

Operating System Project

instructor:

Dr. Ahmed Selim

TA: Eng. Gehad

Project Title:

OS Scheduler Simulation

Team Members

1. Mariam Mohamed - ID: 692300140
2. Maram Mohamed - ID: 692300130
3. Rawan Osama - ID: 692300455
4. Amira Saad - ID: 692300351

introduction:

< In this project, we simulated the role of the CPU scheduler in an operating system by designing and implementing a program that schedules processes using several well-known algorithms. We focused on performance analysis and comparing algorithms under the same conditions.

< The program was developed by using Python language and PyQt5 library

Scheduling Algorithms Implemented:

FCFS First Come First Serve Runs processes in order of arrival.

Round Robin (RR) Each process gets a fixed time slice (quantum).

(Non-Preemptive) Highest Priority First Chooses the process with the highest priority.

SRTF Shortest Remaining Time First Always selects the process with the least remaining time

{SRTF} performed best overall, with the lowest average waiting and turnaround times.

{HPF} came next, as it prioritizes shorter jobs but may cause starvation for low-priority processes

{RR}. showed fair scheduling, but its efficiency depends on the chosen time quantum.

{FCFS} had the highest waiting time, especially when early processes had long burst times

Operating System Project

Summary

This Python script simulates a process generator for an Operating System project.

It utilizes inter-process communication (IPC) with message queues, and signal handling to coordinate the timing and generation of processes.

Key components of the code:

- Reads process information from a file.
- Sends process data to a scheduler at specific time intervals.
- Handles SIGINT to clean up resources.
- Uses a forking method to launch a clock and scheduler program.

It's designed to simulate realistic behavior in a basic OS environment.

Code

```
import sys
import random
import numpy as np
from PyQt5.QtWidgets import (
    QApplication, QMainWindow, QWidget, QVBoxLayout, QHBoxLayout, QLabel,
    QPushButton, QComboBox, QTableWidget, QTableWidgetItem, QSpinBox,
```

Operating System Project

```
        QMessageBox, QFileDialog, QTabWidget, QTextEdit
    )
from PyQt5.QtCore import Qt
from PyQt5.QtGui import QFont

class ProcessGenerator(QWidget):
    def __init__(self):
        super().__init__()
        self.init_ui()

    def init_ui(self):
        layout = QVBoxLayout()

        # Title
        title = QLabel("Process Generator Module")
        title.setFont(QFont('Arial', 14, QFont.Bold))
        title.setAlignment(Qt.AlignCenter)
        layout.addWidget(title)

        # Parameters
        params_layout = QHBoxLayout()

        left_params = QVBoxLayout()
        self.num_processes = QSpinBox()
        self.num_processes.setRange(1, 100)
        self.num_processes.setValue(10)
        left_params.addWidget(QLabel("Number of Processes:"))
        left_params.addWidget(self.num_processes)

        self.arrival_mean = QSpinBox()
        self.arrival_mean.setRange(0, 100)
        self.arrival_mean.setValue(5)
        left_params.addWidget(QLabel("Arrival Time Mean:"))
        left_params.addWidget(self.arrival_mean)

        right_params = QVBoxLayout()
        self.burst_mean = QSpinBox()
        self.burst_mean.setRange(1, 100)
        self.burst_mean.setValue(10)
        right_params.addWidget(QLabel("Burst Time Mean:"))
        right_params.addWidget(self.burst_mean)

        self.priority_lambda = QSpinBox()
        self.priority_lambda.setRange(1, 10)
        self.priority_lambda.setValue(3)
        right_params.addWidget(QLabel("Priority Lambda:"))
        right_params.addWidget(self.priority_lambda)

        params_layout.addLayout(left_params)
```

Operating System Project

```
params_layout.addLayout(right_params)
layout.addLayout(params_layout)

# Buttons
btn_layout = QHBoxLayout()
self.generate_btn = QPushButton("Generate Processes")
self.generate_btn.clicked.connect(self.generate_processes)
self.save_btn = QPushButton("Save to File")
self.save_btn.clicked.connect(self.save_to_file)
btn_layout.addWidget(self.generate_btn)
btn_layout.addWidget(self.save_btn)
layout.addLayout(btn_layout)

# Results Table
self.table = QTableWidgetItem()
self.table.setColumnCount(4)
self.table.setHorizontalHeaderLabels(["PID", "Arrival", "Burst", "Priority"])
layout.addWidget(self.table)

self.setLayout(layout)

def generate_processes(self):
    n = self.num_processes.value()
    arrival_mean = self.arrival_mean.value()
    burst_mean = self.burst_mean.value()
    priority_lambda = self.priority_lambda.value()

    # Generate arrival times (normal distribution)
    arrival_times = np.abs(np.random.normal(arrival_mean, arrival_mean/2, n)).astype(int)
    arrival_times = np.cumsum(arrival_times)

    # Generate burst times (normal distribution)
    burst_times = np.abs(np.random.normal(burst_mean, burst_mean/2, n)).astype(int)
    burst_times = np.where(burst_times < 1, 1, burst_times)

    # Generate priorities (Poisson distribution)
    priorities = np.random.poisson(priority_lambda, n)
    priorities = np.where(priorities < 1, 1, priorities)

    # Populate table
    self.table.setRowCount(n)
    for i in range(n):
        self.table.setItem(i, 0, QTableWidgetItem(f"P{i+1}"))
        self.table.setItem(i, 1, QTableWidgetItem(str(arrival_times[i])))
        self.table.setItem(i, 2, QTableWidgetItem(str(burst_times[i])))
        self.table.setItem(i, 3, QTableWidgetItem(str(priorities[i])))

def save_to_file(self):
    filename, _ = QFileDialog.getSaveFileName(self, "Save Processes", "", "Text Files
```

