OS Lab Assignment 3

# Name: Keshav Garg

## Id: 2018UCP1674

My GitHub repository containing this file: [Click here](https://github.com/gargk747/OS-Lab/tree/master/Assignment%203)

1. FCFS

#include <bits/stdc++.h>

using namespace std;

struct Process

{

int id;

int burst\_time;

int arrival\_time;

int start\_time;

int finish\_time;

int wait\_time;

};

bool compare(Process x, Process y)

{

return x.arrival\_time < y.arrival\_time;

}

void about()

{

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\t" << "FCFS CPU SCHEDULING ALGORITHM SIMULATION" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

}

void PrintProcessData(Process process)

{

cout << process.id << "\t" << process.arrival\_time << "\t" << process.start\_time << "\t"

<< process.burst\_time << "\t" << process.finish\_time << "\t" << process.wait\_time << endl;

}

int main()

{

srand(time(0));

about();

int pcount;

cout << "ENTER NO OF PROCESSES: ";

cin >> pcount;

Process processes[pcount];

cout << "ARRIVAL TIME AND BURST TIME ARE GENERATED THROUGH RANDOM FUNCTION FOR EACH PROCESS." << endl;

cout << endl;

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

{

int id = i + 1;

int burst\_time = rand() % 20;

int arrival\_time = rand() % 15;

processes[i].id = id;

processes[i].arrival\_time = arrival\_time;

processes[i].burst\_time = burst\_time;

}

sort(processes, processes + pcount, compare);

cout << endl;

cout << "PID" << "\t" << "ARRIVAL" << "\t" << "START" << "\t"

<< "BURST" << "\t" << "FINISH" << "\t" << "WAIT" << endl;

processes[0].start\_time = processes[0].arrival\_time;

processes[0].finish\_time = processes[0].start\_time + processes[0].burst\_time;

processes[0].wait\_time = processes[0].start\_time - processes[0].arrival\_time;

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

{

processes[i].start\_time = max(processes[i].arrival\_time, processes[i - 1].finish\_time);

processes[i].finish\_time = processes[i].start\_time + processes[i].burst\_time;

processes[i].wait\_time = processes[i].start\_time - processes[i].arrival\_time;

}

float totalWaitTime = 0;

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

{

PrintProcessData(processes[i]);

totalWaitTime += processes[i].wait\_time;

}

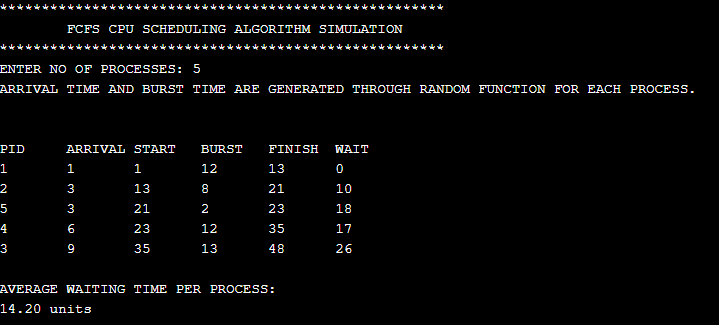
cout << endl;

cout << "AVERAGE WAITING TIME PER PROCESS:" << endl;

cout << fixed << setprecision(2) << totalWaitTime / float(pcount) << " units" << endl;

return 0;

}



1. SJF

#include <bits/stdc++.h>

using namespace std;

struct Process

{

int id;

int burst\_time;

int arrival\_time;

int finish\_time;

int rem\_burst\_time;

int wait\_time;

};

bool compare(Process x, Process y)

{

if (x.arrival\_time == y.arrival\_time)

{

// 2 is more prior than 4 or more

return x.burst\_time < y.burst\_time;

}

return x.arrival\_time < y.arrival\_time;

}

void about()

{

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\t"

<< "SJF PREEMPTIVE CPU SCHEDULING ALGORITHM SIMULATION" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

}

void PrintProcessData(Process process)

{

cout << process.id << "\t" << process.arrival\_time << "\t" << process.burst\_time << "\t" << process.finish\_time << "\t" << process.wait\_time << endl;

}

int main()

