📋 Homework-5: FCFS CPU Scheduler

> ⚠️ **Important:** This assignment focuses on implementing the First-Come, First-Served (FCFS) CPU scheduling algorithm. Students will work with process scheduling concepts including arrival times, burst times, waiting times, and turnaround times.
>
> 💡 All the necessary concepts required to complete this homework are covered in the **CPU Scheduling lecture materials and textbook chapter on scheduling algorithms.**
>
> **You must use these strict gcc compilation flags while building your application:**
> **-std=c17**
> **-D_POSIX_C_SOURCE=200809L**
> **-Wall**
> **-Wextra**
> **-Werror**
> **-g**
> **-O0**
>
> ⚠️**There is only one submission - RESUBMISSION IS NOT ALLOWED. You must test your implementation with the provided testing framework on the Ocelot server before the final submission.**

📧 Grader Contact Information

For any questions or concerns related to assignment grading, please reach out to the TA listed below.

Communication Guidelines:

- Ensure that all communication is polite and respectful
- **You must contact the Graders first for any grading issues**
- If the matter remains unresolved, feel free to reach out to the instructor

Name: Instructor – bbhatkal@fiu.edu (⚠️ via the Canvas inbox only)
Section: COP 4610-U01 (84957)

Name: Angie Martinez – amart1070@fiu.edu (⚠️ via the Canvas inbox only)
Section: COP 4610-U02 (85031)

Name: Srushti Visweswaraiah – svisw003@fiu.edu(⚠️ via the Canvas inbox only)
Sections: COP 4610-UHA (85236)

✅ Section 1: Homework Overview

To implement the First-Come, First-Served (FCFS) CPU scheduling algorithm that schedules processes in the order of their arrival times, demonstrating fundamental CPU scheduling concepts and metrics calculation.

✅ Section 2: Learning Objectives

- Understand and implement the FCFS CPU scheduling algorithm.
- Work with process control blocks containing scheduling information.
- Calculate turnaround time and waiting time for processes.
- Implement proper tie-breaking rules when processes have the same arrival time.
- Handle CPU idle time when no processes are ready to execute.

✅ Section 3: Problem Description

You will implement the **First-Come, First-Served (FCFS) scheduling algorithm** that schedules processes based on their arrival times. FCFS is a non-preemptive scheduling algorithm where the process that arrives first gets executed first.

Algorithm Overview

1. **Process Ordering**: Processes are sorted by arrival time (earlier arrivals first)
2. **Tie-Breaking**: If multiple processes have the same arrival time, use Process ID (lower PID first)
3. **Non-Preemptive**: Once a process starts execution, it runs to completion
4. **CPU Idle Time**: If no process has arrived, CPU remains idle until the next process arrives
5. **Metrics Calculation**: Calculate waiting time and turnaround time for each process

Key Concepts

Turnaround Time (TAT): Total time from arrival to completion

- Formula: `TAT = Completion Time - Arrival Tim`

Waiting Time (WT): Time a process spends waiting in the ready queue before execution

- Formula: `Turnaroud Time - CPU burst Time`

Average Turnaround Time (ATT): Mean of all processes' turnaround times

- Formula: `ATT = Sum of all TAT / Number of Processes`

Average Waiting Time (AWT): Mean of all processes' waiting times

- Formula: `AWT = Sum of all WT / Number of Processes`

⚠️ **Note:** All these times are automatically computed by the framework. Students do not need to implement this functionality.

Tie-Breaking Rules

FCFS Tie-Breaking: When multiple processes have the same arrival time:

1. **Primary**: Arrival Time (ascending order - earlier first)
2. **Secondary**: Process ID (ascending order - lower PID first)

Example: If P3 and P1 both arrive at time 0, P1 executes first (lower PID)

☑ Section 4: Provided Files Framework

📖 **Refer to these files** for understanding the exact requirements.

⚠️ **Please do not make any modifications to the framework files provided.** You are required to implement **only** the file `submission_HW5.c`. Any changes to the framework files may result in errors during evaluation or loss of credit.

Folder Structure

```
PROVIDED_FILES/
├── submission_HW5.c      # Template file (implement your solution here)
├── fcfs_scheduler.h      # Header file (Do not modify!)
├── driver.c              # Main driver program (Do not modify!)
├── Makefile              # Build system (Do not modify!)
├── autograder_HW5.sh     # Testing script (Do not modify!)
├── batchgrader_HW5.sh    # Batch testing script (Do not modify!)
├── Sample_Executable/
│   └── fcfs              # Sample executable for testing
└── Testing/              # Test cases (Do not modify!)
    ├── Testcases/
    │   ├── input1.txt    # Test case 1
    │   ├── input2.txt    # Test case 2
    │   ├── input3.txt    # Test case 3
    │   └── input4.txt    # Test case 4
    └── Expected_Output/
        ├── output1.txt   # Expected output for test 1
        ├── output2.txt   # Expected output for test 2
        ├── output3.txt   # Expected output for test 3
        └── output4.txt   # Expected output for test 4
```

☑ Section 5: Submission Requirements

File Requirements (⚠️ exact names required)

