📋 Homework-6: Round Robin (RR) CPU Scheduler

> ⚠️ **Important:** This homework focuses on implementing the Round-Robin (RR) CPU scheduling algorithm. Students will work with process scheduling concepts including arrival times, burst times, time quantum, 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 Round-Robin (RR) CPU scheduling algorithm that schedules processes using a fixed time quantum in a circular queue fashion, demonstrating preemptive CPU scheduling concepts, context switching, and metrics calculation.

☑ Section 2: Learning Objectives

- Understand and implement the Round-Robin (RR) CPU scheduling algorithm.
- Work with process control blocks containing scheduling information.
- Implement preemptive scheduling with time quantum.
- Manage a ready queue using circular queue principles.
- 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.
- Track context switches in preemptive scheduling.

☑ Section 3: Problem Description

You will implement the **Round-Robin (RR) scheduling algorithm** that schedules processes using a fixed time quantum in a circular manner. RR is a preemptive scheduling algorithm where each process gets a small unit of CPU time (time quantum), after which it is preempted and added to the end of the ready queue.

Algorithm Overview

1. **Ready Queue Management**: Maintain a ready queue of processes waiting for CPU time
2. **Time Quantum**: Each process executes for at most one time quantum before being preempted
3. **Preemption**: If a process doesn't complete within its time quantum, it's moved to the back of the ready queue
4. **Process Arrival Handling**: New arrivals are added to the ready queue in order of arrival time
5. **Tie-Breaking**: If multiple processes arrive at the same time, use Process ID (lower PID first)
6. **CPU Idle Time**: If no process is ready, CPU remains idle until the next process arrives
7. **Metrics Calculation**: Calculate waiting time and turnaround time for each process

Key Concepts

Time Quantum (TQ): Fixed time slice allocated to each process (provided as input)

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

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

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

- Formula: `WT = Turnaround 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 time metrics are automatically computed by the framework. Students do not need to implement this functionality.

Round-Robin Execution Rules

1. **Initial Queue Setup**: At time 0, all processes with arrival time 0 enter the ready queue (sorted by PID if tied)
2. **Process Execution**:
    - The first process in the ready queue executes for min(time_quantum, remaining_time)
    - If remaining_time ≤ time_quantum, the process completes
    - If remaining_time > time_quantum, the process is preempted
3. **New Arrivals During Execution**:
    - Check for processes that arrive during the current execution window
    - Add newly arrived processes to the ready queue (in order of arrival, then PID)
4. **Queue Reordering After Execution**:
    - If the current process was preempted, add it to the end of the ready queue
    - Move to the next process in the ready queue
5. **CPU Idle Handling**: If the ready queue is empty, advance time to the next process arrival

Tie-Breaking Rules

RR 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 enters the ready queue 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_HW6.c`. Any changes to the framework files may result in errors during evaluation or loss of credit.

Folder Structure

```
PROVIDED_FILES/
├── skeleton_submission_HW6.c   # Implementation template file (rename to submission_HW6.c)
├── rr_scheduler.h              # Header file (Do not modify!)
├── driver.c                    # Main driver program (Do not modify!)
├── Makefile                    # Build system (Do not modify!)
├── autograder_HW6.sh           # Testing script (Do not modify!)
```

```
├── batchgrader_HW6.sh          # Batch testing script (Do not modify!)
├── Sample_Executable/
│   └── rr                      # 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
    │   ├── input5.txt          # Test case 5
    │   ├── input6.txt          # Test case 6
    │   ├── input7.txt          # Test case 7
    │   └── input8.txt          # Test case 8
    └── 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
        ├── output5.txt         # Expected output for test 5
        ├── output6.txt         # Expected output for test 6
        ├── output7.txt         # Expected output for test 7
        └── output8.txt         # Expected output for test 8
```

☑️ Section 5: Submission Requirements

File Requirements (⚠️ exact names required)

- **submission_HW6.c**: Your complete Round-Robin 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: Round-Robin CPU Scheduler
[Brief description of your implementation approach]
[Explain how you implemented the Round-Robin scheduling algorithm]
[Describe how you managed the ready queue and time quantum]
[Explain how you handled process arrivals during execution]
[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_HW6.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_HW6.c  # Your Round-Robin 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_HW6.sh, batchgrader_HW6.sh, rr_scheduler.h, driver.c, Makefile, and Testing/ folder.

