Activity based

Project Report on

Operating System

Submitted to Vishwakarma University, Pune

Under the Initiative of

Contemporary Curriculum, Pedagogy, and Practice (C2P2)

By

Atharva Shevate

SRN No : 202201727

Roll No : 02

Div : E

Third Year Engineering

Project Report : 02

Department of Computer Engineering

Faculty of Science and Technology

Academic Year

2024-2025

CODE :-

```
#include <windows.h>
#include <commctrl.h>
#include <stdio.h>

// Global variables
HINSTANCE hInst;
LPCSTR szTitle = "CPU Scheduling Algorithms GUI"; // Window title
LPCSTR szWindowClass = "CPU_SCHEDULING_CLASS";    // Window class name

// Control IDs
#define IDC_PROCESS_ID 101
#define IDC_BURST_TIME 102
#define IDC_QUANTUM_TIME 103
#define IDC_RESULTS 104
#define IDC_ALGORITHM_FCFS 105
#define IDC_ALGORITHM_SJF 106
#define IDC_ALGORITHM_RR 107
#define IDC_RUN_BUTTON 108

// Function declarations
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
void CalculateScheduling(HWND hWnd);

// WinMain: Entry point for a Windows application
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR
lpCmdLine, int nCmdShow) {
    hInst = hInstance; // Store instance handle in global variable

    // Register the window class
```

```
WNDCLASSEX wcex;
wcex.cbSize = sizeof(WNDCLASSEX);
wcex.style = CS_HREDRAW | CS_VREDRAW;
wcex.lpfnWndProc = WndProc;
wcex.cbClsExtra = 0;
wcex.cbWndExtra = 0;
wcex.hInstance = hInstance;
wcex.hIcon = LoadIcon(NULL, IDI_APPLICATION);
wcex.hCursor = LoadCursor(NULL, IDC_ARROW);
wcex.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
wcex.lpszMenuName = NULL;
wcex.lpszClassName = szWindowClass;
wcex.hIconSm = LoadIcon(NULL, IDI_APPLICATION);

if (!RegisterClassEx(&wcex)) {
    return FALSE;
}

// Create the window
HWND hWnd = CreateWindow(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
                         CW_USEDEFAULT, 0, 600, 400, NULL, NULL, hInstance, NULL);
if (!hWnd) {
    return FALSE;
}

ShowWindow(hWnd, nCmdShow);
UpdateWindow(hWnd);

// Main message loop
MSG msg;
while (GetMessage(&msg, NULL, 0, 0)) {
    TranslateMessage(&msg);
    DispatchMessage(&msg);
}
```

```
    return (int)msg.wParam;
}

// Window procedure to process messages
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam) {
    static HWND hAlgorithmGroup, hBurstTimeEdit, hProcessIDEdit, hResultDisplay;
    static HWND hQuantumTimeEdit;

    switch (message) {
        case WM_CREATE: {
            // Create a group box for algorithm selection
            hAlgorithmGroup = CreateWindow("BUTTON", "Select Scheduling Algorithm", WS_VISIBLE | WS_CHILD | BS_GROUPBOX,
                                           20, 20, 250, 120, hWnd, NULL, hInst, NULL);

            // Create radio buttons for algorithm selection
            CreateWindow("BUTTON", "FCFS", WS_VISIBLE | WS_CHILD | BS_RADIOBUTTON,
                         30, 40, 100, 25, hWnd, (HMENU)IDC_ALGORITHM_FCFS, hInst, NULL);
            CreateWindow("BUTTON", "SJF", WS_VISIBLE | WS_CHILD | BS_RADIOBUTTON,
                         30, 70, 100, 25, hWnd, (HMENU)IDC_ALGORITHM_SJF, hInst, NULL);
