OS Lab Number 11

Name: Ahmed Kasteer

Roll Number: 20F-0336

Section: 4D

Question # 1

FCFS

#include<iostream>

using namespace std;

// Function to find the waiting time for all

// processes

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];

}

// Function to calculate turn around time

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];

}

//Function to calculate average time

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

{

int wt[5], tat[5], total\_wt = 0, total\_tat = 0;

//Function to find waiting time of all processes

findWaitingTime(processes, n, bt, wt);

//Function to find turn around time for all processes

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

//Display processes along with all details

cout << "Processes " << " Burst time "<< " Waiting time " << " Turn around time\n";

// Calculate total waiting time and total turn

// around time

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

{

total\_wt = total\_wt + wt[i];

total\_tat = total\_tat + tat[i];

cout << " " << i + 1 << "\t\t" << bt[i] << "\t "<< wt[i] << "\t\t " << tat[i] << endl;

}

cout << "Average waiting time = "

<< (float)total\_wt / (float)n;

cout << "\nAverage turn around time = "

<< (float)total\_tat / (float)n;

}

// Driver code

int main()

{

//process id's

int processes[] = { 1, 2, 3,4,5 };

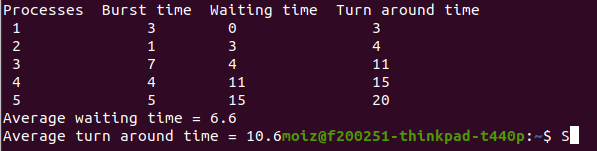
int n = sizeof processes / sizeof processes[0];

//Burst time of all processes

int burst\_time[] = { 3,1,7,4,5 };

findavgTime(processes, n, burst\_time);

return 0;

}

SJF

#include<iostream>

using namespace std;

using namespace std;

//structure for every process

struct Process {

int pid; // Process ID

int bt; // Burst Time

int art; // Arrival Time

};

void findTurnAroundTime(Process proc[], int n, int wt[], int tat[]) {

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

tat[i] = proc[i].bt + wt[i];

}

//waiting time of all process

void findWaitingTime(Process proc[], int n, int wt[]) {

int rt[5];

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

rt[i] = proc[i].bt;

int complete = 0, t = 0, minm = INT\_MAX;

int shortest = 0, finish\_time;

bool check = false;

while (complete != n) {

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

if ((proc[j].art <= t) && (rt[j] < minm) && rt[j] > 0) {

minm = rt[j];

shortest = j;

check = true;

}

}

if (check == false) {

t++;

continue;

}

// decrementing the remaining time

rt[shortest]--;

minm = rt[shortest];

if (minm == 0)

minm = INT\_MAX;

// If a process gets completely

// executed

if (rt[shortest] == 0) {

complete++;

check = false;

finish\_time = t + 1;

// Calculate waiting time

wt[shortest] = finish\_time -

proc[shortest].bt -

proc[shortest].art;

if (wt[shortest] < 0)

wt[shortest] = 0;

}

// Increment time

t++;

}

}

// Function to calculate average time

void findavgTime(Process proc[], int n) {

int wt[5], tat[5], total\_wt = 0,

total\_tat = 0;

// Function to find waiting time of all

// processes

findWaitingTime(proc, n, wt);

// Function to find turn around time for

// all processes

findTurnAroundTime(proc, n, wt, tat);

cout << "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];

cout << " " << proc[i].pid << "\t\t" << proc[i].bt << "\t\t " << wt[i] << "\t\t " << tat[i] << endl;

}

cout << "\nAverage waiting time = " << (float)total\_wt / (float)n; cout << "\nAverage turn around time = " << (float)total\_tat / (float)n;

