**Round Robin Algorithm Implementation**

#include <iostream>

#include <queue>

#include <iomanip>

using namespace std;

class Process

{

friend void Calculate();

int id;

static int count;

int burst\_time;

int arrival\_time;

int completion\_time;

int turnaround\_time;

int waiting\_time;

public:

Process() :id(count) { completion\_time = turnaround\_time = waiting\_time = 0; count++; }

void input()

{

cout << "Process: " << id << endl

<< "Enter Arrival Time: "; cin >> arrival\_time;

cout << "Enter Burst Time: "; cin >> burst\_time;

}

int getid() { return id; }

void Display();

void Display\_Before();

};

int Process::count = 1;

void Process::Display()

{

cout << setw(15) << id << setw(15) << arrival\_time << setw(15) << burst\_time << setw(15) << turnaround\_time << setw(15) << waiting\_time;

cout << endl;

}

void Process::Display\_Before()

{

cout << setw(15) << id << setw(15) << arrival\_time << setw(15) << burst\_time << endl;

}

int time\_slice;

void Calculate();

queue<Process\*> Queue;

Process \*obj;

int main()

{

cout << endl << left << setw(30) << " " << "ROUND ROBIN ALGORITHM" << endl << endl;

int num;

do

{

cout << "Enter Number of Processes: ";

cin >> num;

} while (num < 1);

obj = new Process[num];

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

obj[i].input();

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

Queue.push(&obj[i]);

cout << "Enter Time Slice for Round Robin: "; cin >> time\_slice;

// Printing Before Calculation

cout << endl << endl << setw(15) << "Process ID" << setw(15) << "Arrival Time" << setw(15) << "Burst Time" << endl

<< "------------------------------------------------------------------" << endl;

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

obj[i].Display\_Before();

Calculate();

// Printing After Calculation

cout << endl << endl << setw(15) << "Process ID" << setw(15) << "Arrival Time" << setw(15) << "Burst Time"

<< setw(15) << "Turnaround" << setw(15) << "Waiting Time" << endl

<< "------------------------------------------------------------------" << endl;

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

{

for (int j = 0; j < num - i; ++j)

if (obj[j].getid() > obj[j + 1].getid())

swap(obj[j], obj[j + 1]);

}

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

obj[i].Display();

cout << endl;

system("pause");

return 0;

}

void Calculate()

{

int total\_time = 0;

while (!Queue.empty())

{

Process \*temp;

temp = Queue.front();

Queue.pop();

if (temp->arrival\_time > total\_time)

Queue.push(temp);

else

{

if (temp->burst\_time <= time\_slice)

{

total\_time = total\_time + temp->burst\_time;

obj[temp->id-1].completion\_time = total\_time;

obj[temp->id - 1].turnaround\_time = temp->completion\_time - temp->arrival\_time;

obj[temp->id - 1].waiting\_time = temp->turnaround\_time - temp->burst\_time;

}

else

{

total\_time = total\_time + time\_slice;

temp->burst\_time = temp->burst\_time - time\_slice;

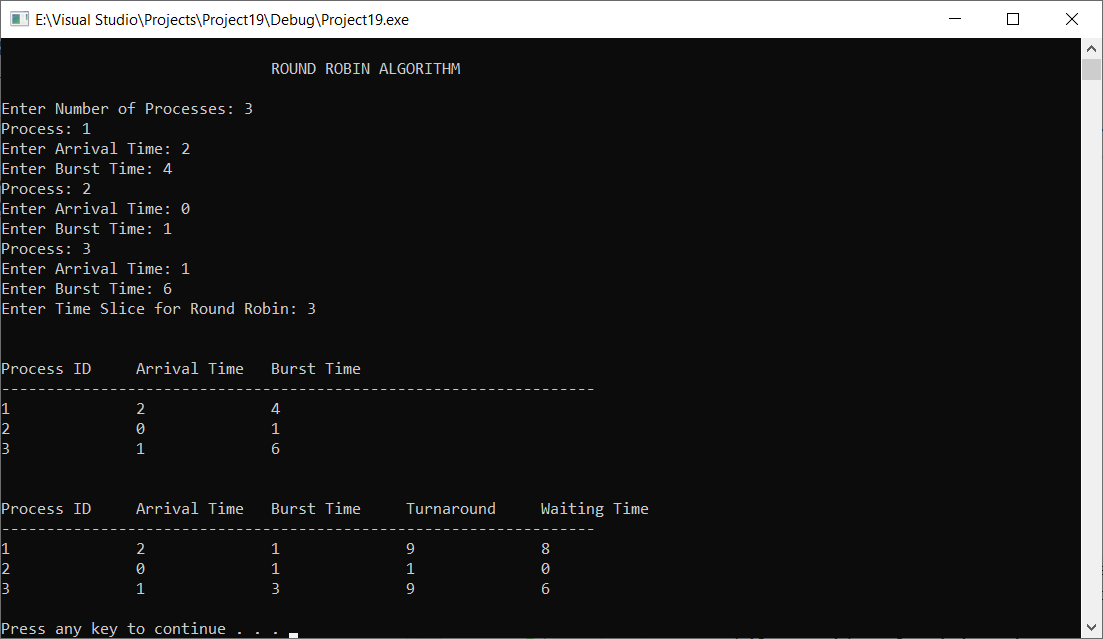
Queue.push(temp);

}

}

}

}



**Priority Scheduling Algorithm (Non-Preemptive)**