**UNIVERSITY OF SARGODHA**

**CS&IT DEPARTMENT**

**Group Members:**

1. Zain Abbas (BSIT51F22R049)
2. Muhammad Farhan (BSIT51F22R040)

**Class:** BSIT-4th SS1

**Subject:** Operating System(Lab)

**Project:** Scheduling Algorithms with GUI Interface

**Teacher:** Sir Waleed

**DOCUMENTATION:**

**1. Introduction**

- Purpose and goals of the project: We developed this operating system project to explore new scheduling criteria and demonstrate concepts in operating system design. My goal was to address specific needs and challenges in scheduling algorithms while providing a platform for learning and experimentation.

- Target audience: The intended users of this operating system are developers, researchers, students, and anyone interested in understanding scheduling criteria in operating systems.

**2. System Architecture**

- Overview of the kernel and user space: The operating system has a clear distinction between kernel space (privileged mode) and user space (user mode), ensuring system stability and security.

- Description of key components and subsystems: Key components such as the scheduler, memory manager, file system, and device drivers form the backbone of the operating system, enabling efficient resource management and device interaction.

**3. Design Principles**

- Principles guiding the development: Our design philosophy focuses on modularity, efficiency, and simplicity, ensuring that the operating system is both robust and easy to understand and extend.

**4. Functional Overview**

- Process management: Processes are managed using various scheduling algorithms, including First Come First Serve, Round Robin, and Priority Scheduling, providing flexibility and responsiveness.

- Memory management: Memory is efficiently managed and allocated, utilizing virtual memory systems and paging mechanisms to optimize system performance.

- File system support: The operating system supports various file system operations, allowing users to create, read, write, and delete files with ease.

- Device drivers: A wide range of devices is supported, and device drivers facilitate seamless interaction between the operating system and hardware peripherals.

**5. System Requirements**

- Hardware requirements: The minimum hardware requirements include CPU architecture, memory size, and storage capacity to ensure smooth operation of the operating system.

- Software dependencies: Certain software dependencies, such as compilers, libraries, and development tools, are needed for building, installing, and running the operating system.

**6. Installation and Configuration**

- Instructions for installation: Step-by-step instructions guide users through the installation process, ensuring a smooth setup experience.

- Configuration options: Users can customize various settings and configurations to tailor the operating system to their specific needs and preferences.

**7. User Interface**

- Command-line interface: The command-line interface (CLI) provides powerful commands and syntax for interacting with the operating system, offering flexibility and control.

- Graphical user interface: A user-friendly graphical interface (GUI) enhances the user experience, with intuitive layout, features, and interaction mechanisms.

**8. Security**

- Security features implemented: Robust security features are implemented to safeguard against unauthorized access, data breaches, and other security threats.

- Measures to protect against security threats: Users are advised to follow security best practices and adhere to recommended security measures to ensure the integrity and confidentiality of their data.

**9. Performance**

- Performance benchmarks: Performance benchmarks and metrics are provided to evaluate the responsiveness, throughput, and resource utilization of the operating system.

- Optimization techniques used: Various optimization techniques are employed to enhance the performance of the operating system, ensuring efficient use of system resources.

**10. Development and Contribution**

- Guidelines for contributing to the project: Developers are encouraged to contribute to the project by following coding standards, version control practices, and collaboration tools.

- Coding standards: Consistent coding standards and conventions are maintained to promote readability and maintainability of the codebase.

**11. License and Legal**

- License under which the project is distributed: The operating system project is distributed under a specified license, along with associated legal terms and conditions.

- Copyright information: Copyright notices and attribution for third-party components or libraries used in the project are included.

- Third-party dependencies: A list of third-party dependencies and their respective licenses and copyright information is provided for transparency and compliance.