            CreateWindow("BUTTON", "Round Robin", WS_VISIBLE | WS_CHILD | BS_RADIOBUTTON,
                         30, 100, 100, 25, hWnd, (HMENU)IDC_ALGORITHM_RR, hInst, NULL);

            // Create input fields for process ID and burst time
            CreateWindow("STATIC", "Process ID:", WS_VISIBLE | WS_CHILD, 300, 40, 80, 25, hWnd, NULL, hInst, NULL);
            hProcessIDEdit = CreateWindow("EDIT", "", WS_VISIBLE | WS_CHILD | WS_BORDER,
                                          390, 40, 100, 25, hWnd, (HMENU)IDC_PROCESS_ID, hInst, NULL);
```

```
        CreateWindow("STATIC", "Burst Time:", WS_VISIBLE | WS_CHILD, 300, 80, 80, 25,
hWnd, NULL, hInst, NULL);
        hBurstTimeEdit = CreateWindow("EDIT", "", WS_VISIBLE | WS_CHILD |
WS_BORDER,
                                      390, 80, 100, 25, hWnd, (HMENU)IDC_BURST_TIME, hInst,
NULL);

        // Create input field for quantum time (for Round Robin)
        CreateWindow("STATIC", "Quantum Time:", WS_VISIBLE | WS_CHILD, 300, 120,
80, 25, hWnd, NULL, hInst, NULL);
        hQuantumTimeEdit = CreateWindow("EDIT", "", WS_VISIBLE | WS_CHILD |
WS_BORDER,
                                        390, 120, 100, 25, hWnd, (HMENU)IDC_QUANTUM_TIME,
hInst, NULL);

        // Create a button to run the scheduling algorithm
        CreateWindow("BUTTON", "Run Scheduling Algorithm",
                WS_TABSTOP | WS_VISIBLE | WS_CHILD | BS_DEFPUSHBUTTON,
                300, 160, 190, 40, hWnd, (HMENU)IDC_RUN_BUTTON, hInst, NULL);

        // Create a result display area
        CreateWindow("STATIC", "Results:", WS_VISIBLE | WS_CHILD, 20, 220, 80, 25,
hWnd, NULL, hInst, NULL);
        hResultDisplay = CreateWindow("EDIT", "", WS_VISIBLE | WS_CHILD |
WS_BORDER | ES_MULTILINE | ES_READONLY,
                                      20, 240, 550, 120, hWnd, (HMENU)IDC_RESULTS, hInst,
NULL);

        break;
    }
    case WM_COMMAND: {
        switch (LOWORD(wParam)) {
            case IDC_RUN_BUTTON: // Run button clicked
                CalculateScheduling(hWnd);
                break;
```

```
            }
            break;
        }
        case WM_PAINT: {
            PAINTSTRUCT ps;
            HDC hdc = BeginPaint(hWnd, &ps);
            FillRect(hdc, &ps.rcPaint, (HBRUSH)(COLOR_WINDOW + 1));
            EndPaint(hWnd, &ps);
            break;
        }
        case WM_DESTROY: {
            PostQuitMessage(0);
            break;
        }
        default: {
            return DefWindowProc(hWnd, message, wParam, lParam);
        }
    }
    return 0;
}

// Function to calculate scheduling
void CalculateScheduling(HWND hWnd) {
    char processID[10];
    char burstTime[10];
    char quantumTime[10];
    char result[512];

    // Get input from the user
    GetWindowText(GetDlgItem(hWnd, IDC_PROCESS_ID), processID, sizeof(processID));
// Get Process ID
    GetWindowText(GetDlgItem(hWnd, IDC_BURST_TIME), burstTime, sizeof(burstTime));
// Get Burst Time
    GetWindowText(GetDlgItem(hWnd, IDC_QUANTUM_TIME), quantumTime,
sizeof(quantumTime)); // Get Quantum Time
```

