**LOVELY PROFESSIONAL UNIVERSITY**

**Academic Task-3**

**(Operating System)**

School of Computer Science and Engineering( Faculty of Technology And Sciences )

**Name of the faculty member:**  Neha Sharma

**Course Code: CSE 316 Course Title:** Operating System

**Term:** 18192

Date of Allotment: 11/02/2018 Date of Submission: 17/04/2018

**Name:** Kartik Tyagi

**Reg No:** 11702672

**Roll No: 01**

**Section:** K17EB G1

**Email Address:** kartik9756@gmail.com

**GitHub Link:** <https://github.com/kartik9756/OperatingSystemAssignment>

**Problem** **1**:

Considering 4 processes with the arrival time and the burst time requirement of the processes the scheduler schedules the processes by interrupting the processor after every 3 units of time and does consider the completion of the process in this iteration. The schedulers then checks for the number of processes waiting for the processor and allots the processor to the process but interrupting the processor after every 6 units of time and considers the completion of the process in this iteration. The scheduler after the second iteration checks for the number of processes waiting for the processor and now provides the processor to the process with the least time requirement to go in the terminated state.

The inputs for the number of requirements, arrival time and burst time should be provided by the user.

Consider the following units for reference.

**Process Arrival time Burst time**

P1 0 18

P2 2 23

P3 4 13

P4 13 10

Develop a scheduler which submits the processes to the processor in the above defined scenario, and compute the scheduler performance by providing the waiting time for process, turnaround time for process and average waiting time and turnaround time.

**Description:**

Above first Question is about to perform and implement code for Process Scheduling. In which Every process will be Iterated in its first iteration according to Round Robin with Time Quantum =3 unit, and 6 unit when it is iterated Second Time. When the process is traversed Third time, the processor will be assigned to the process which have less time remaining i.e. Shortest Remaining Time First(SRTF). We need to calculate Average Waiting time and Average Turn Around Time.

**Alogithm:**

1. For First and Second Iteration.
2. For first iteration is time\_quantum=3 and for second Iteration time\_quantum=6.
3. For(i=0 to no of processes)

If(arrival\_time<time\_duration)

If(burst\_time>time\_quantam1){

Decrease burst\_time by time\_quantum1;

}

Else{

Time\_duration+=burst\_time;

Compilation\_time=time\_duration;

Turnaroundtime=compilation\_time-arrival\_time;

Burst\_time-=time\_quantum;

Add the process to the array ar[]; //as to check whether this process is completed later.

}

1. Sort remaining processes according to remaining burst\_time.
2. Check for the process which is already completed.
3. For(remaining process) {

Time\_duration+=burst\_time;

Compilation\_time=time\_duration;

Turnaroundtime=compilation\_time-arival\_time;

}

1. Print all the process. And Average TurnAroundTime and Average Waiting Time.

**Complexity:**

Complexity=O(n2)

**Code Snippet:**

#include<stdio.h>

int main(){

int i,m;

printf("\nEnter the Total number of processes: ");

scanf("%d",&m);

int arrival[m],burst[m],burst1[m], waiting[m], tarrival[m];

int tq1=3;

int tq2=6;

float avgT=0;

float avgW=0;

int time=0;

int x=0;

int ct[m], ar[m];

printf("\nEnter Arrival Time and the Burst Time of Processes:");

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

{ scanf("%d%d",&arrival[i],&burst[i]);

burst1[i]=burst[i];

}

printf("\nFirst Iterarrivalion(TQ=3)\n"); //first Iteration

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

if(arrival[i]<=time) {

if(burst[i]>tq1){

time=time+tq1;

burst[i]=burst[i]-tq1;

printf("\nP[%d] : %d",i+1,burst[i]);

}

else {

time+=burst[i];

ct[i]=time;

tarrival[i]=ct[i]-arrival[i];

ar[x]=i;

x++;

burst[i]-=tq1;

printf("\nP[%d] : %d",i+1,burst[i]);

}

}

}

printf("\n\nAFTER ITERarrivalION 2 (TQ=6)"); //Second Iteration

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

if(arrival[i]<=time) {

if(burst[i]>tq1){

time+=tq2;

burst[i]=burst[i]-tq2;

printf("\nP[%d] : %d",i+1,burst[i]);

}

else {

time+=burst[i];

ct[i]=time;

tarrival[i]=ct[i]-arrival[i];

ar[x]=i;

x++;

burst[i]-=tq2;

printf("\nP[%d] : %d",i+1,burst[i]);

}

}

}

printf("\n\n3RD ITERarrivalION"); //Third Iteration

int j,temp;

//sorting for SRTF

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

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

if(burst[j]>burst[j+1]) {

temp=burst[j];

burst[j]=burst[j+1];

burst[j+1]=temp;

}

}

}

int y=0;

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

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

if(i==ar[j]);

y=1;

}

if(y==1){

continue;

}

else{

time+=burst[i];

ct[i]=time;

tarrival[i]=ct[i]-arrival[i];

