temp2,temp3,sqt,avg;

 void input()

 {

        printf("Enter Number of processes:");

        scanf("%d",&n);

        int i;

        for(i=0;i<n;i++)

            p[i]=i+1;

        for(i=0;i<n;i++)

        {

          printf("Enter Burst Time of process %d:",i+1);

          scanf("%d",&bt[i]);

          printf("Enter Arrival Time of process %d:",i+1);

          scanf("%d",&at[i]);

        }

        for(i=0;i<5;i++)

        {

            awt[i]=0.0;

            atat[i]=0.0;

        }

 }

 void changeArrival(){

    int a=at[0];

    int i;

    for(i=0;i<n;i++){

        if(at[i]<a)

            a=at[i];

     }

     if(a!=0){

        for(i=0;i<n;i++)

            at[i]=at[i]-a;

     }

 }

 void fcfs(){

    wt[0]=0;

    atat[0]=tat[0]=bt[0];

    int btt=bt[0];

    int i;

    for(i=1;i<n;i++){

      wt[i]=btt-at[i];

      btt+=bt[i];

      awt[0]+=wt[i];

      tat[i]= wt[i]+bt[i];

      atat[0]+=tat[i];

    }

    atat[0]/=n;

    awt[0]/=n;

    printf("SR.\tA.T.\tB.T.\tW.T.\tT.A.T.\n");

    for(i=0;i<n;i++)

    {

    printf("%3d\t%3d\t%3d\t%3d\t%4d\n",i+1,at[i],bt[i],wt[i],tat[i]);

    }

 }

 void innovative()

 {

    int bt1[n],i,j,temp,qt;

    int b[n];

    float twt,ttat;

    for(i=0;i<n;i++)

        bt1[i]=bt[i];

    for(i=0;i<n;i++)

        b[i]=bt[i];

    int num=n;

    int time=0;

    int max;

    int sum,t,a,ap;

    ap=0;

        //sorting in ascending order

        for (i = 0; i < n; i++)

        {

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

            {

                if (bt[j] > bt[j + 1])

                {

                      temp1 = bt[j];

                      temp2 = p[j];

                      temp3 = at[j];

                      bt[j] = bt[j + 1];

                      p[j] = p[j + 1];

                      at[j] = at[j + 1];

                      bt[j + 1] = temp1;

                      p[j + 1] = temp2;

                      at[j + 1] = temp3;

                 }

              }

          }

        max=bt[n-1];

        sum=0;

        for(i=0;i<n;i++)

        {

            sum=sum+bt[i];

        }

        avg=sum/n;

        //printf("\n %d  %d \n",max,avg);

        qt=(avg+max)/2;

        printf("\nDynamic Quantum time calculated is : %d\n",qt);

        while(num>0){

        a=0;

        max=0;

        sum=0;

        t=0;

        for(i=0;i<n;i++){

            if(at[i]<=time && b[i]!=0)

            {

                if(b[i]<qt)

                {

                    t+=b[i];

                    b[i]=0;

                }

                else

                {

                    t+=qt;

                    b[i]-=qt;

                }

                if(b[i]<qt && b[i]!=0)

                {

                    t+=b[i];

                    b[i]=0;

                }

                if(b[i]==0){

                    wt[i]=(time+t)-bt1[i];

                    tat[i]=time+t;

                    num--;

                }

            }

        }

        time+=t;

        }

        printf("Processes\tWaitingtime\tTurnAroundTime\n");

        for(j=1;j<=n;j++)

            {

                for(i=0;i<n;i++)

                {

                    if(j==p[i])

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

                }

            }

        twt=0;

        ttat=0;

        for(i=0;i<n;i++)

            {twt=twt+wt[i];}

        awt[4]=twt/n;

        for(i=0;i<n;i++)

            {ttat=ttat+tat[i];}

        atat[4]=(ttat/n);

 }

 void rr()

 {

    int i, total = 0, x, counter = 0, time\_quantum;

    int wait\_time = 0, turnaround\_time = 0, temp[100];

    x=n;

    for(i = 0; i < n; i++)