{

srand(time(0));

about();

int pcount;

cout << "ENTER NO OF PROCESSES: ";

cin >> pcount;

Process processes[pcount];

cout << "BURST TIME AND ARRIVAL TIME ARE GENERATED USING RANDOM FUNCTION FOR EACH PROCESS." << endl;

cout << endl;

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

{

int id = i + 1;

int burst\_time, arrival\_time;

burst\_time = rand() % 20 + 1;

arrival\_time = rand() % 15;

// cin >> burst\_time >> arrival\_time;

processes[i].id = id;

processes[i].arrival\_time = arrival\_time;

processes[i].burst\_time = burst\_time;

processes[i].rem\_burst\_time = burst\_time;

}

sort(processes, processes + pcount, compare);

int current\_time = 0;

while (1)

{

// we are checking for shortest burst\_time after each second

int short\_burst\_index = 0;

int temp\_burst\_time = INT\_MAX;

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

{

if (

processes[i].rem\_burst\_time &&

(processes[i].burst\_time < temp\_burst\_time) &&

(processes[i].arrival\_time <= current\_time)

) {

short\_burst\_index = i;

temp\_burst\_time = processes[i].burst\_time;

}

}

current\_time++;

if (processes[short\_burst\_index].rem\_burst\_time)

{

processes[short\_burst\_index].rem\_burst\_time -= 1;

if (processes[short\_burst\_index].rem\_burst\_time == 0)

{

processes[short\_burst\_index].finish\_time = current\_time;

processes[short\_burst\_index].wait\_time = processes[short\_burst\_index].finish\_time - processes[short\_burst\_index].burst\_time - processes[short\_burst\_index].arrival\_time;

}

}

bool poss = true;

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

if (processes[i].rem\_burst\_time) {

poss = false;

}

}

if (poss) {

break;

}

// cout << current\_time << " " << processes[short\_burst\_index].id << endl;

}

cout << "PID" << "\t" << "ARRIVAL" << "\t" << "BURST" << "\t"

<< "FINISH" << "\t" << "WAIT" << endl;

float totalWaitTime = 0;

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

{

PrintProcessData(processes[i]);

totalWaitTime += processes[i].wait\_time;

}

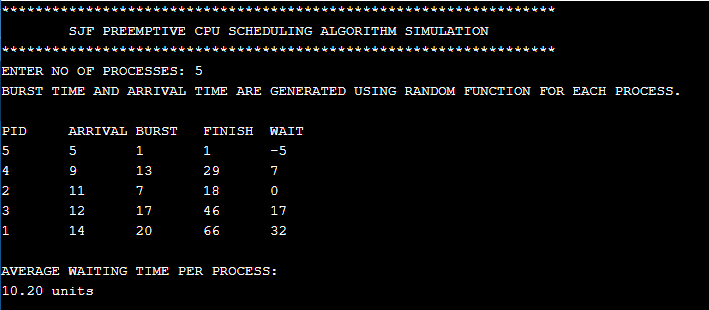
cout << endl;

cout << "AVERAGE WAITING TIME PER PROCESS:" << endl;

cout << fixed << setprecision(2) << totalWaitTime / float(pcount) << " units" << endl;

return 0;

}



1. Priority (non-preemptive)

#include <bits/stdc++.h>

using namespace std;

struct Process

{

int id;

int burst\_time;

int arrival\_time;

int priority;

int start\_time;

int finish\_time;

int wait\_time;

};

bool compare(Process x, Process y)

{

if (x.arrival\_time == y.arrival\_time)

{

// 2 is more prior than 4 or more

return x.priority < y.priority;

}

return x.arrival\_time < y.arrival\_time;

}

void about()

{

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\t" << "PRIORITY NON PREEMPTIVE CPU SCHEDULING ALGORITHM SIMULATION" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

}

void PrintProcessData(Process process)

{

cout << process.id << "\t" << process.priority << "\t" << process.arrival\_time << "\t" << process.start\_time << "\t"

<< process.burst\_time << "\t" << process.finish\_time << "\t" << process.wait\_time << endl;

}

int main()

{

srand(time(0));

about();

int pcount;

cout << "ENTER NO OF PROCESSES: ";

cin >> pcount;