**CODES:**

**Round Robin:**

#include <iostream>

#include <windows.h>

#include <sstream>

#include <string>

#include <vector>

using namespace std;

class RoundRobin

{

private:

int N;

int quantum;

int \*CPU, \*arr\_t, \*turn\_tm, \*wait\_tm, \*ct, \*remainingTime;

float avg\_tat = 0.0, avg\_wt = 0.0;

public:

~RoundRobin()

{

delete[] CPU;

delete[] arr\_t;

delete[] turn\_tm;

delete[] wait\_tm;

delete[] ct;

delete[] remainingTime;

}

void Process(int n, int q, int\* cpu, int\* arrival)

{

N = n;

quantum = q;

CPU = new int[N];

arr\_t = new int[N];

turn\_tm = new int[N];

wait\_tm = new int[N];

ct = new int[N];

remainingTime = new int[N];

for (int j = 0; j < N; j++)

{

CPU[j] = cpu[j];

arr\_t[j] = arrival[j];

remainingTime[j] = cpu[j];

}

}

void Execution()

{

int currentTime = 0;

int completedProcesses = 0;

while (completedProcesses < N)

{

bool done = true;

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

{

if (remainingTime[i] > 0)

{

done = false;

if (remainingTime[i] > quantum)

{

currentTime += quantum;

remainingTime[i] -= quantum;

}

else

{

currentTime += remainingTime[i];

remainingTime[i] = 0;

ct[i] = currentTime;

turn\_tm[i] = ct[i] - arr\_t[i];

wait\_tm[i] = turn\_tm[i] - CPU[i];

completedProcesses++;

}

}

}

if (done)

break;

}

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

{

avg\_tat += turn\_tm[i];

avg\_wt += wait\_tm[i];

}

avg\_tat /= N;

avg\_wt /= N;

}

string Display()

{

ostringstream os;

os << "Process \tCPU burst \tArrival Time \tWaiting Time \tTurn around Time" << endl;

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

{

os << "P" << i << "\t\t" << CPU[i] << "\t\t" << arr\_t[i] << "\t\t" << wait\_tm[i] << "\t\t" << turn\_tm[i] << endl;

}

os << "The average waiting time is =" << avg\_wt << endl;

os << "The average turn around time is =" << avg\_tat << endl;

return os.str();

}

};

RoundRobin R;

HWND Num\_Processes, QuantumTime, ArrivalEdits[10], BurstEdits[10], ArrivalLabels[10], BurstLabels[10], ExecuteButton, ResultLabel;

int numProcesses = 0;

int\* Arrival;

int\* Burst;

int quantum = 0;

int currentProcess = 0;

HINSTANCE g\_hInstance;

LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)

{

switch (msg)

{

case WM\_CREATE:

CreateWindowEx(0, "STATIC", "Number of Processes:", WS\_VISIBLE | WS\_CHILD, 10, 10, 150, 30, hwnd, NULL, g\_hInstance, NULL);

Num\_Processes = CreateWindowEx(0, "EDIT", "", WS\_VISIBLE | WS\_CHILD | WS\_BORDER | ES\_NUMBER, 180, 10, 50, 30, hwnd, NULL, g\_hInstance, NULL);

CreateWindowEx(0, "STATIC", "Quantum Time:", WS\_VISIBLE | WS\_CHILD, 10, 50, 150, 30, hwnd, NULL, g\_hInstance, NULL);

QuantumTime = CreateWindowEx(0, "EDIT", "", WS\_VISIBLE | WS\_CHILD | WS\_BORDER | ES\_NUMBER, 180, 50, 50, 30, hwnd, NULL, g\_hInstance, NULL);

ExecuteButton = CreateWindowEx(0, "BUTTON", "Add Process", WS\_VISIBLE | WS\_CHILD, 10, 90, 150, 30, hwnd, (HMENU)1, g\_hInstance, NULL);

ResultLabel = CreateWindowEx(0, "STATIC", "", WS\_VISIBLE | WS\_CHILD | WS\_BORDER, 10, 400, 400, 200, hwnd, NULL, g\_hInstance, NULL);

break;

case WM\_COMMAND:

if (LOWORD(wParam) == 1)

{

if (currentProcess == 0)

{

char buffer[256];

GetWindowText(Num\_Processes, buffer, sizeof(buffer));

numProcesses = atoi(buffer);

GetWindowText(QuantumTime, buffer, sizeof(buffer));

quantum = atoi(buffer);

Arrival = new int[numProcesses];

Burst = new int[numProcesses];

}

if (currentProcess < numProcesses)

{

ostringstream os;

os << "Arrival Time P" << currentProcess << ":";

string arrivalText = os.str();

os.str("");

os << "Burst Time P" << currentProcess << ":";

string burstText = os.str();