}

}

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

tarrival[i]=ct[i]-arrival[i];

waiting[i]=tarrival[i]-burst1[i];

printf("\n\nProcess | Compilation Time | TurnAround Time | Waiting Time\n");

printf(" P[%d] \t\t %d \t\t %d \t\t %d \n",i+1,ct[i],tarrival[i],waiting[i]);

avgT=avgT+tarrival[i];

avgW =avgW +waiting[i];

}

avgT =avgT\*1.0/m;

avgW =avgW\*1.0 /m;

printf("\n\nAverage turn arrounf time: %f",avgT);

printf("\nAverage waiting time:%f",avgW);

}

**GitHub Link:**

**<https://github.com/kartik9756/OperatingSystemAssignment/blob/master/Assignment1.c>**

**Problem 2:**

Given five memory partitions of 100 KB, 500 KB, 200 KB, 300 KB, and 600 KB (in order), how would each of the first-fit, best-fit, and worst-fit algorithms place processes of 212 KB, 417 KB, 112 KB, and 426 KB (in order)? Write a program in C which will print the mapping of processes with memory partitions for both the algorithms.(for conceptual clarity refer the textbook).

**Description:**

Above Question is about to make Memory Partition according to the given conditions and constraints. In which, We are given 5 blocks of Memory having different Block Size each.

All we need to do is Partitioning the given memory according to First-Fit Algorithm, Worst-fit Algorithm, and Best-Fit Algorithm with given Processes Size. Output is the mapping of process with memory Partitions.

**Algorithm:**

1. **First Fit:**

For( i=0 to number of process)

{

for(j=0 to number of blocks)

{

if(B[j]>=P[i])

{

tempar[i] = j;

B[j] -= P[i];

break;

} } }

1. **Best Fit:**

for (i=0 to no of process){

temp = -1;

for (j=0 to no of blocks) {

if (B[j] >= P[i]) {

if (temp == -1)

temp = j;

else if (B[temp] > B[j])

temp = j;

} }

if (temp != -1){

tempar[i] = temp;

B[temp] -= P[i];

} }

1. **Worst Fit:**

for (i=0 to no of process){

int t = -1;

for (j=0 to no of blocks){

if (B[j] >= P[i]){

if (t == -1)

t = j;

else if (B[t] < B[j])

t = j;

} }

if (t != -1) {

tempar[i] = t;

B[t] -= P[i];

} }

**Complexity:**

Complexity= O(n2)

**Code snippet:**

#include<stdio.h>

void firstFit(int B[],int m,int P[],int n)

{

int tempar[n];

int i,j;

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

tempar[i]=-1;

}

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

{

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

{

if(B[j]>=P[i])

{

tempar[i] = j;

B[j] -= P[i];

break;

}

}

}

printf("\nProcess No.\tProcess Size\tBlock no.\n");

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

{

printf(" %d\t\t%d\t\t",i+1,P[i]);

if (tempar[i]!=-1)

printf("%d\n",tempar[i]+1);

else

printf( "Not Allocated\n");

}

}

void bestFit(int B[], int m, int P[], int n)

{

int i,j;

int tempar[n];

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

tempar[i]=-1;

}

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

{

int temp = -1;

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

{

if (B[j] >= P[i])

{

if (temp == -1)

temp = j;

else if (B[temp] > B[j])

temp = j;

}

}

if (temp != -1)

{

tempar[i] = temp;

B[temp] -= P[i];

}

}

printf( "\nProcess No.\tProcess Size\tBlock no.\n");

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

{

printf(" %d\t\t%d\t\t",i+1, P[i]);

if (tempar[i] != -1)

printf("%d\n", tempar[i] + 1);

else

printf("Not Allocated\n");

}

}

void worstFit(int B[], int m, int P[], int n)

{

int i,j;

int tempar[n];

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

tempar[i]=-1;

}

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

{

int t = -1;

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

{

if (B[j] >= P[i])

{

if (t == -1)

t = j;

else if (B[t] < B[j])

t = j;

}

}

if (t != -1)

{

tempar[i] = t;

B[t] -= P[i];

}

}

printf( "\nProcess No.\tProcess Size\tBlock no.\n");

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

{

printf(" %d\t\t%d\t\t",i+1, P[i]);

if (tempar[i] != -1)

printf("%d\n", tempar[i] + 1);

else

printf("Not Allocated\n");

}

}

int main()

{

int m,n; int i,j;

int Block[m],Process[n];

printf("\nEnter Total No of Blocks: ");

scanf("%d",&m);

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

printf("\nEnter block Size of Block B[%d]: ",i+1);

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

}

printf("\nEnter Total No of Process: ");

scanf("%d",&n);

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

printf("\nEnter Process Size of Process P[%d]: ",j+1);

scanf("%d",&Process[j]);

}

int b1[99],b2[99],b3[99];

int p1[99],p2[99],p3[99];

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

b1[i]=Block[i];

b2[i]=Block[i];

b3[i]=Block[i];

}

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

p1[j]=Process[j];

p2[j]=Process[j];

p3[j]=Process[j];

}

firstFit(b1, m, p1, n);

bestFit(b2, m, p2, n);

worstFit(b3, m, p3, n);

return 0 ;

}

**GitHub Link:**

**<https://github.com/kartik9756/OperatingSystemAssignment/blob/master/Assignment%2028.cpp>**