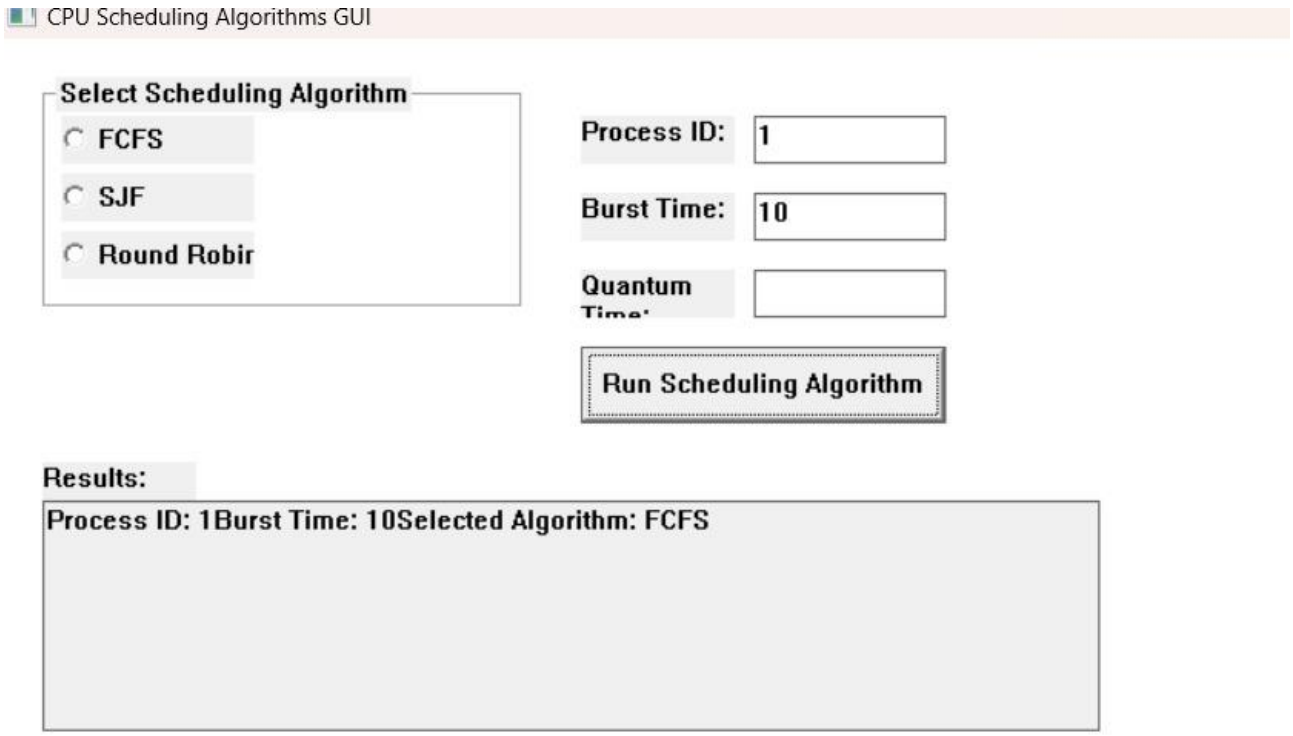
```
    // Determine selected algorithm
    int algorithm = 0; // 0 = FCFS, 1 = SJF, 2 = Round Robin
    if (IsDlgButtonChecked(hWnd, IDC_ALGORITHM_FCFS)) algorithm = 0;
    else if (IsDlgButtonChecked(hWnd, IDC_ALGORITHM_SJF)) algorithm = 1;
    else if (IsDlgButtonChecked(hWnd, IDC_ALGORITHM_RR)) algorithm = 2;

    // For demonstration, create a result string based on the selected algorithm
    sprintf(result, "Process ID: %s\nBurst Time: %s\n", processID, burstTime);

    if (algorithm == 0) {
        strcat(result, "Selected Algorithm: FCFS\n");
    } else if (algorithm == 1) {
        strcat(result, "Selected Algorithm: SJF\n");
    } else {
        strcat(result, "Selected Algorithm: Round Robin\n");
        strcat(result, "Quantum Time: ");
        strcat(result, quantumTime);
        strcat(result, "\n");
    }

    // Display results
    SetWindowText(GetDlgItem(hWnd, IDC_RESULTS), result); // Display results in the result area
}
```

OUTPUT:-


CPU Scheduling Algorithms GUI
Select Scheduling Algorithm
FCFS
SJF
Round Robir
Process ID:
1
Burst Time:
10
Quantum
Run Scheduling Algorithm
Results:
Process ID: 1Burst Time: 10Selected Algorithm: FCFS