ArrivalLabels[currentProcess] = CreateWindow("STATIC", arrivalText.c\_str(), WS\_VISIBLE | WS\_CHILD, 10, 130 + currentProcess \* 40, 100, 30, hwnd, NULL, g\_hInstance, NULL);

ArrivalEdits[currentProcess] = CreateWindow("EDIT", "", WS\_VISIBLE | WS\_CHILD | WS\_BORDER | ES\_NUMBER, 120, 130 + currentProcess \* 40, 80, 30, hwnd, NULL, g\_hInstance, NULL);

BurstLabels[currentProcess] = CreateWindow("STATIC", burstText.c\_str(), WS\_VISIBLE | WS\_CHILD, 210, 130 + currentProcess \* 40, 100, 30, hwnd, NULL, g\_hInstance, NULL);

BurstEdits[currentProcess] = CreateWindow("EDIT", "", WS\_VISIBLE | WS\_CHILD | WS\_BORDER | ES\_NUMBER, 320, 130 + currentProcess \* 40, 80, 30, hwnd, NULL, g\_hInstance, NULL);

}

currentProcess++;

if (currentProcess > numProcesses)

{

DestroyWindow(Num\_Processes);

DestroyWindow(QuantumTime);

DestroyWindow(ExecuteButton);

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

{

char arrivalBuffer[256], burstBuffer[256];

GetWindowText(ArrivalEdits[i], arrivalBuffer, sizeof(arrivalBuffer));

Arrival[i] = atoi(arrivalBuffer);

GetWindowText(BurstEdits[i], burstBuffer, sizeof(burstBuffer));

Burst[i] = atoi(burstBuffer);

}

R.Process(numProcesses, quantum, Burst, Arrival);

R.Execution();

R.Display();

string result = R.Display();

SetWindowText(ResultLabel, result.c\_str());

}

}

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

return DefWindowProc(hwnd, msg, wParam, lParam);

}

return 0;

}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow)

{

g\_hInstance = hInstance;

// Register window class

WNDCLASS wc = { 0 };

wc.lpfnWndProc = WndProc;

wc.hInstance = hInstance;

wc.lpszClassName = "RoundRobinClass";

RegisterClass(&wc);

// Create window

HWND hwnd = CreateWindow(wc.lpszClassName, "Round Robin Scheduling", WS\_OVERLAPPEDWINDOW, CW\_USEDEFAULT, CW\_USEDEFAULT, 600, 600, NULL, NULL, hInstance, NULL);

// Show window

ShowWindow(hwnd, nCmdShow);

UpdateWindow(hwnd);

// Message loop

MSG msg;

while (GetMessage(&msg, NULL, 0, 0))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return msg.wParam;

}

**FCFS:**

#include <iostream>

#include <windows.h>

#include <sstream>

#include <string>

using namespace std;

class FCFS

{

private:

int N;

int \*CPU, \*arr\_t, \*turn\_tm, \*wait\_tm, \*ct;

float avg\_tat = 0.0, avg\_wt = 0.0;

public:

~FCFS()

{

delete[] CPU;

delete[] arr\_t;

delete[] turn\_tm;

delete[] wait\_tm;

delete[] ct;

}

void Process(int n, int\* cpu, int\* arrival)

{

N = n;

CPU = new int[N];

arr\_t = new int[N];

turn\_tm = new int[N];

wait\_tm = new int[N];

ct = new int[N];

for (int j = 0; j < N; j++)

{

CPU[j] = cpu[j];

arr\_t[j] = arrival[j];

}

}

void Execution()

{

int sum = arr\_t[0];

for (int k=0; k<N; k++)

{

sum=sum+CPU[k];

ct[k]=sum;

}

for (int l=0; l<N; l++)

{

turn\_tm[l]=ct[l]-arr\_t[l];

avg\_tat=avg\_tat+turn\_tm[l];

}

avg\_tat=avg\_tat/N;

for (int k=0; k<N; k++)

{

wait\_tm[k]=turn\_tm[k]-CPU[k];

avg\_wt=avg\_wt+wait\_tm[k];

}

avg\_wt=avg\_wt/N;

}

string Display()