}

// main function

int main() {

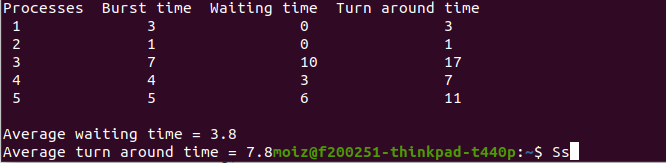
Process proc[] = { { 1, 3, 2 }, { 2, 1, 1 }, { 3, 7, 4 }, { 4, 4, 2 },{ 5, 5, 3 } };

int n = sizeof(proc) / sizeof(proc[0]);

findavgTime(proc, n);

return 0;

}



Priority Scheduling

#include<iostream>

#include<algorithm>

#include<vector>

using namespace std;

#define totalprocess 5

// Making a struct to hold the given input

struct process

{

int at, bt, pr, pno;

};

process proc[50];

/\*

Writing comparator function to sort according to priority if

arrival time is same

\*/

bool comp(process a, process b)

{

if (a.at == b.at)

{

return a.pr < b.pr;

}

else

{

return a.at < b.at;

}

}

// Using FCFS Algorithm to find Waiting time

void get\_wt\_time(int wt[])

{

// declaring service array that stores cumulative burst time

int service[50];

// Initialising initial elements of the arrays

service[0] = proc[0].at;

wt[0] = 0;

for (int i = 1; i < totalprocess; i++)

{

service[i] = proc[i - 1].bt + service[i - 1];

wt[i] = service[i] - proc[i].at;

// If waiting time is negative, change it into zero

if (wt[i] < 0)

{

wt[i] = 0;

}

}

}

void get\_tat\_time(int tat[], int wt[])

{

// Filling turnaroundtime array

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

{

tat[i] = proc[i].bt + wt[i];

}

}

void findgc()

{

//Declare waiting time and turnaround time array

int wt[50], tat[50];

double wavg = 0, tavg = 0;

// Function call to find waiting time array

get\_wt\_time(wt);

//Function call to find turnaround time

get\_tat\_time(tat, wt);

int stime[50], ctime[50];

stime[0] = proc[0].at;

ctime[0] = stime[0] + tat[0];

// calculating starting and ending time

for (int i = 1; i < totalprocess; i++)

{

stime[i] = ctime[i - 1];

ctime[i] = stime[i] + tat[i] - wt[i];

}

cout << "Process\_no\tStart\_time\tComplete\_time\tTurn\_Around\_Time\tWaiting\_Time" << endl;

// display the process details

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

{

wavg += wt[i];

tavg += tat[i];

cout << proc[i].pno << "\t\t" <<

stime[i] << "\t\t" << ctime[i] << "\t\t" <<

tat[i] << "\t\t\t" << wt[i] << endl;

}

// display the average waiting time

//and average turn around time

cout << "Average waiting time is : ";

cout << wavg / (float)totalprocess << endl;

cout << "average turnaround time : ";

cout << tavg / (float)totalprocess << endl;

}

int main()

{

int arrivaltime[] = { 1, 2, 3, 4, 5 };

int bursttime[] = { 3, 1, 7, 4,5 };

int priority[] = { 2,1,4,2,3 };

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

{

proc[i].at = arrivaltime[i];

proc[i].bt = bursttime[i];

proc[i].pr = priority[i];

proc[i].pno = i + 1;

}

//Using inbuilt sort function

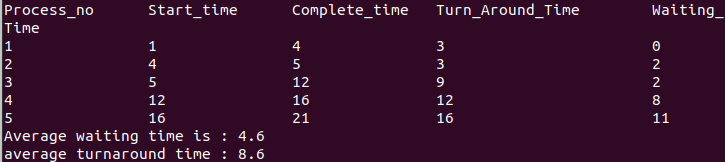
sort(proc, proc + totalprocess, comp);

//Calling function findgc for finding Gantt Chart

findgc();

return 0;

}



Round Robin

#include<iostream>

using namespace std;

// Function to find the waiting time for all

// processes

void findWaitingTime(int processes[], int n,

int bt[], int wt[], int quantum)

{

// Make a copy of burst times bt[] to store remaining

// burst times.