Process processes[pcount];

cout << "PRIORITY, BURST TIME AND ARRIVAL TIME ARE GENERATED USING RANDOM FUNCTION FOR EACH PROCESS." << endl;

cout << endl;

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

{

int id = i + 1;

int burst\_time = rand() % 20 + 1;

int arrival\_time = rand() % 15;

int priority = rand() % 10;

processes[i].id = id;

processes[i].priority = priority;

processes[i].arrival\_time = arrival\_time;

processes[i].burst\_time = burst\_time;

}

sort(processes, processes + pcount, compare);

cout << endl;

cout << "PID" << "\t" << "PRIOR" << "\t" << "ARRIVAL" << "\t" << "START" << "\t"

<< "BURST" << "\t" << "FINISH" << "\t" << "WAIT" << endl;

processes[0].start\_time = processes[0].arrival\_time;

processes[0].finish\_time = processes[0].start\_time + processes[0].burst\_time;

processes[0].wait\_time = processes[0].start\_time - processes[0].arrival\_time;

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

{

int best\_burst\_index = i;

for (int j = i; j < pcount; j++)

{

if (processes[j].arrival\_time >= processes[i - 1].finish\_time)

{

break;

}

else

{

if (processes[j].priority < processes[best\_burst\_index].priority)

{

best\_burst\_index = j;

}

}

}

// shifting

Process best\_burst\_process = processes[best\_burst\_index];

for (int k = best\_burst\_index; k > i; k--)

{

processes[k] = processes[k - 1];

}

processes[i] = best\_burst\_process;

processes[i].start\_time = max(processes[i].arrival\_time, processes[i - 1].finish\_time);

processes[i].finish\_time = processes[i].start\_time + processes[i].burst\_time;

processes[i].wait\_time = processes[i].start\_time - processes[i].arrival\_time;

}

float totalWaitTime = 0;

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

{

PrintProcessData(processes[i]);

totalWaitTime += processes[i].wait\_time;

}

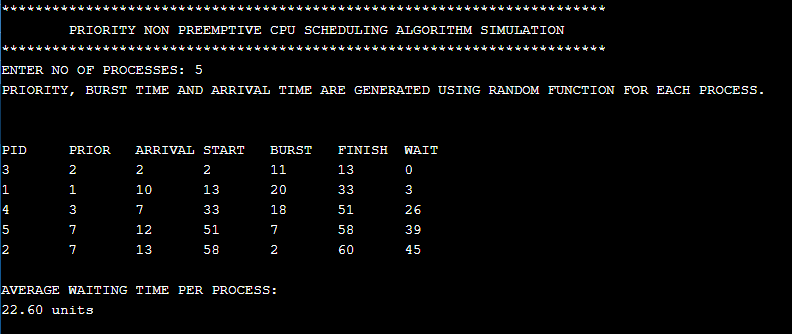
cout << endl;

cout << "AVERAGE WAITING TIME PER PROCESS:" << endl;

cout << fixed << setprecision(2) << totalWaitTime / float(pcount) << " units" << endl;

return 0;

}



1. Preemptive priority

#include <bits/stdc++.h>

using namespace std;

struct Process

{

int id;

int burst\_time;

int arrival\_time;

int finish\_time;

int priority;

int rem\_burst\_time;

int wait\_time;

};

bool compare(Process x, Process y)

{

if (x.arrival\_time == y.arrival\_time)

{

// 2 is more prior than 4 or more

return x.priority < y.priority;

}

return x.arrival\_time < y.arrival\_time;

}

void about()

{

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\t"

<< "PRIORITY PREEMPTIVE CPU SCHEDULING ALGORITHM SIMULATION" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

}

void PrintProcessData(Process process)

{

cout << process.id << "\t" << process.priority << "\t" << process.arrival\_time << "\t" << process.burst\_time << "\t" << process.finish\_time << "\t" << process.wait\_time << endl;

}

int main()

{

srand(time(0));

about();

int pcount;

cout << "ENTER NO OF PROCESSES: ";

cin >> pcount;

Process processes[pcount];

cout << "PRIORITY, BURST TIME AND ARRIVAL TIME ARE GENERATED USING RANDOM FUNCTION FOR EACH PROCESS." << endl;

cout << endl;