{

ostringstream os;

os<<"Process \tCPU burst \tArrival Time \tWaiting Time \tTurn around Time" << endl;

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

{

os<<"P"<<i<<"\t\t"<<CPU[i]<<"\t\t"<<arr\_t[i]<<"\t\t"<<wait\_tm[i]<<"\t\t"<<turn\_tm[i]<<endl;

}

os<<"The average waiting time is ="<<avg\_wt<<endl;

os<<"The average turn around time is ="<<avg\_tat<<endl;

return os.str();

}

};

FCFS F;

HWND Num\_Processes,ArrivalEdits[10],BurstEdits[10],ArrivalLabels[10],BurstLabels[10],ExecuteButton,ResultLabel;

int numProcesses=0;

int\* Arrival;

int\* Burst;

int currentProcess=0;

HINSTANCE g\_hInstance;

LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)

{

switch (msg)

{

case WM\_CREATE:

CreateWindowEx(0, "STATIC", "Number of Processes:", WS\_VISIBLE | WS\_CHILD, 10, 10, 150, 30, hwnd, NULL, g\_hInstance, NULL);

Num\_Processes=CreateWindowEx(0, "EDIT", "", WS\_VISIBLE | WS\_CHILD | WS\_BORDER | ES\_NUMBER, 180, 10, 50, 30, hwnd, NULL, g\_hInstance, NULL);

ExecuteButton=CreateWindowEx(0, "BUTTON", "Add Process", WS\_VISIBLE | WS\_CHILD, 10, 50, 150, 30, hwnd, (HMENU)1, g\_hInstance, NULL);

ResultLabel=CreateWindowEx(0, "STATIC", "", WS\_VISIBLE | WS\_CHILD | WS\_BORDER, 10, 400, 400, 200, hwnd, NULL, g\_hInstance, NULL);

break;

case WM\_COMMAND:

if(LOWORD(wParam)==1)

{

if(currentProcess==0)

{

char buffer[256];

GetWindowText(Num\_Processes, buffer, sizeof(buffer));

numProcesses=atoi(buffer);

Arrival=new int[numProcesses];

Burst=new int[numProcesses];

}

if(currentProcess<numProcesses)

{

ostringstream os;

os<<"Arrival Time P"<<currentProcess<<":";

string arrivalText=os.str();

os.str("");

os<<"Burst Time P"<<currentProcess<<":";

string burstText=os.str();

ArrivalLabels[currentProcess]=CreateWindow("STATIC", arrivalText.c\_str(), WS\_VISIBLE | WS\_CHILD, 10, 90 + currentProcess \* 40, 100, 30, hwnd, NULL, g\_hInstance, NULL);

ArrivalEdits[currentProcess]=CreateWindow("EDIT", "", WS\_VISIBLE | WS\_CHILD | WS\_BORDER | ES\_NUMBER, 120, 90 + currentProcess \* 40, 80, 30, hwnd, NULL, g\_hInstance, NULL);

BurstLabels[currentProcess]=CreateWindow("STATIC", burstText.c\_str(), WS\_VISIBLE | WS\_CHILD, 210, 90 + currentProcess \* 40, 100, 30, hwnd, NULL, g\_hInstance, NULL);

BurstEdits[currentProcess]=CreateWindow("EDIT", "", WS\_VISIBLE | WS\_CHILD | WS\_BORDER | ES\_NUMBER, 320, 90 + currentProcess \* 40, 80, 30, hwnd, NULL, g\_hInstance, NULL);

}

currentProcess++;

if(currentProcess>numProcesses)

{

DestroyWindow(Num\_Processes);

DestroyWindow(ExecuteButton);

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

{

char arrivalBuffer[256], burstBuffer[256];

GetWindowText(ArrivalEdits[i], arrivalBuffer, sizeof(arrivalBuffer));

Arrival[i]=atoi(arrivalBuffer);

GetWindowText(BurstEdits[i], burstBuffer, sizeof(burstBuffer));

Burst[i]=atoi(burstBuffer);

}

F.Process(numProcesses, Burst, Arrival);

F.Execution();

F.Display();

string result=F.Display();

SetWindowText(ResultLabel, result.c\_str());

}

}

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

return DefWindowProc(hwnd, msg, wParam, lParam);

