ENDTERMPROJECT

Design of Operating System (CSE4049)

Submitted by

Name	**: SWAPNESWAR SATAPATHY**
Reg.No.	**: 2141019225**
Branch	**: Computer Science and Engineering**
Semester	**: 5th**
Section	**: K**
Session	**: 2023-2024**
Admission Batch	**: 2021**

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

FACULTY OF ENGINEERING & TECHNOLOGY (ITER)

SIKSHA 'O' ANUSANDHAN DEEMED TO BE UNIVERSITY

BHUBANESWAR, ODISHA – 751030

Project 1:

CODE:

```
import java.util.Scanner;

publicclassOBJ1{

public static void main(String[] args) {
    Scanner sc=new Scanner(System.in);
    boolean fcfsChosen = false;

        while (true) {
            System.out.println();
            System.out.println("Choose the scheduling algorithm:");
            System.out.println("1. First-Come First-Served(FCFS)");
            System.out.println("2. Round Robin (RR)");
            System.out.println("3. Terminate Program");
            intchoice=sc.nextInt();

            switch (choice) {
            case1:
            if (!fcfsChosen) {
                fcfsAlgorithm(sc);
                fcfsChosen=true;
                    }else{
                        rrAlgorithm(sc);
                    }
                    break;
                case 2:
                    rrAlgorithm(sc);
                    break;
                case3:
                    System.out.println("Terminating the program...");
                    System.exit(0);
                    break;
                default:
                    System.out.println("Invalid choice!");
            }
        }
    }

    private static void fcfsAlgorithm(Scanner sc) {
        System.out.print("Enter the number of processes:");
        int n = sc.nextInt();

        int burstTimes[] = new int[n];

        int arrivalTimes[]=new int[n];
```

```
    System.out.println("\n Enter the Arrival Time and Burst Time for each process.");
    for (int i = 0; i < n; i++) {
      System.out.print("\n For Process" +(i+1)+ "-Arrival Time:");
      Arrival Times[i] = sc.nextInt();
      System.out.print("For Process"+(i+1)+"-Burst Time:");
      burstTimes[i] = sc.nextInt(); }

      calculateAndDisplayTimes(n,
    brustTimes,arrivalTimes); }

private static void rrAlgorithm(Scanner sc) {
    System.out.print("Enter the number of processes:");
    int n = sc.nextInt();
    int processes[] = new int[n];
    int burstTimes[] = new int[n];
    int arrivalTimes[] = newint[n];
    for(inti=0;i<n;i++)
    {
      System.out.print("Enter Arrival Time for Process"+(i+1)+":");
      arrivalTimes[i] = sc.nextInt();
      System.out.print("Enter Burst Time for Process"+(i+1)+":");
      burstTimes[i] = sc.nextInt();
      processes[i]=i+1;
    }

    System.out.print("Enter the time quantum:");
    int quantum = sc.nextInt();

    findAvgTime(processes,n,burstTimes,quantum,arrivalTimes);
}

Private static void calculate And Display Times(int n,int[] burstTimes,int[] arrivalTimes)
    {
int wt[] = new int[n];
int rt[] = new int[n];
int ct[] = new int[n];
int at[]=new int[n];

    wt[0]=0;
    ct[0]=burstTimes[0];
    tat[0]=ct[0] - arrivalTimes[0];

    //Corrected turn around time calculation

    for(inti=1; i<n; i++) {
    wt[i] = ct[i-1] - arrivalTimes[i];
    if (wt[i] < 0) {
    wt[i]=0;
      }
      rt[i]=wt[i];
      ct[i] = ct[i-1] + burstTimes[i];
      tat[i] = ct[i] - arrivalTimes[i];
    }
```

```
    System.out.println("\n Processes||Burst Time||Arrival Time||Waiting Time||Response Time|| Turn
    around Time || Completion Time ");


Float awt=0;
float art = 0;
float att = 0;

    for(int i=0 ; i<n ; i++) {
       System.out.println((i + 1) + "\t ||\t" + burstTimes[i] + "\t||\t"+arrivalTimes[i]+"\t||\t"+wt[i]+"\t||\t"
             +rt[i]+"\t||\t"+tat[i]+"\t||\t"+ct[i]);