Required Function

You must implement this function in `submission_HW6.c`:

`void round_robin_scheduling(SchedulerContext *ctx, int time_quantum)`

- **Input**:
  - Pointer to SchedulerContext structure containing process information
  - Integer time_quantum representing the time slice for each process
- **Process**:
  1. Reset process states using the 💡 **provided** `reset_process_states(ctx)`
  2. Initialize a ready queue to manage processes
  3. Sort processes by arrival time (use Process ID for tie-breaking)
  4. Simulate Round-Robin scheduling:
     - Track current time
     - Manage ready queue (add new arrivals, handle preemptions)
     - Execute processes for at most time_quantum units
     - Handle CPU idle time when needed
     - Calculate waiting time for each process

5. Display results using 💡 **provided** `display_results(ctx, "RR")`

- **Important Notes:**
  - Do NOT modify the global `ctx->processes[]` array order
  - Create a local copy/queue for scheduling
  - Update waiting times in the original global array
  - Properly handle process arrivals that occur during execution
  - Track remaining time for each process

Input File Format

Each test case file follows this format:

```
1  ========================================
2  TEST CASE 1: All Arrive at Time 0 (TQ=4)
3  Purpose: Basic Round Robin with multiple processes requiring multiple rounds
4  ========================================
5  Process     Burst Time    Priority    Arrival Time
6  ====================================================
7  P1          6             0           0
8  P2          5             0           0
9  P3          8             0           0
10 P4          7             0           0
```

Format Explanation:

- **Lines 1-4**: Test case header and description (informational only)
- **Line 5**: Column headers (Process, Burst Time, Priority, Arrival Time)
- **Line 6**: Separator line (equal signs)
- **Lines 7+**: 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 RR, but present in data)
  - **Column 4**: Arrival Time (when process enters ready queue)

Note:

- Your program will receive the time quantum (TQ=4) through the driver
- Priority field is ignored in Round-Robin scheduling but will be used in future assignments
- The driver automatically skips header lines and reads only the process data

Output Format

Your implementation must produce output in this exact format:

```
Round-Robin (RR):
PID      Turnaround_Time      Waiting_Time
1        3                    0
2        10                   5
3        9                    7
Average Turnaround Time: 7.33
Average Waiting Time: 4.00
```

Output Format Explanation:

- **Line 1**: Algorithm name ("Round-Robin")
- **Lines 2-7**: Process results (one per process, in original input order)
  - **Column 1**: Process ID (PID)
  - **Column 2**: Turnaround Time for that process
  - **Column 3**: Waiting Time for that process
  - **Line 6**: Average Turnaround Time
  - **Line 7**: Average Waiting Time

STEP 1 - Sample Executable Testing:

💡 To better understand the Round-Robin 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 rr

# Run with test cases (redirect input from test files)
# Repeat the process for all 8 test cases by changing the input file name accordingly.
./rr < ../Testing/Testcases/input1.txt

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

STEP 2 - Manual testing:

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

- **Your implementation**: `submission_HW6.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 eight test cases

./rr < ../Testing/Testcases/input1.txt


# Save your output for comparison
# Repeat the process for all 8 test cases by changing the input file name
# and student_output file name accordingly.
./rr < Testing/Testcases/input1.txt > student_output1.txt

# Compare your output with expected output
# Repeat the process for all 8 test cases by changing the student_output file name
# and  expected output file name accordingly.
diff student_output1.txt Testing/Expected_Output/output1.txt

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

STEP 3 - Autograder testing:

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

- **Your implementation**: `submission_HW6.c`
- **All the provided framework files**
- **The provided testcase folder:** `Testing/`

```
# Make autograder executable
chmod +x autograder_HW6.sh

# Run the autograder
./autograder_HW6.sh
```

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

STEP 4 - Batch Autograder testing:

> ⚠️ **Important:** The batch autograder may take a bit of time. 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_HW6.sh

# Run the batch-autograder
./batchgrader_HW6.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_HW6.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 (15.5 points)**: Basic Round Robin with multiple processes requiring multiple rounds
- **Test 2 (15.5 points)**: Processes arriving at different times with large gaps causing CPU idle periods
- **Test 3 (15.5 points)**: New process arrives exactly when current process's time quantum expires
- **Test 4 (15.5 points)**: Test CPU idle time handling with large gaps between process arrivals
- **Test 5 (15.5 points)**: Mix of processes with burst times both less than and greater than time quantum
- **Test 6 (15.55 points)**: Edge case with only one process requiring multiple time quantum
- **Test 7 (15.5 points)**: Multiple processes arrive while first process is executing within its time quantum
- **Test 8 (15.5 points)**: All processes have burst time less than time quantum, testing immediate completion without preemption

⛔ Penalties

- ⚠️ **Missing README**: **-10 points**
- ⚠️ **Missing** `submission_HW6.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 `rr_scheduler.h`:

```c
typedef struct {
    int pid;                // Process ID
    int priority;           // Priority (not used in RR)
    int burst_time;         // CPU burst time
    int arrival_time;       // Arrival time
    int remaining_time;     // Remaining time (YOU must update this)
    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;
```

☑️ 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 Round-Robin CPU Scheduler implementation!**