}

return 0;

}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow)

{

g\_hInstance = hInstance;

// Register window class

WNDCLASS wc = { 0 };

wc.lpfnWndProc = WndProc;

wc.hInstance = hInstance;

wc.lpszClassName = "FCFSClass";

RegisterClass(&wc);

// Create window

HWND hwnd = CreateWindow(wc.lpszClassName, "FCFS Scheduling", WS\_OVERLAPPEDWINDOW, CW\_USEDEFAULT, CW\_USEDEFAULT, 600, 600, NULL, NULL, hInstance, NULL);

// Show window

ShowWindow(hwnd, nCmdShow);

UpdateWindow(hwnd);

// Message loop

MSG msg;

while (GetMessage(&msg, NULL, 0, 0))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return msg.wParam;

}

**Priority:**

#include<iostream>

#include<windows.h>

#include<sstream>

#include<string>

using namespace std;

class Priority

{

private:

int N;

int \*P, \*CPU, \*arr\_t, \*turn\_tm, \*wait\_tm, \*pr;

float avg\_tat = 0.0, avg\_wt = 0.0;

public:

~Priority()

{

delete[] P;

delete[] CPU;

delete[] arr\_t;

delete[] turn\_tm;

delete[] wait\_tm;

delete[] pr;

}

void Process(int n, int\* Pro, int\* cpu, int\* arrival, int\* priorties)

{

N = n;

P = new int[N];

CPU = new int[N];

arr\_t = new int[N];

turn\_tm = new int[N];

wait\_tm = new int[N];

pr = new int[N];

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

{

P[i] = Pro[i];

CPU[i] = cpu[i];

arr\_t[i] = arrival[i];

pr[i] = priorties[i];

}

}

void Execution()

{

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

{

int position = i;

for (int j = i + 1; j < N; j++)

{

if (pr[j] < pr[position])

{

position = j;

}

}

swap(P[i], P[position]);

swap(CPU[i], CPU[position]);

swap(arr\_t[i], arr\_t[position]);

swap(pr[i], pr[position]);

}

wait\_tm[0] = 0;

turn\_tm[0] = CPU[0];

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

{

wait\_tm[i] = wait\_tm[i - 1] + CPU[i - 1];

turn\_tm[i] = wait\_tm[i] + CPU[i];

}

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

{

avg\_wt += wait\_tm[i];

avg\_tat += turn\_tm[i];

}

avg\_wt /= N;

avg\_tat /= N;

}

string Display()

{

ostringstream os;

os << "Process\tCPU burst\tArrival Time\tWaiting Time\tTurn around\tPriority" << endl;

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

{

os << "P" << P[i] << "\t" << CPU[i]<<"\t\t" << arr\_t[i] << "\t\t" << wait\_tm[i] << "\t\t" << turn\_tm[i] << "\t\t" << pr[i] << endl;

}

os << "The average waiting time is =" << avg\_wt << endl;

os << "The average turn around time is =" << avg\_tat << endl;

return os.str();

}

};

Priority PP;

HWND Num\_Processes, PriorityEdit[10], Prioritylabel[10], ProcessEdit[10], Processlabel[10], ArrivalEdit[10], BustEdit[10], Arrivallabel[10], Bustlabel[10], ExuecuteButton, Resultlabel;

int num\_process = 0, current\_process = 0;

int\* Arrival;

int\* Bust;

int\* Processnum;

int\* Proit;

HINSTANCE g\_hInstance;

LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)

{

switch (msg)

{

case WM\_CREATE:

CreateWindowEx(0, "STATIC", "Enter Number of Process: ", WS\_VISIBLE | WS\_CHILD, 10, 10, 150, 30, hwnd, NULL, g\_hInstance, NULL);

Num\_Processes = CreateWindowEx(0, "EDIT", "", WS\_VISIBLE | WS\_CHILD | WS\_BORDER | ES\_NUMBER, 180, 10, 50, 30, hwnd, NULL, g\_hInstance, NULL);

ExuecuteButton = CreateWindowEx(0, "BUTTON", "Add Process", WS\_VISIBLE | WS\_CHILD, 250, 10, 100, 30, hwnd, (HMENU)1, g\_hInstance, NULL);