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

{

int id = i + 1;

int priority, burst\_time, arrival\_time;

burst\_time = rand() % 20 + 1;

arrival\_time = rand() % 15;

priority = rand() % 20;

// cin >> priority >> burst\_time >> arrival\_time;

processes[i].id = id;

processes[i].priority = priority;

processes[i].arrival\_time = arrival\_time;

processes[i].burst\_time = burst\_time;

processes[i].rem\_burst\_time = burst\_time;

}

sort(processes, processes + pcount, compare);

int current\_time = 0;

while (1)

{

// we are checking for shortest burst\_time after each second

int short\_burst\_index = 0;

int temp\_priority = INT\_MAX;

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

{

if (

processes[i].rem\_burst\_time &&

(processes[i].priority < temp\_priority) &&

(processes[i].arrival\_time <= current\_time)

) {

short\_burst\_index = i;

temp\_priority = processes[i].priority;

}

}

current\_time++;

if (processes[short\_burst\_index].rem\_burst\_time)

{

processes[short\_burst\_index].rem\_burst\_time -= 1;

if (processes[short\_burst\_index].rem\_burst\_time == 0)

{

processes[short\_burst\_index].finish\_time = current\_time;

processes[short\_burst\_index].wait\_time = processes[short\_burst\_index].finish\_time - processes[short\_burst\_index].burst\_time - processes[short\_burst\_index].arrival\_time;

}

}

bool poss = true;

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

if (processes[i].rem\_burst\_time) {

poss = false;

}

}

if (poss) {

break;

}

// cout << current\_time << " " << processes[short\_burst\_index].id << endl;

}

cout << "PID" << "\t" << "PRIOR" << "\t" << "ARRIVAL" << "\t" << "BURST" << "\t"

<< "FINISH" << "\t" << "WAIT" << endl;

float totalWaitTime = 0;

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

{

PrintProcessData(processes[i]);

totalWaitTime += processes[i].wait\_time;

}

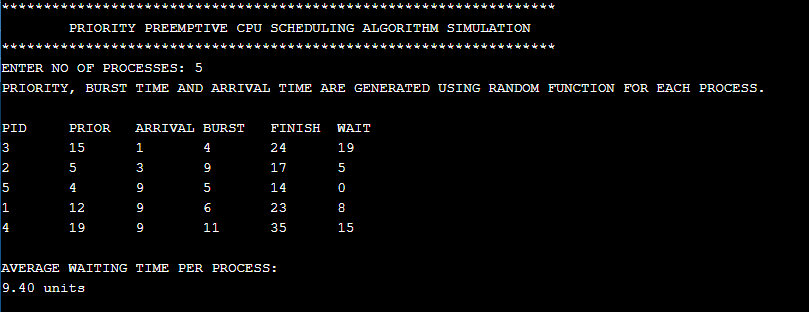
cout << endl;

cout << "AVERAGE WAITING TIME PER PROCESS:" << endl;

cout << fixed << setprecision(2) << totalWaitTime / float(pcount) << " units" << endl;

return 0;

}



1. MLFQ

#include <bits/stdc++.h>

using namespace std;

struct Process

{

int id;

int burst\_time;

int arrival\_time;

int start\_time;

int finish\_time;

int wait\_time;

};

bool compare(Process x, Process y)

{

if (x.arrival\_time == y.arrival\_time)

{

return x.burst\_time < y.burst\_time;

}

return x.arrival\_time < y.arrival\_time;

}

void about()

{

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\t" << "MLFQ CPU SCHEDULING ALGORITHM SIMULATION" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

}

void PrintProcessData(Process process)

{

cout << process.id << "\t" << process.arrival\_time << "\t" << process.start\_time << "\t"

<< process.burst\_time << "\t" << process.finish\_time << "\t" << process.wait\_time << endl;

}

int main()

{

srand(time(0));

about();

int pcount;

cout << "ENTER NO OF PROCESSES: ";

cin >> pcount;

Process processes[pcount];

cout << "BURST TIME AND ARRIVAL TIME ARE GENERATED USING RANDOM FUNCTION FOR EACH PROCESS." << endl;

cout << endl;