       Awt += wt[i];
       art +=  rt[i];
       att  +=  tat[i];
    }

    Awt = awt/n;
    art = art / n;
    att = att / n;

    System.out.println("\n Average waiting time="+awt);
    System.out.println("Average response time = " + art);
    System.out.println("Average turn around time="+att);
  }

  Private static void findAvgTime(int processes[],int n,int burstTimes[],int quantum,int
    arrivalTimes[]) {

     Int wt[] = new int[n],tat[] = new int[n],ct[] = new int[n],rt[] = new int[n]; double total_wt = 0,
    total_tat = 0, total_rt = 0;

    findWaitingTime(processes,n,burstTimes,wt,quantum,arrivalTimes,ct,rt);
    findTurnAroundTime(processes, n, burstTimes, wt, tat, ct, arrivalTimes);

    System.out.println("Processes"+"Bursttime"+"Waitingtime"+"Turnaroundtime"+" Response
time");
    for(inti=0 ; i<n ; i++) {
    total_wt  += wt[i];
    total_tat += tat[i];
    total_rt += rt[i];
    System.out.println(""+processes[i]+"\t\t"+burstTimes[i]+"\t"+wt[i]+"\t\t"+tat[i]+"\t\t " + rt[i]);
 }

    System.out.println("Average waiting time = " + total_wt / n);
    System.out.println("Average turn around time="+total_tat/n);
    System.out.println("Average response time = " + total_rt / n);

    // Compare the efficiency of algorithms based on average waiting time.

    compareAlgorithmsEfficiency(total_wt/n,calculateFCFSAvgWaitingTime(burstTimes,
    arrivalTimes));
  }
```

```
Private static void findWaitingTime(int processes[],int n,int burstTimes[],int wt[],int quantum,int
arrivalTimes[], int ct[], int rt[]) {
intrem_bt[] = newint[n];

for (int i = 0 ; i < n; i++)
rem_bt[i] = burstTimes[i];
int t = 0;
      Boolean visited[]=new boolean[n];
      while (true) {
      boolean done = true;
      for(int i = 0 ; i<n ; i++) { if(rem_bt[i] > 0 &&
         arrivalTimes[i] <= t) {
        done = false;
      if(!visited[i]) {
                  rt[i] = t-arrivalTimes[i];
                  visited[i] = true;
               }
               if(rem_bt[i] > quantum) {
               t += quantum;
               rem_bt[i] -= quantum;
               }  else  {
                  t+=rem_bt[i];
                  wt[i] = t-burstTimes[i]-
                  arrivalTimes[i];
                  rem_bt[i] = 0;
                  ct[i]=t;
               }
            }
         }
         if(done)
         break;
      }
   }

   Private static void findTurnAroundTime(int processes[],int n,int burstTimes[],int wt[],int tat[],int ct[],
   int arrivalTimes[]) {
   for(inti=0 ; i<n ; i++)
   tat[i] = ct[i] - arrivalTimes[i];
   }

   Private static double calculate FCFSAvgWaitingTime(int[] burstTimes,int[] arrivalTimes)
   {
   int n = burstTimes.length;
   int wt[] = new int[n];
   int ct[] = new int[n];
      int prevCT = 0;

      for(inti=0 ; i<n ; i++)
      {
      wt[i] = prevCT - arrivalTimes[i];
      if (wt[i] < 0)
       {
```

```
        wt[i] = 0;
            }
            ct[i] = prevCT + burstTimes[i];
            prevCT = ct[i];
        }
    double total_wt=0;

        for(int I = 0 ; I < n ; i++) {
            total_wt + = wt[i];
        }

        Return total_wt /n;
    }

    Private static void compare AlgorithmsEfficiency(double avgWaitingTime RR,double
avgWaitingTimeFCFS)
 {
if(avg WaitingTimeRR<avg WaitingTimeFCFS)
{
            System.out.println("Round Robin(RR) algorithm results in the minimum average waiting time= "
+ avgWaitingTimeRR);
        } elseif(avg WaitingTimeRR>avg WaitingTimeFCFS)
        {
         System.out.println("FCFS algorithm results in the minimum average waiting time="+ avg
        WaitingTimeFCFS);
        }else
        {
            System.out.println("Both algorithms have the same average waiting time.");
        }
    }
}
```