Resultlabel = CreateWindowEx(0, "STATIC", "", WS\_VISIBLE | WS\_CHILD | WS\_BORDER, 10, 400, 560, 200, hwnd, NULL, g\_hInstance, NULL);

break;

case WM\_COMMAND:

if (LOWORD(wParam) == 1)

{

if (current\_process == 0)

{

char buffer[200];

GetWindowText(Num\_Processes, buffer, sizeof(buffer));

num\_process = atoi(buffer);

Processnum = new int[num\_process];

Proit = new int[num\_process];

Arrival = new int[num\_process];

Bust = new int[num\_process];

int yPos = 50;

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

{

ostringstream oo;

oo << "P" << (i + 1);

string processText = oo.str();

Processlabel[i] = CreateWindow("STATIC", processText.c\_str(), WS\_VISIBLE | WS\_CHILD, 10, yPos, 50, 30, hwnd, NULL, g\_hInstance, NULL);

ProcessEdit[i] = CreateWindow("EDIT", "", WS\_VISIBLE | WS\_CHILD | WS\_BORDER | ES\_NUMBER, 70, yPos, 50, 30, hwnd, NULL, g\_hInstance, NULL);

Arrivallabel[i] = CreateWindow("STATIC", "Arrival Time:", WS\_VISIBLE | WS\_CHILD, 130, yPos, 80, 30, hwnd, NULL, g\_hInstance, NULL);

ArrivalEdit[i] = CreateWindow("EDIT", "", WS\_VISIBLE | WS\_CHILD | WS\_BORDER | ES\_NUMBER, 220, yPos, 50, 30, hwnd, NULL, g\_hInstance, NULL);

Bustlabel[i] = CreateWindow("STATIC", "Burst Time:", WS\_VISIBLE | WS\_CHILD, 280, yPos, 80, 30, hwnd, NULL, g\_hInstance, NULL);

BustEdit[i] = CreateWindow("EDIT", "", WS\_VISIBLE | WS\_CHILD | WS\_BORDER | ES\_NUMBER, 370, yPos, 50, 30, hwnd, NULL, g\_hInstance, NULL);

Prioritylabel[i] = CreateWindow("STATIC", "Priority:", WS\_VISIBLE | WS\_CHILD, 430, yPos, 80, 30, hwnd, NULL, g\_hInstance, NULL);

PriorityEdit[i] = CreateWindow("EDIT", "", WS\_VISIBLE | WS\_CHILD | WS\_BORDER | ES\_NUMBER, 520, yPos, 50, 30, hwnd, NULL, g\_hInstance, NULL);

yPos += 40;

}

ExuecuteButton = CreateWindowEx(0, "BUTTON", "Run Algo", WS\_VISIBLE | WS\_CHILD, 430, yPos, 100, 30, hwnd, (HMENU)2, g\_hInstance, NULL);

}

}

else if (LOWORD(wParam) == 2)

{

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

{

char arrivalbuffer[256], bustbuffer[256], processnumbuffer[256], proibuffer[256];

GetWindowText(ProcessEdit[i], processnumbuffer, sizeof(processnumbuffer));

Processnum[i] = atoi(processnumbuffer);

GetWindowText(ArrivalEdit[i], arrivalbuffer, sizeof(arrivalbuffer));

Arrival[i] = atoi(arrivalbuffer);

GetWindowText(BustEdit[i], bustbuffer, sizeof(bustbuffer));

Bust[i] = atoi(bustbuffer);

GetWindowText(PriorityEdit[i], proibuffer, sizeof(proibuffer));

Proit[i] = atoi(proibuffer);

}

PP.Process(num\_process, Processnum, Bust, Arrival, Proit);

PP.Execution();

string result = PP.Display();

SetWindowText(Resultlabel, result.c\_str());

}

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

return DefWindowProc(hwnd, msg, wParam, lParam);

}

return 0;

}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow)

{

g\_hInstance = hInstance;

// Register window class

WNDCLASS wc = { 0 };

wc.lpfnWndProc = WndProc;

wc.hInstance = hInstance;

wc.lpszClassName = "PriorityClass";

RegisterClass(&wc);