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

{

int id = i + 1;

int burst\_time = rand() % 20;

int arrival\_time = rand() % 15;

processes[i].id = id;

processes[i].arrival\_time = arrival\_time;

processes[i].burst\_time = burst\_time;

}

sort(processes, processes + pcount, compare);

cout << endl;

cout << "ID" << "\t" << "ARRIVAL" << "\t" << "START" << "\t"

<< "BURST" << "\t" << "FINISH" << "\t" << "WAIT" << endl;

processes[0].start\_time = processes[0].arrival\_time;

processes[0].finish\_time = processes[0].start\_time + processes[0].burst\_time;

processes[0].wait\_time = processes[0].start\_time - processes[0].arrival\_time;

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

{

int best\_burst\_index = i;

for (int j = i; j < pcount; j++)

{

if (processes[j].arrival\_time >= processes[i - 1].finish\_time)

{

break;

}

else

{

if (processes[j].burst\_time < processes[best\_burst\_index].burst\_time)

{

best\_burst\_index = j;

}

}

}

// shifting

Process best\_burst\_process = processes[best\_burst\_index];

for (int k = best\_burst\_index; k > i; k--)

{

processes[k] = processes[k - 1];

}

processes[i] = best\_burst\_process;

processes[i].start\_time = max(processes[i].arrival\_time, processes[i - 1].finish\_time);

processes[i].finish\_time = processes[i].start\_time + processes[i].burst\_time;

processes[i].wait\_time = processes[i].start\_time - processes[i].arrival\_time;

}

float totalWaitTime = 0;

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

{

PrintProcessData(processes[i]);

totalWaitTime += processes[i].wait\_time;

}

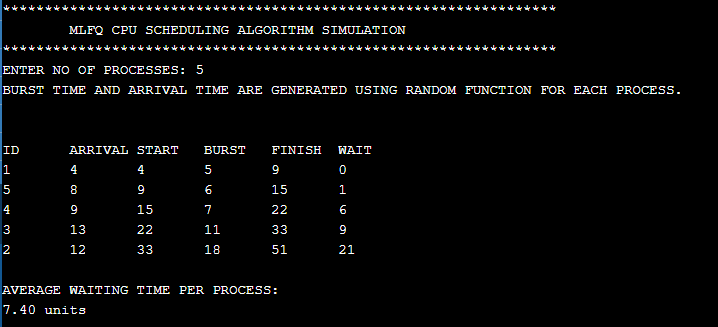
cout << endl;

cout << "AVERAGE WAITING TIME PER PROCESS:" << endl;

cout << fixed << setprecision(2) << totalWaitTime / float(pcount) << " units" << endl;

return 0;

}



1. Round Robin

#include <bits/stdc++.h>

using namespace std;

struct Process

{

int id;

int burst\_time;

int finish\_time;

int rem\_burst\_time;

int wait\_time;

};

void about()

{

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\t"

<< "ROUND ROBIN CPU SCHEDULING ALGORITHM SIMULATION" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

}

void PrintProcessData(Process process)

{

cout << process.id << "\t"

<< process.burst\_time << "\t" << process.finish\_time << "\t" << process.wait\_time << endl;

}

int main()

{

srand(time(0));

about();

int pcount, time\_slice;

cout << "ENTER NO OF PROCESSES: ";

cin >> pcount;

cout << "ENTER TIME SLICE: ";

cin >> time\_slice;

if (time\_slice <= 0)

{

cout << "Time slice must be greater than zero.";

return 0;

}

vector<Process> processes(pcount);

cout << "BURST TIME IS GENERATED USING RANDOM FUNCTION FOR EACH PROCESS." << endl;

cout << endl;

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

{

int id = i + 1;

int burst\_time;

burst\_time = rand() % 20 + 1;

// cin >> burst\_time;

processes[i].id = id;

processes[i].burst\_time = burst\_time;

processes[i].rem\_burst\_time = burst\_time;

}

cout << "Round robin sequence for processes:" << endl;

int current\_time = 0;

while (1)

{

bool poss = true;

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

{

if (processes[i].rem\_burst\_time)

{

if (processes[i].rem\_burst\_time > time\_slice)