- **submission_HW5.c**: Your complete FCFS implementation
- **README.txt or README.pdf or README.md**: Student information and documentation
    - Your README ⚠️ **MUST include**:

```
# Student Information
- Full Name: [Your Full Name]
- PID: [Your FIU Panther ID]
- Section: [Your Course Section]
- FIU Email: [Your @fiu.edu email]

# Homework: FCFS CPU Scheduler
[Brief description of your implementation approach]
[Explain how you implemented the FCFS scheduling algorithm]
[Describe how you handled tie-breaking when processes have same arrival time]
[Mention any challenges faced and how you solved them]
```

Deliverable Folder (`ZIP file`) Structure - ⚠️ **exact structure required** - **no additional files/subfolders**

> ⚠️ **DO NOT submit any other files other than these required files. The batch autograder may treat any additional items in the ZIP file as invalid, which will result in a grade of zero: submission_HW5.c, README.txt (or README.pdf or README.md)**
>
> 📁 All other required framework files will be supplied by the instructor to the autograder during final grading.

- ⚠️ You are required to submit your work as **a single ZIP archive file**.
- ⚠️ **Filename Format**: your `Firstname_Lastname.zip` (**exact format required**).
- **You will be held responsible for receiving a ZERO grade if the submission guidelines are not followed**.

```
Harry_Potter.zip
├── submission_HW5.c   # Your FCFS scheduler implementation
└── README.txt         # Your details and implementation approach
```

✅ Section 6: Developing and Testing Your Implementation

> ⚠️ **DO NOT modify the following provided framework files:**
> autograder_HW5.sh, batchgrader_HW5.sh, fcfs_scheduler.h, driver.c, Makefile, and Testing/ folder.

Required Function

You must implement this function in `submission_HW5.c`:

`void fcfs_scheduling(SchedulerContext *ctx)`

- **Input**: Pointer to SchedulerContext structure containing process information
- **Process**:
    1. Reset process states using the 💡 **provided** `reset_process_states(ctx)`
    2. Sort processes by arrival time (use Process ID for tie-breaking)
    3. Simulate FCFS scheduling:

        - Track current time
        - Handle CPU idle time when needed
        - Calculate waiting time for each process
    4. Display results using 💡 **provided** `display_results(ctx, "FCFS")`
- **Important Notes:**
    - Do NOT modify the global `ctx->processes[]` array order
    - Create a local copy for sorting if needed
    - Update waiting times in the original global array

Input File Format

Each test case file follows this format:

```
Process     Burst Time    Priority    Arrival Time
====================================================
P1          8             0           0
P2          3             0           1
P3          4             0           2
```

Format Explanation:

- **Line 1**: Header line (Process, Burst Time, Priority, Arrival Time)
- **Line 2**: Separator line (equal signs)
- **Lines 3+**: Process data rows
    - **Column 1**: Process ID (format: P1, P2, P3, etc.)
    - **Column 2**: Burst Time (CPU time required)
    - **Column 3**: Priority (not used in FCFS, but present in data)
    - **Column 4**: Arrival Time (when process enters ready queue)

Note: Priority field is ignored in FCFS scheduling but will be used in future assignments.

Output Format

Your implementation must produce output in this exact format:

```
FCFS
1       0        8
2       7        10
3       9        13
AWT:5.33
ATT:10.33
```

Output Format Explanation:

- **Line 1**: Algorithm name ("FCFS")
- **Lines 2-N**: Process results (one per process, in original input order)
    - **Column 1**: Process ID (PID)

    - **Column 2**: Waiting Time (WT) for that process
    - **Column 3**: Turnaround Time (TAT) for that process
- **Second-to-last line**: Average Waiting Time (format: `AWT:X.XX`)
- **Last line**: Average Turnaround Time (format: `ATT:X.XX`)
- **Blank line**: Single blank line at the end

STEP 1️⃣ - Sample Executable Testing:

💡 To better understand the FCFS implementation requirements, you are **highly encouraged** to test the instructor-provided executable:

```
# Navigate to Sample_Executable folder
cd Sample_Executable

# Provide execute permissions
chmod 777 fcfs

# Run with test cases (redirect input from test files)
./fcfs < ../Testing/Testcases/input1.txt
./fcfs < ../Testing/Testcases/input2.txt
./fcfs < ../Testing/Testcases/input3.txt
./fcfs < ../Testing/Testcases/input4.txt

# Compare output with expected results
./fcfs < ../Testing/Testcases/input1.txt > my_output1.txt
diff my_output1.txt ../Testing/Expected_Output/output1.txt
```

STEP 2️⃣ - Manual testing:

Ensure the following files are located in the **same directory**:

- **Your implementation**: `submission_HW5.c`
- **All the provided framework files**
- **The provided testcase folder:** `Testing/`

```
# Explore the Makefile first to understand build targets and compilation settings
cat Makefile

# Build the executable
make clean
make

# Test with all four test cases
./fcfs < Testing/Testcases/input1.txt
./fcfs < Testing/Testcases/input2.txt
./fcfs < Testing/Testcases/input3.txt
./fcfs < Testing/Testcases/input4.txt

# Save your output for comparison
```

```
./fcfs < Testing/Testcases/input1.txt > student_output1.txt
./fcfs < Testing/Testcases/input2.txt > student_output2.txt
./fcfs < Testing/Testcases/input3.txt > student_output3.txt
./fcfs < Testing/Testcases/input4.txt > student_output4.txt

# Compare your output with expected output
diff student_output1.txt Testing/Expected_Output/output1.txt
diff student_output2.txt Testing/Expected_Output/output2.txt
diff student_output3.txt Testing/Expected_Output/output3.txt
diff student_output4.txt Testing/Expected_Output/output4.txt

# If diff shows no output, your implementation is correct!
```

STEP 3️⃣ - Autograder testing:

Ensure the following files are located in the **same directory**, and then run the **Autograder**:

- **Your implementation**: `submission_HW5.c`
- **All the provided framework files**
- **The provided testcase folder:** `Testing/`

```
# Make autograder executable
chmod +x autograder_HW5.sh

# Run the autograder
./autograder_HW5.sh
```

- **Check your grade** and fix any issues
- **Repeat until** you achieve the desired score

STEP 4️⃣ - Batch Autograder testing:

> ⚠️ **Important:** The batch autograder processes multiple submissions. Please be patient!

The batch autograder, **used by the instructor**, processes all student submissions at once. It utilizes the provided framework files, extracts the required files from your submitted `.zip` file, and performs grading accordingly.

⚠️ **Backup Files Before Running Batch Grader - The script deletes existing files to rebuild the environment from scratch.**

Ensure the following files are located in the **same directory**, and then run the **Batch Autograder**:

- **All the provided framework files**
- **The provided testcase folder:** `Testing/`
- **Your final submission `ZIP` file** consisting the required files for the submission: example, `Harry_Potter.zip`

```
# Make batch autograder executable
chmod +x batchgrader_HW5.sh

# Run the batch-autograder
./batchgrader_HW5.sh
```

Final Testing requirements:

- ⚠️ **MUST test on ocelot server**: `ocelot-bbhatkal.aul.fiu.edu`
- ⚠️ **Test thoroughly before submission** - no excuses accepted
- ⚠️ **"Works on my computer" is NOT accepted** as an excuse
- ✅ **If you pass all test cases on the server, you will likely pass instructor's final grading**
- ✅ **What you see is what you get** - autograder results predict your final grade

---

☑️ Section 7: Grading Criteria

🤖 Autograder Testing

- ⚠️ Your implementation `submission_HW5.c` will be tested against the provided test cases as well as additional instructor test cases using the autograder.
- ⚠️ **Exact output matching required** - any deviation results in point deduction
- ⚠️ **Program must compile** without errors or warnings

Test Case Distribution:

- **Test 1 (25 points)**: FCFS with convoy effect and with processes having same arrival times (tests tie-breaking by PID)
- **Test 2 (25 points)**: FCFS with convoy effect removed and with processes having same arrival times (tests tie-breaking by PID)
- **Test 3 (25 points)**: FCFS with arrival times with CPU idle time (processes arrive with gaps)
- **Test 4 (25 points)**: FCFS with arrival times without CPU idle time

⛔ Penalties

- ⚠️ **Missing README**: **-10 points**
- ⚠️ **Missing** `submission_HW5.c`: **ZERO grade** (autograder compilation failure)
- ⚠️ **Incorrect ZIP filename**: **ZERO grade** (autograder extraction failure)
- ⚠️ **Wrong source filename**: **ZERO grade** (autograder compilation failure)
- 🔒 **Resubmission**: **NOT ALLOWED**

✅ Section 8: Technical Specifications

Data Structures

You will work with these structures defined in `fcfs_scheduler.h`:

```
typedef struct {
    int pid;                // Process ID
    int priority;           // Priority (not used in FCFS)
    int burst_time;         // CPU burst time
    int arrival_time;       // Arrival time
    int remaining_time;     // Remaining time (for preemptive algorithms)
    int waiting_time;       // Waiting time (YOU calculate this)
    int turnaround_time;    // Turnaround time (framework calculates)
    int completion_time;    // Completion time
    bool is_completed;      // Completion flag
} Process;

typedef struct {
    Process processes[MAX_PROCESSES];  // Array of processes
    int num_processes;                 // Number of processes
} SchedulerContext;
```

Input/Output Specifications

- **Input**: Read from stdin (redirected from test case files)
- **Input Format**: Process data with header and separator lines
- **Output**: Write to stdout in exact format specified
- **Output Precision**: Floating-point values formatted to 2 decimal places

✅ Section 9: Academic Integrity

- 👤 **This is an individual assignment**
- 💭 **You may discuss concepts** but not share code
- 📝 **All submitted code** must be your original work
- 📚 **You are encouraged to refer to textbook and lecture materials** but must write your own implementation
- ⚠️ **Plagiarism** is a serious offense and will result in penalties - **ZERO grade** (academic integrity violation).

🚀 **Good luck with your FCFS CPU Scheduler implementation!**