// Create window

HWND hwnd = CreateWindow(wc.lpszClassName, "Priority Scheduling", WS\_OVERLAPPEDWINDOW, CW\_USEDEFAULT, CW\_USEDEFAULT, 600, 600, NULL, NULL, hInstance, NULL);

// Show window

ShowWindow(hwnd, nCmdShow);

UpdateWindow(hwnd);

// Message loop

MSG msg;

while (GetMessage(&msg, NULL, 0, 0))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return msg.wParam;

}

**SJF:**

#include <iostream>

#include <windows.h>

#include <sstream>

#include <string>

using namespace std;

class SJF

{

private:

int N;

int \*P, \*CPU, \*arr\_t, \*turn\_tm, \*wait\_tm;

float avg\_tat=0.0, avg\_wt=0.0;

public:

~SJF()

{

delete[] P;

delete[] CPU;

delete[] arr\_t;

delete[] turn\_tm;

delete[] wait\_tm;

}

void Process(int n, int\*Pro, int\* cpu, int\* arrival)

{

N=n;

P=new int[N];

CPU=new int[N];

arr\_t=new int[N];

turn\_tm=new int[N];

wait\_tm=new int[N];

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

{

//P[i]=i;

P[i]=Pro[i];

CPU[i]=cpu[i];

arr\_t[i]=arrival[i];

}

}

void Execution()

{

for (int i=0; i<N-1; i++)

{

for (int j=0; j<N-i-1; j++)

{

if (CPU[j]>CPU[j + 1])

{

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

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

swap(arr\_t[j],arr\_t[j + 1]);

}

else if (CPU[j]==CPU[j + 1] && arr\_t[j]>arr\_t[j + 1])

{

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

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

swap(arr\_t[j],arr\_t[j + 1]);

}

}

}

wait\_tm[0]=0;

turn\_tm[0]=CPU[0];

for (int k=1; k<N; k++)

{

wait\_tm[k]=wait\_tm[k - 1]+CPU[k - 1];

turn\_tm[k]=wait\_tm[k]+CPU[k];

}

for (int k=0; k<N; k++)

{

avg\_wt=avg\_wt+wait\_tm[k];

avg\_tat=avg\_tat+turn\_tm[k];

}

avg\_wt=avg\_wt / N;

avg\_tat=avg\_tat / N;

}

string Display()

{

ostringstream os;

os<<"Process \tCPU burst \tArrival Time \tWaiting Time \tTurn around Time" << endl;

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

{

os<<"P"<<i<<"\t\t"<<CPU[i]<<"\t\t"<<arr\_t[i]<<"\t\t"<<wait\_tm[i]<<"\t\t"<<turn\_tm[i]<<endl;

}

os<<"The average waiting time is ="<<avg\_wt<<endl;

os<<"The average turn around time is ="<<avg\_tat<<endl;

return os.str();

}

};

SJF S;

HWND Num\_Processes,ProcessEdit[10],Processlabel[10],ArrivalEdit[10],BustEdit[10],Arrivallabel[10],Bustlabel[10],ExuecuteButton,Resultlabel;

int num\_process=0,current\_process=0;

int\* Arrival;

int\* Bust;

int\* Processnum;

HINSTANCE g\_hInstance;

LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)

{

switch(msg)

{

case WM\_CREATE:

CreateWindowEx(0, "STATIC", "Enter Number of Process: ", WS\_VISIBLE | WS\_CHILD, 10, 10, 150, 30, hwnd, NULL, g\_hInstance, NULL);

Num\_Processes=CreateWindowEx(0, "EDIT", "", WS\_VISIBLE | WS\_CHILD | WS\_BORDER | ES\_NUMBER, 180, 10, 50, 30, hwnd, NULL, g\_hInstance, NULL);

ExuecuteButton=CreateWindowEx(0, "BUTTON", "Add Process", WS\_VISIBLE | WS\_CHILD, 10, 50, 150, 30, hwnd, (HMENU)1, g\_hInstance, NULL);

Resultlabel=CreateWindowEx(0, "STATIC", "", WS\_VISIBLE | WS\_CHILD | WS\_BORDER, 10, 400, 400, 200, hwnd, NULL, g\_hInstance, NULL);

break;

case WM\_COMMAND:

if(LOWORD(wParam)==1)

{

if (current\_process==0)