{

processes[i].rem\_burst\_time -= time\_slice;

current\_time += time\_slice;

}

else

{

current\_time += processes[i].rem\_burst\_time;

processes[i].rem\_burst\_time = 0;

processes[i].finish\_time = current\_time;

processes[i].wait\_time = processes[i].finish\_time - processes[i].burst\_time;

}

cout << "P" << processes[i].id << " ";

}

if (processes[i].rem\_burst\_time) {

poss = false;

}

}

if (poss)

{

break;

}

}

cout << "\n" << endl;

cout << "PID" << "\t" << "BURST" << "\t" << "FINISH" << "\t" << "WAIT" << endl;

float totalWaitTime = 0;

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

{

PrintProcessData(processes[i]);

totalWaitTime += processes[i].wait\_time;

}

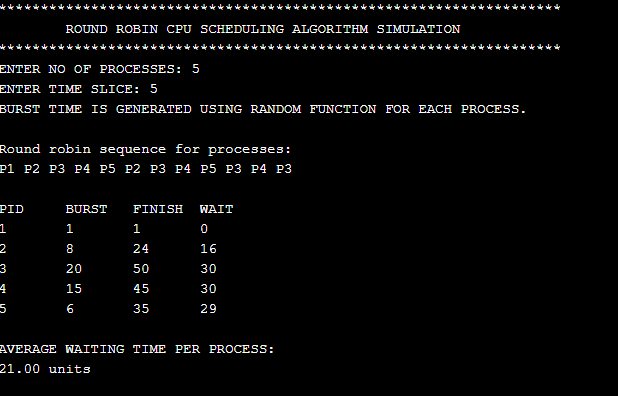
cout << endl;

cout << "AVERAGE WAITING TIME PER PROCESS:" << endl;

cout << fixed << setprecision(2) << totalWaitTime / float(pcount) << " units" << endl;

return 0;

}



1. Lottery

#include <bits/stdc++.h>

using namespace std;

struct Process

{

int id;

int burst\_time;

int arrival\_time;

int start\_time;

int finish\_time;

int wait\_time;

};

bool compare(Process x, Process y)

{

return x.arrival\_time < y.arrival\_time;

}

void about()

{

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\t" << "Lottery CPU SCHEDULING ALGORITHM SIMULATION" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

}

void PrintProcessData(Process process)

{

cout << process.id << "\t" << process.arrival\_time << "\t" << process.start\_time << "\t"

<< process.burst\_time << "\t" << process.finish\_time << "\t" << process.wait\_time << endl;

}

int main()

{

srand(time(0));

about();

int pcount;

cout << "ENTER NO OF PROCESSES: ";

cin >> pcount;

Process processes[pcount];

cout << "ARRIVAL TIME AND BURST TIME ARE GENERATED THROUGH RANDOM FUNCTION FOR EACH PROCESS." << endl;

cout << endl;

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

{

int id = i + 1;

int burst\_time = rand() % 20;

int arrival\_time = rand() % 15;

processes[i].id = id;

processes[i].arrival\_time = arrival\_time;

processes[i].burst\_time = burst\_time;

}

sort(processes, processes + pcount, compare);

cout << endl;

cout << "PID" << "\t" << "ARRIVAL" << "\t" << "START" << "\t"

<< "BURST" << "\t" << "FINISH" << "\t" << "WAIT" << endl;

processes[0].start\_time = processes[0].arrival\_time;

processes[0].finish\_time = processes[0].start\_time + processes[0].burst\_time;

processes[0].wait\_time = processes[0].start\_time - processes[0].arrival\_time;

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

{

processes[i].start\_time = max(processes[i].arrival\_time, processes[i - 1].finish\_time);

processes[i].finish\_time = processes[i].start\_time + processes[i].burst\_time;

processes[i].wait\_time = processes[i].start\_time - processes[i].arrival\_time;

}

float totalWaitTime = 0;

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

{

PrintProcessData(processes[i]);

totalWaitTime += processes[i].wait\_time;

}

cout << endl;

cout << "AVERAGE WAITING TIME PER PROCESS:" << endl;

cout << fixed << setprecision(2) << totalWaitTime / float(pcount) << " units" << endl;

return 0;

}