Operating System Project

```
(* .txt)")
    if filename:
        with open(filename, 'w') as f:
            f.write("PID,Arrival,Burst,Priority\n")
            for row in range(self.table.rowCount()):
                pid = self.table.item(row, 0).text()
                arrival = self.table.item(row, 1).text()
                burst = self.table.item(row, 2).text()
                priority = self.table.item(row, 3).text()
                f.write(f"{pid},{arrival},{burst},{priority}\n")
    QMessageBox.information(self, "Success", "Processes saved to file successfully!")
```

Operating System Project

The screenshot shows the 'Process Generator Module' of the CPU Scheduling Simulator. It features input fields for 'Number of Processes' (set to 10), 'Burst Time Mean' (set to 10), 'Arrival Time Mean' (set to 5), and 'Priority Lambda' (set to 3). Below these are buttons for 'Generate Processes' and 'Save to File'. A table displays the generated processes:

	PID	Arrival	Burst	Priority
1	P1	3	3	3
2	P2	5	2	2
3	P3	11	1	4
4	P4	13	4	3
5	P5	17	1	1
6	P6	25	1	3
7	P7	28	3	3
8	P8	30	2	1
9	P9	36	20	3
10	P10	42	12	3

Below the table, a code editor shows the following Python code:

```
30 self.num_processes = QSpinBox()
31 self.num_processes.setRange(1, 100)
32 self.num_processes.setValue(10)
33 left_params.addWidget(QLabel("Number of Processes:"))
```

The terminal at the bottom shows the command: `PS C:\Users\hp> & C:/Users/hp/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/hp/OneDrive/Documents/import sys.py"`

The screenshot shows the 'CPU Scheduler' module of the CPU Scheduling Simulator. It features a 'Load Processes' button, a file path input field (set to 'C:/Users/hp/process project.txt'), and a 'Scheduling Algorithm' dropdown menu (set to 'First Come First Serve (FCFS)'). A green 'Run Scheduling' button is prominently displayed. Below these are tabs for 'Input Processes', 'Scheduling Results', 'Gantt Chart', and 'Statistics'. The 'Scheduling Results' tab is active, showing a table with the following data:

	PID	Arrival	Burst	Priority	Start	Finish	Waiting
1	P1	3	3	3	3	6	0
2	P2	5	2	2	6	8	1
3	P3	11	1	4	11	12	0
4	P4	13	4	3	13	17	0
5	P5	17	1	1	17	18	0
6	P6	25	1	3	25	26	0
7	P7	28	3	3	28	31	0
8	P8	30	2	1	31	33	1
9	P9	36	20	3	36	56	0
10	P10	42	12	3	56	68	14

Below the table, the same code editor from the previous screenshot is visible, showing the same Python code.

The terminal at the bottom shows the same command as the previous screenshot.

Operating System Project

The screenshot displays a Python IDE with a CPU Scheduling Simulator application running. The simulator window is titled "CPU Scheduling Simulator" and has two tabs: "Process Generator" and "CPU Scheduler". The "Process Generator" tab is active, showing a "Load Processes" button and a file path "C:/Users/hp/processproject.txt". Below this, the "Scheduling Algorithm" is set to "Round Robin (RR)" and the "Time Quantum" is set to 4. A green "Run Scheduling" button is visible.

The "CPU Scheduler" tab is also visible, showing a table of input processes. The table has four columns: "PID", "Arrival", "Burst", and "Priority". The data is as follows:

PID	Arrival	Burst	Priority
1 P1	3	3	3
2 P2	5	2	2
3 P3	11	1	4
4 P4	13	4	3
5 P5	17	1	1
6 P6	25	1	3
7 P7	28	3	3
8 P8	30	2	1
9 P9	36	20	3
10 P10	42	12	3

The IDE shows the Python code for the simulator. The code is as follows:

```
30 self.num_processes = QSpinBox()
31 self.num_processes.setRange(1, 100)
32 self.num_processes.setValue(10)
33 left_params.addWidget(QLabel("Number of Processes:"))
```

The terminal output shows the command to run the simulator:

```
PS C:\Users\hp> & C:/Users/hp/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/hp/OneDrive/Documents/import sys.py"
```