{

char buffer[200];

GetWindowText(Num\_Processes,buffer,sizeof(buffer));

num\_process=atoi(buffer);

Arrival=new int[num\_process];

Bust=new int[num\_process];

}

if(current\_process<num\_process)

{

ostringstream oo;

oo<<"Process Number : P";

string processnumberText=oo.str();

oo<<"Arrival time of P"<<current\_process<<":";

string arrivalText=oo.str();

oo.str("");

oo<<"CPU Burst of P"<<current\_process<<":";

string burstText=oo.str();

Processlabel[current\_process]=CreateWindow("STATIC", processnumberText.c\_str(), WS\_VISIBLE | WS\_CHILD, 110, 90 + current\_process \* 40, 100, 30, hwnd, NULL, g\_hInstance, NULL);

ProcessEdit[current\_process]=CreateWindow("EDIT", "", WS\_VISIBLE | WS\_CHILD | WS\_BORDER | ES\_NUMBER, 120, 90 + current\_process \* 40, 80, 30, hwnd, NULL, g\_hInstance, NULL);

Arrivallabel[current\_process]=CreateWindow("STATIC", arrivalText.c\_str(), WS\_VISIBLE | WS\_CHILD, 10, 90 + current\_process \* 40, 100, 30, hwnd, NULL, g\_hInstance, NULL);

ArrivalEdit[current\_process]=CreateWindow("EDIT", "", WS\_VISIBLE | WS\_CHILD | WS\_BORDER | ES\_NUMBER, 120, 90 + current\_process \* 40, 80, 30, hwnd, NULL, g\_hInstance, NULL);

Bustlabel[current\_process]=CreateWindow("STATIC", burstText.c\_str(), WS\_VISIBLE | WS\_CHILD, 210, 90 + current\_process \* 40, 100, 30, hwnd, NULL, g\_hInstance, NULL);

BustEdit[current\_process]=CreateWindow("EDIT", "", WS\_VISIBLE | WS\_CHILD | WS\_BORDER | ES\_NUMBER, 320, 90 + current\_process \* 40, 80, 30, hwnd, NULL, g\_hInstance, NULL);

}

current\_process++;

if(current\_process>num\_process)

{

DestroyWindow(Num\_Processes);

DestroyWindow(ExuecuteButton);

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

{

char arrivalbuffer[256],bustbuffer[256],processnumbuffer[256];

GetWindowText(ProcessEdit[i] ,processnumbuffer,processnumbuffer[256]);

Processnum[i]=atoi(processnumbuffer);

GetWindowText(ArrivalEdit[i],arrivalbuffer,sizeof(arrivalbuffer));

Arrival[i]=atoi(arrivalbuffer);

GetWindowText(BustEdit[i],bustbuffer,sizeof(bustbuffer));

Bust[i]=atoi(bustbuffer);

}

S.Process(num\_process, Processnum,Bust, Arrival);

S.Execution();

string result=S.Display();

SetWindowText(Resultlabel, result.c\_str());

}

}

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

return DefWindowProc(hwnd, msg, wParam, lParam);

}

return 0;

}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow)

{

g\_hInstance = hInstance;

// Register window class

WNDCLASS wc = { 0 };

wc.lpfnWndProc = WndProc;

wc.hInstance = hInstance;

wc.lpszClassName = "FCFSClass";

RegisterClass(&wc);

// Create window

HWND hwnd = CreateWindow(wc.lpszClassName, "SJF Scheduling", WS\_OVERLAPPEDWINDOW, CW\_USEDEFAULT, CW\_USEDEFAULT, 600, 600, NULL, NULL, hInstance, NULL);

// Show window

ShowWindow(hwnd, nCmdShow);

UpdateWindow(hwnd);

// Message loop

MSG msg;

while (GetMessage(&msg, NULL, 0, 0))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return msg.wParam;

}

**How to run?**

* Compile the codes in c++ version 6.3. It might give errors in old versions of c++ .
* Execute the algorithm.
* GUI interface will be opened.
* Provide the number of processes demanded in every algorithm and click on add process button.
* Give the arrival time and burst time to every process in each algorithm.
* Let the code be executed.
* Waiting time and turnaround time will be displayed on the GUI interface after some calculations.