

COP 4610 – ASSIGNMENT 2

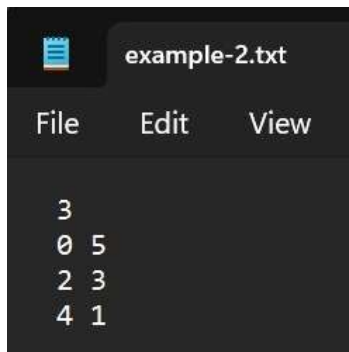
Members:

- Anthony L Carvalho (leader)
- Thales Moraes

This assignment involves creating a program that will read numbers from a separate .txt file and use these numbers to create a given amount of concurrent processes. The program will take as arguments the name of the .txt file to be read, and the algorithm that will be used to perform the calculations (“FCFS” or “SJF”). The .txt file contains the amount of processes as well as the arrival and burst times for each one of them. The program will then calculate the waiting time and turnaround time for each process and output the order of execution of the processes, together with the average waiting time and average turnaround time.

FUNCTIONALITY

The program reads a list of numbers from a separate text file. The first number represents how many processes should be created. The following numbers represent the arrival and burst times for each individual process.



The program then uses the numbers from the separate .txt file to perform calculations (waiting time and turnaround time) using a specific algorithm (“FCFS” or “SJF”), which was given to the program as an argument in the beginning.

```
int main(int argc, char *argv[]) {
    if (argc != 3) {
        printf("Usage: %s <filename> <FCFS/SJF>\n", argv[0]);
        return 1;
    }

    const char *filename = argv[1];
    const char *algorithm = argv[2];

    int numProcesses;
    Process processes[100]; // Assuming max 100 processes

    // Read the input from the file
    readFile(filename, &numProcesses, processes);

    // Choose the scheduling algorithm
    if (strcmp(algorithm, "FCFS") == 0) {
        fcfs(processes, numProcesses);
    } else if (strcmp(algorithm, "SJF") == 0) {
        sjf(processes, numProcesses);
    } else {
        printf("Unknown scheduling algorithm: %s\n", algorithm);
        return 1;
    }
}
```

Functions for “FCFS” and “SJF” algorithms, respectively:

```
// Function to calculate FCFS scheduling
void fcfs(Process processes[], int numProcesses) {
    int currentTime = 0;

    for (int i = 0; i < numProcesses; i++) {
        if (currentTime < processes[i].arrivalTime) {
            currentTime = processes[i].arrivalTime;
        }
        processes[i].waitingTime = currentTime - processes[i].arrivalTime;
        processes[i].turnaroundTime = processes[i].waitingTime + processes[i].burstTime;
        currentTime += processes[i].burstTime;
    }
}

// Function to calculate SJF scheduling
void sjf(Process processes[], int numProcesses) {
    int currentTime = 0, completed = 0;
    while (completed != numProcesses) {
        int shortestJob = -1;
        for (int i = 0; i < numProcesses; i++) {
            if (!processes[i].isCompleted && processes[i].arrivalTime <= currentTime) {
                if (shortestJob == -1 || processes[i].burstTime < processes[shortestJob].burstTime) {
                    shortestJob = i;
                }
            }
        }

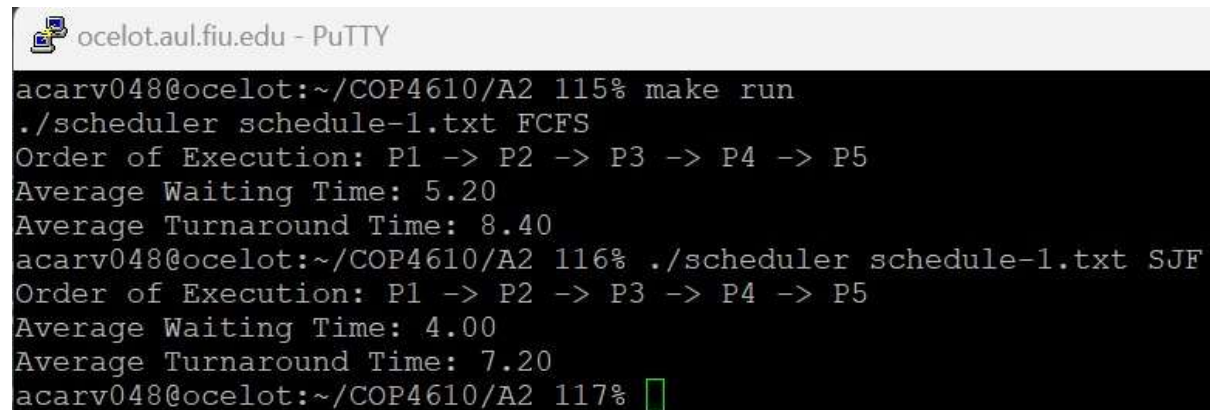
        // SJF algorithm logic, ensuring processes with the shortest burst time are prioritized
        if (shortestJob == -1) {
            currentTime++;
        } else {
            processes[shortestJob].waitingTime = currentTime - processes[shortestJob].arrivalTime;
            processes[shortestJob].turnaroundTime = processes[shortestJob].waitingTime + processes[shortestJob].burstTime;
            currentTime += processes[shortestJob].burstTime;
            processes[shortestJob].isCompleted = 1;
            completed++;
        }
    }
}
```

PROGRAM OUTPUT

After calculating the individual waiting and turnaround times for each process, using the correct algorithm, the program then calculates and displays the average waiting and turnaround time, as well as the order of execution of the processes.

```
// Function to display results
void displayResults(Process processes[], int numProcesses) {
    printf("Order of Execution: ");
    for (int i = 0; i < numProcesses; i++) {
        printf("P%d", processes[i].processID);
        if (i < numProcesses - 1) {
            printf(" -> ");
        }
    }
    printf("\n");
    printf("Average Waiting Time: %.2f\n", calculateAverageWaitingTime(processes, numProcesses));
    printf("Average Turnaround Time: %.2f\n", calculateAverageTurnaroundTime(processes, numProcesses));
}
```

Output example on PuTTY:



```
ocelot.aul.fiu.edu - PuTTY
acarv048@ocelot:~/COP4610/A2 115% make run
./scheduler schedule-1.txt FCFS
Order of Execution: P1 -> P2 -> P3 -> P4 -> P5
Average Waiting Time: 5.20
Average Turnaround Time: 8.40
acarv048@ocelot:~/COP4610/A2 116% ./scheduler schedule-1.txt SJF
Order of Execution: P1 -> P2 -> P3 -> P4 -> P5
Average Waiting Time: 4.00
Average Turnaround Time: 7.20
acarv048@ocelot:~/COP4610/A2 117% █
```