    {

        temp[i] = bt[i];

    }

    printf("\nEnter Time Quantum:\t");

    scanf("%d", &time\_quantum);

    printf("\nProcess ID\t\tBurst Time\t Turnaround Time\t Waiting Time\n");

    for(total = 0, i = 0; x != 0;)

    {

            if(temp[i] <= time\_quantum && temp[i] > 0)

            {

                total = total + temp[i];

                temp[i] = 0;

                counter = 1;

            }

            else if(temp[i] > 0)

            {

                temp[i] = temp[i] - time\_quantum;

                total = total + time\_quantum;

            }

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

            {

                x--;

                printf("Process[%d]\t\t%d\t\t %d\t\t\t %d\n", i + 1, bt[i], total - at[i], total - at[i] - bt[i]);

                wait\_time = wait\_time + total - at[i] - bt[i];

                turnaround\_time = turnaround\_time + total - at[i];

                counter = 0;

            }

            if(i == n - 1)

            {

            i = 0;

        }

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

            {

            i++;

        }

            else

            {

            i = 0;

        }

    }

    awt[2] = wait\_time \* 1.0 / n;

    atat[2] = turnaround\_time \* 1.0 / n;

 }

 void srtf()

 {

    int i,j,x[10],b[10],count=0,time,smallest;

    int avg1=0,tt1=0,end1;

    int bt2[100];

    for(i=0;i<n;i++)

        bt2[i]=bt[i];

 for(i=0;i<n;i++)

 {

    x[i]=bt2[i];

 }

 b[9]=9999;

 for(time=0;count!=n;time++)

 {

   smallest=9;

  for(i=0;i<n;i++)

  {

   if(at[i]<=time && bt2[i]<b[smallest] && bt2[i]>0 )

   smallest=i;

  }

  bt2[smallest]--;

  if(bt2[smallest]==0)

  {

   count++;

   end1=time+1;

   avg1=avg1+end1-at[smallest]-x[smallest];

   tt1= tt1+end1-at[smallest];

  }

 }

 awt[3]=avg1/n;

 atat[3]=tt1/n;

 printf("Processes\tWaitingtime\tTurnAroundTime\n");

        for(j=1;j<=n;j++)

            {

                for(i=0;i<n;i++)

                {

                    if(j==p[i])

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

                }

            }

}

 void display(int c)

 {

    int i;

    printf("Average Waiting Time: %f\nAverage Turn Around Time:%f",awt[c-2],atat[c-2]);

 }

void sjf()

{

    float wavg=0,tavg=0,tsum=0,wsum=0;

    int i,j,temp,sum=0,ta=0;

     for(i=0;i<n;i++)

    {

        for(j=0;j<n;j++)

        {

        if(at[i]<at[j])

        {

        temp=p[j];

        p[j]=p[i];

        p[i]=temp;

        temp=at[j];

        at[j]=at[i];

        at[i]=temp;

        temp=bt[j];

        bt[j]=bt[i];

        bt[i]=temp;

        }

    }

}

 int btime=0,min,k=1;

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

{

btime=btime+bt[j];

min=bt[k];

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

{

if (btime>=at[i] && bt[i]<min)

{

temp=p[k];

p[k]=p[i];

p[i]=temp;

temp=at[k];

at[k]=at[i];

at[i]=temp;

temp=bt[k];

bt[k]=bt[i];

bt[i]=temp;

}

}

k++;

}

wt[0]=0;

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

{

sum=sum+bt[i-1];

wt[i]=sum-at[i];

wsum=wsum+wt[i];

}

awt[1]=(wsum/n);

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

{

ta=ta+bt[i];

tat[i]=ta-at[i];

tsum=tsum+tat[i];

}

atat[1]=(tsum/n);

printf("SR.\tA.T.\tB.T.\tW.T.\tT.A.T.\n");

    for(i=0;i<n;i++)

    {

    printf("%3d\t%3d\t%3d\t%3d\t%4d\n",i+1,at[i],bt[i],wt[i],tat[i]);

    }

 }

 int main(){

       input();

        changeArrival();

        //rr();

        innovative();

 }