int rem\_bt[5];

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

rem\_bt[i] = bt[i];

int t = 0; // Current time

// Keep traversing processes in round robin manner

// until all of them are not done.

while (1)

{

bool done = true;

// Traverse all processes one by one repeatedly

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

{

// If burst time of a process is greater than 0

// then only need to process further

if (rem\_bt[i] > 0)

{

done = false; // There is a pending process

if (rem\_bt[i] > quantum)

{

// Increase the value of t i.e. shows

// how much time a process has been processed

t += quantum;

// Decrease the burst\_time of current process

// by quantum

rem\_bt[i] -= quantum;

}

// If burst time is smaller than or equal to

// quantum. Last cycle for this process

else

{

// Increase the value of t i.e. shows

// how much time a process has been processed

t = t + rem\_bt[i];

// Waiting time is current time minus time

// used by this process

wt[i] = t - bt[i];

// As the process gets fully executed

// make its remaining burst time = 0

rem\_bt[i] = 0;

}

}

}

// If all processes are done

if (done == true)

break;

}

}

// Function to calculate turn around time

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];

}

// Function to calculate average time

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

int quantum)

{

int wt[5], tat[5], total\_wt = 0, total\_tat = 0;

// Function to find waiting time of all processes

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

// Function to find turn around time for all processes

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

// Display processes along with all details

cout << "PN\t " << " \tBT\t " << " WT\t " << " \tTAT\n";

// Calculate total waiting time and total turn

// around time

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

{

total\_wt = total\_wt + wt[i];

total\_tat = total\_tat + tat[i];

cout << " " << i + 1 << "\t\t" << bt[i] << "\t " << wt[i] << "\t\t " << tat[i] << endl;

}

cout << "Average waiting time = "

<< (float)total\_wt / (float)n;

cout << "\nAverage turn around time = "

<< (float)total\_tat / (float)n;

}

// Driver code

int main()

{

// process id's

int processes[] = { 1,2,3,4,5 };

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

// Burst time of all processes

int burst\_time[] = { 3,1,7,4,5 };

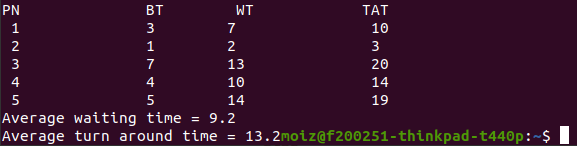
// Time quantum

int quantum = 2;

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

return 0;

}



Question # 2

#include<iostream>

#include<algorithm>

#include<vector>

using namespace std;

struct Process {

int pid; // Process ID

int bt; // CPU Burst time required

int priority; // Priority of this process

};

// sorting the Process acc. to the priority

bool compare(Process a, Process b) {

return (a.priority > b.priority);

}

void waitingtime(Process pro[], int n, int wt[]) {

// Initial waiting time for a process is 0

wt[0] = 0;

// calculating waiting time

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

wt[i] = pro[i - 1].bt + wt[i - 1];

}

// Function to calculate turn around time

void turnarround(Process pro[], int n, int wt[], int tat[]) {

// calculating turnaround time by adding

// bt[i] + wt[i]

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

tat[i] = pro[i].bt + wt[i];

}

//Function to calculate average time

void avgtime(Process pro[], int n) {

int wt[5], tat[5], total\_wt = 0, total\_tat = 0;

//Function to find waiting time of all processes

waitingtime(pro, n, wt);

//Function to find turn around time for all processes

turnarround(pro, n, wt, tat);

//Display processes along with all details

cout << "\nProcesses " << " Burst time " << " Waiting time " << " Turn around time\n";

// Calculate total waiting time and total turn

// around time

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

total\_wt = total\_wt + wt[i];

total\_tat = total\_tat + tat[i];

cout << " " << pro[i].pid << "\t\t" << pro[i].bt << "\t " << wt[i] << "\t\t " << tat[i] << endl;