Operating System Project

The screenshot displays a Windows IDE environment with a project named "proc". The main window is titled "CPU Scheduling Simulator" and contains the following elements:

- Process Generator** tab: A text box labeled "Load Processes" containing the file path "C:/Users/hp/process project.txt".
- CPU Scheduler** tab: A dropdown menu for "Scheduling Algorithm:" set to "First Come First Serve (FCFS)".
- Run Scheduling**: A large green button.
- Input Processes** tab: A section titled "Gantt Chart:" displaying the following process execution times:
 - [P1] 3-6 ==
 - [P2] 6-8 ==
 - [P3] 11-12 =
 - [P4] 13-17 ====
 - [P5] 17-18 =
 - [P6] 25-26 =
 - [P7] 28-31 ===
 - [P8] 31-33 ==
 - [P9] 36-56 =====
 - [P10] 56-68 =====
- Scheduling Results**, **Gantt Chart**, and **Statistics** tabs are also visible.

The background shows the source code for "import sys.py" in a Python file editor. The code includes a class definition for a scheduling simulator, with methods for loading processes and scheduling them using the First Come First Serve (FCFS) algorithm. The code is as follows:

```
30 self.num_processes = QSpinBox()
31 self.num_processes.setRange(1, 100)
32 self.num_processes.setValue(10)
33 left_params.addWidget(QLabel("Number of Processes:"))
```

The terminal at the bottom shows the command to run the application:

```
PS C:\Users\hp> & C:/Users/hp/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/hp/OneDrive/Documents/import sys.py"
```


Operating System Project

The image shows a Python IDE interface with a CPU Scheduling Simulator window and a code editor.

CPU Scheduling Simulator - Process Generator Module

Number of Processes: 10
Burst Time Mean: 10
Arrival Time Mean: 5
Priority Lambda: 3

Buttons: Generate Processes, Save to File

PID	Arrival	Burst	Priority
-----	---------	-------	----------

Code Editor (app.py):

```
30 self.num_processes = QSpinBox()
31 self.num_processes.setRange(1, 100)
32 self.num_processes.setValue(10)
33 left_params.addWidget(QLabel("Number of Processes:"))
```

Terminal:

```
PS C:\Users\hp> & C:/Users/hp/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/hp/OneDrive/Documents/import sys.py"
```

Status Bar: Ln 1, Col 1 Spaces: 4 UTF-8 CRLF {} Python 3.13.1 64-bit

Operating System Project

The image shows a Python IDE with a window titled "CPU Scheduling Simulator". The window has two tabs: "Process Generator" and "CPU Scheduler". The "CPU Scheduler" tab is active, displaying the simulator's interface. The interface includes a "Load Processes" button, a "No file loaded" status, a "Scheduling Algorithm:" dropdown menu set to "First Come First Serve (FCFS)", a green "Run Scheduling" button, and a table with columns "PID", "Arrival", "Burst", and "Priority". The table is currently empty.

The source code for the simulator is visible in the background, showing the following Python code:

```
30 self.num_processes = QSpinBox()
31 self.num_processes.setRange(1, 100)
32 self.num_processes.setValue(10)
33 left_params.addWidget(QLabel("Number of Processes:"))
```

The IDE's terminal at the bottom shows the command used to run the application:

```
PS C:\Users\hp> & C:/Users/hp/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/hp/OneDrive/Documents/import sys.py"
```

Operating System Project

The screenshot shows a code editor with a Python application titled "CPU Scheduling Simulator". The application is running, and its output is displayed in the "Scheduling Results" tab. The "Process Generator" tab shows the "Load Processes" field set to "C:/Users/hp/processproject.txt" and the "Scheduling Algorithm" set to "First Come First Serve (FCFS)". A green "Run Scheduling" button is visible. The "Scheduling Results" tab displays the following information:

Scheduling Algorithm: First Come First Serve (FCFS)

Average Waiting Time: 1.60
Average Turnaround Time: 6.50

Individual Process Results:

Process	Waiting	Turnaround
P1	0	3
P2	1	3
P3	0	1
P4	0	4
P5	0	1
P6	0	1
P7	0	3
P8	1	3
P9	0	20
P10	14	26

The code editor shows the following Python code:

```
30 self.num_processes = QSpinBox()
31 self.num_processes.setRange(1, 100)
32 self.num_processes.setValue(10)
33 left_params.addWidget(QLabel("Number of Processes:"))
```

The terminal at the bottom shows the command: `PS C:\Users\hp> & C:/Users/hp/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/hp/OneDrive/Documents/import sys.py"`

Conclusions

We successfully implemented and compared four scheduling algorithms. Each algorithm has strengths and weaknesses depending on the scenario. We learned how scheduling affects performance and user experience

References :

1. Course Lectures and Notes, Operating Systems, Spring 2025.
2. Project Specifications PDF, OS Scheduler.

Thanks