OUTPUT:

TESTCASE 1:

```
1. First-Come, First-Served (FCFS)
2. Round Robin (RR)
3. Terminate Program
1
Enter the number of processes: 5

Enter the Burst Time for each process.

For Process 1: 10

For Process 2: 1

For Process 3: 2

For Process 4: 1

For Process 5: 5

Enter the arrival time for each process.

For Process 1: 0

For Process 2: 1

For Process 3: 2

For Process 4: 3

For Process 5: 6

Processes || Arrival Time || Burst Time || Waiting Time || Turnaround Time || Completion Time
1         ||     0        ||     10     ||      0       ||       10        ||       10
2         ||     1        ||     1      ||      9       ||       10        ||       11
3         ||     2        ||     2      ||      9       ||       11        ||       13
4         ||     3        ||     1      ||      10      ||       11        ||       14
5         ||     6        ||     5      ||      8       ||       13        ||       19

Average waiting time = 7.2

Average response time = 7.2

Average turnaround time = 11.0
```

Test Case 2:

```
Choose the scheduling algorithm:
1. First-Come, First-Served (FCFS)
2. Round Robin (RR)
3. Terminate Program
2
Enter the number of processes: 5
Enter Arrival Time for Process 1: 0
Enter Burst Time for Process 1: 10
Enter Arrival Time for Process 2: 1
Enter Burst Time for Process 2: 1
Enter Arrival Time for Process 3: 2
Enter Burst Time for Process 3: 2
Enter Arrival Time for Process 4: 3
Enter Burst Time for Process 4: 1
Enter Arrival Time for Process 5: 6
Enter Burst Time for Process 5: 5
Enter the time quantum: 4
Processes  Burst time  Waiting time  Turnaround time  Response time
 1             10        9               19               0
 2             1         3               4                3
 3             2         3               5                3
 4             1         4               5                4
 5             5         6               11               2
Average waiting time = 5.0
Average turnaround time = 8.8
Average response time = 2.4
Round Robin (RR) algorithm results in the minimum average waiting time = 5.0
```

Project 2:

CODE:

```
Public class OBJ2 {

    Public static void main(String[]  args)
    {
    int n, m, i, j, k;
    n =5;
    //Number of processes
     m=4;
    //Number of resources
       Int alloc[][] = {{0,0,1,2},//P1//Allocation Matrix {2,0,0,0},// P2 {0,0,3,4},// P3 {2,3,5,4},// P4
       {0,3,3,2}};// P5



       Int max[][] = {{0,0,1,2},//P1//MAX Matrix {2,7,5,0},// P2 {6,6,5,6},// P3 {4,3,5,6},//
       {0,6,5,2}};// P5

       Int avail[] = {6,7,12,12};
       //Available Resources int
       f[] = new int[n];

       int ans[] = new int[n];

       int ind = 0;

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

       f[k]=0;

       }

       Int need[][] = int[n][m];

       for (i = 0; i < n; i++) {
           for(j=0; j<m;j++){
             need[i][j] = max[i][j] - alloc[i][j];
           }
       }

       //Print the content of the Need matrix

       System.out.println("Need  Matrix:");
       for (i = 0; i < n; i++) {
           for (j = 0; j < m; j++) {
               System.out.print(need[i][j]+"");
           }
           System.out.println();
       }
```

```
        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<m;j++){
                    if(need[i][j]>avail
                    [j]) {
                            flag = 1;
                            break;
                        }
                    }

                    if (flag == 0) {
                        ans[ind++] = i;

                        for(y=0; y<m; y++){
                          avail[y] += alloc[i][y];
                        }
                        f[i]=1;
                    }
                }
            }
        }

        Int flag=1;
        for(i=0;i<n;i++)
        {
         if (f[i] == 0) {
         flag=0;
                System.out.println("The following system is not safe");
            break;
                }
        }

        if(flag==1) {
            System.out.println("Following is the safe sequence");
            for (i = 0; i < n - 1; i++) {
                System.out.print("P"+ans[i]+"->");
            }
            System.out.print("P"+ans[n-1]);
        }
    }
}
```