}

cout << "\nAverage waiting time = " << (float)total\_wt / (float)n;

cout << "\nAverage turn around time = " << (float)total\_tat / (float)n;

}

void scheduling(Process pro[], int n) {

// Sort processes by priority

sort(pro, pro + n, compare);

cout << "Order in which processes gets executed \n";

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

cout << pro[i].pid << " ";

avgtime(pro, n);

}

// main function

int main() {

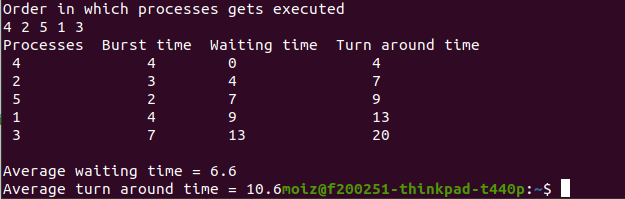
Process pro[] = { {1, 4, 1}, {2, 3, 2}, {3, 7, 1},{4, 4, 3},{5, 2, 2} };

int n = sizeof pro / sizeof pro[0];

scheduling(pro, n);

return 0;

}



Question # 3

#include <iostream>

using namespace std;

struct Process {

int pid; // Process ID

int bt; // Burst Time

int art; // Arrival Time

};

// Function to find the waiting time for all

// processes

void findWaitingTime(Process proc[], int n,int wt[])

{

int rt[5];

// Copy the burst time into rt[]

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

rt[i] = proc[i].bt;

int complete = 0, t = 0, minm = INT\_MAX;

int shortest = 0, finish\_time;

bool check = false;

// Process until all processes gets

// completed

while (complete != n) {

// Find process with minimum

// remaining time among the

// processes that arrives till the

// current time`

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

if ((proc[j].art <= t) &&

(rt[j] < minm) && rt[j] > 0) {

minm = rt[j];

shortest = j;

check = true;

}

}

if (check == false) {

t++;

continue;

}

// Reduce remaining time by one

rt[shortest]--;

// Update minimum

minm = rt[shortest];

if (minm == 0)

minm = INT\_MAX;

// If a process gets completely

// executed

if (rt[shortest] == 0) {

// Increment complete

complete++;

check = false;

// Find finish time of current

// process

finish\_time = t + 1;

// Calculate waiting time

wt[shortest] = finish\_time -

proc[shortest].bt -

proc[shortest].art;

if (wt[shortest] < 0)

wt[shortest] = 0;

}

// Increment time

t++;

}

}

// Function to calculate turn around time

void findTurnAroundTime(Process proc[], int n,

int wt[], int tat[])

{

// calculating turnaround time by adding

// bt[i] + wt[i]

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

tat[i] = proc[i].bt + wt[i];

}

// Function to calculate average time

void findavgTime(Process proc[], int n)

{

int wt[5], tat[5], total\_wt = 0,

total\_tat = 0;

// Function to find waiting time of all

// processes

findWaitingTime(proc, n, wt);

// Function to find turn around time for

// all processes

findTurnAroundTime(proc, n, wt, tat);

// Display processes along with all

// details

cout << " P\t\t"<< "BT\t\t"<< "WT\t\t"<< "TAT\t\t\n";

// Calculate total waiting time and

// total turnaround time

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

total\_wt = total\_wt + wt[i];

total\_tat = total\_tat + tat[i];

cout << " " << proc[i].pid << "\t\t"<< proc[i].bt << "\t\t " << wt[i]<< "\t\t " << tat[i] << endl;

}

cout << "\nAverage waiting time = "<< (float)total\_wt / (float)n;

cout << "\nAverage turn around time = "<< (float)total\_tat / (float)n;

}

// Driver code

int main()

{

Process proc[] = { { 1, 6, 2 }, { 2, 2, 5 },{ 3, 8, 1 }, { 4, 3, 0}, {5, 4, 4} };

int n = sizeof(proc) / sizeof(proc[0]);

findavgTime(proc, n);

return 0;

}