TEST CASES:

a) Find the content of the need matrix.

```
Need Matrix:
0 0 0 0
0 7 5 0
6 6 2 2
2 0 0 2
0 3 2 0
```

b) Is the system in a safe state? If so, give a safe sequence of the process.

```
Following is the SAFE Sequence
 P0 -> P1 -> P2 -> P3 -> P4
```

c) If P3 will request for 1 more instances of type R2, Can the request be granted immediately or not?

CODE:

```
Public class OBJ2 {

    Public static void main(String[] args) {
        int n, m, i, j, k;
        n =5;
        // Number of processes
        m=4;
        //Number of resources
        Int alloc[][] = {{0,0,1,2},//P1//Allocation Matrix {2,0,0,0},// P2 {0,0,3,4},// P3 {2,3,5,4},// P4
        {0,3,3,2}};// P5

        Int max[][]={{0,0,1,2},//P1//MAX Matrix {2,7,5,0},// P2 {6,6,5,6},// P3 {4,3,5,6},// P4
        {0,6,5,2}};// P5

        int avail[] = {6,7,12,12};
        //Available Resources
        int f[] = new int[n];
        int ans[] = new int[n];
        int ind = 0;

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

```
f[k] = 0;

}

Int need[][] = int[n][m];

for (i = 0; i < n; i++) {
 for(j=0; j<m;j++) {
need[i][j] = max[i][j]-alloc[i][j];
}
}

//Check If P3 requests 1 more instance of type R2



int request[] = {0, 1, 0, 0};
// P3's request
Int processIndex=2;
//Index of P3
//Check if the request can be granted immediately
for (j = 0; j < m; j++)
{
if (request[j] > need[processIndex][j] || request[j] > avail[j]) {
               System.out.println("There quest cannot be granted immediately.");
              return;
    }
}

//Grant the request
for(j=0 ; j<m ; j++) {
avail[j] -= request[j];
alloc[processIndex][j] += request[j];
need[processIndex][j] -= request[j]; }

//Safety check after granting the request
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<m;j++) {
    if(need[i][j]>avail[j]) {
    flag = 1;
    break;
               }
            }

            if (flag == 0) {
                ans[ind++]=i;
                for(y=0; y<m; y++) {
                avail[y] += alloc[i][y];
```

```
                    }
                    f[i]=1;
                }
            }
        }
    }

    Int flag=1;
    for(i=0;i<n;i++){
    if (f[i] == 0) {
    flag=0;
                System.out.println("The request cannot be granted immediately.");
                return;
        }
    }


    if(flag==1)
    {
        System.out.println(" The request can be granted
        immediately.");
        System.out.print("Following is the safe sequence \n");
        for (i = 0; i < n - 1; i++)
        System.out.print("P"+ans[i]+"->");
        System.out.println("P"+ans[n-1]);
    }
  }
}
```

OUTPUT:

```
The request can be granted immediately.
Following is the SAFE Sequence
 P0 -> P2 -> P3 -> P4 -> P1
```