Priority Based CPU Scheduler

Kieran Beharry, Jacob Hayes, Owen O'Donnell, Luke Sansonetti

CSE 4300 – Fall 2024

Project Scope

- **Overview**: Simulate a CPU and threads using C++ and implement a priority-based scheduler.
- **Objectives**:
  - Develop a simulation environment for CPU and threads.
  - Create a scheduler that prioritizes threads based on their priority levels.
  - Implement dynamic priority adjustments (aging).
  - Test and document the scheduler's performance and behavior.

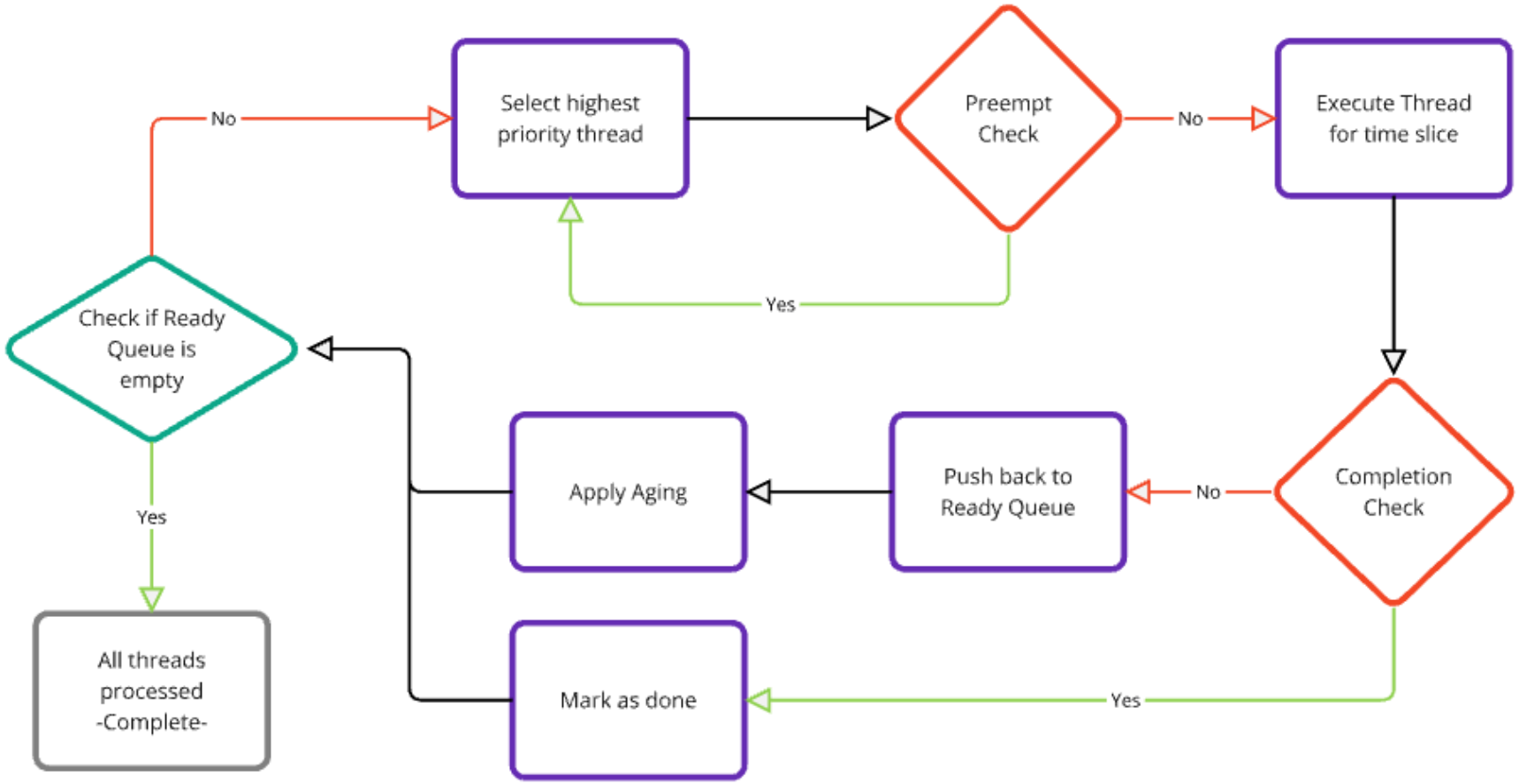
CPU and Thread Simulation

- **Goal**: Simulate the interaction between a CPU and multiple threads to evaluate a priority-based scheduling system.
- **Simulation Environment**:
  - Built using C++ to simulate thread properties and CPU processes.
  - **Thread Class**: Models individual threads, with properties like ID, priority, Time of Arrival (TOA), and Time to Complete (TTC).
- **Simulation Loop**:
  - Manages time increments, thread queues, and transitions between states.
  - The loop runs until all threads are completed, keeping track of states such as ready, running, or blocked.

CPU Overview

- **CPU Model**: The simulated CPU is responsible for handling thread execution.
- **Components**:
  - **Ready Queue**: Holds threads that are ready to be scheduled.
  - **Running Thread**: Only one thread can be running at a time.
  - **Time Slice**: Each thread is given a time slice, defining how long it can use the CPU before potentially being swapped.
- **Execution Cycle**:
  - **Fetch Next Thread**: The scheduler selects a thread based on priority.
  - **Run Thread**: Executes the thread for a specified time slice or until completion.
  - **Preemption**: If a higher-priority thread arrives, the currently running thread can be interrupted.

CPU Flow Diagram



Thread Properties

- **ID**: A unique identifier for each thread.
- **Priority**: Determines scheduling order; higher priority threads are executed first.
- **Time of Arrival (TOA)**: When the thread arrives in the system.
- **Time to Complete (TTC)**: The time required for the thread to complete.

Priority-Based Scheduler

- **Scheduler Goals**:
  - Prioritize threads to improve responsiveness for high-priority tasks.
  - Maximize CPU utilization by reducing idle time.
- **Priority Rules**:
  - Threads are ordered by priority; higher-priority threads are executed first.
  - In cases of equal priority, First-Come-First-Served (FCFS) is used.
- **Preemption**:
  - The system allows for preemption: a higher-priority thread arriving while a lower-priority one is running will cause the current thread to be suspended.

Dynamic Priority Adjustment - Aging

- **The Problem of Starvation**:
  - Threads with lower priorities can potentially be starved if higher-priority threads keep arriving.
- **Aging Mechanism**:
  - Gradually increases the priority of a thread the longer it waits in the ready queue.
  - Ensures all threads get a fair chance at execution.
- **Benefits of Aging**:
  - Reduces the risk of starvation.
  - Balances between prioritizing high-priority tasks and ensuring fairness.

Implementation Challenges

- **Tracking Time and Priority**:
  - Correctly implementing priority queues and handling arrival times.
  - Difficulties in managing threads arriving at different times while ensuring fair scheduling.
- **Handling Preemption**:
  - Implementing logic for stopping and saving the state of a currently running thread if a higher-priority one arrives.

Testing and Analysis

- **Test Scenarios**:
  - **High Priority Burst**: Threads with extremely high priority arriving in bursts.
  - **Equal Priority Jobs**: All threads have equal priority to test FCFS scheduling.
  - **Mixed Priorities**: A combination of high, medium, and low-priority threads to simulate real-world conditions.
  - **Thread Starvation:** Jobs coming in with high potential for starvation.
- **Input Files**: Jobs defined in various configurations using `.txt` files.
- **Expected Outcomes**: Assess correct order of execution, fairness, and accurate turnaround and response times.

Basic Testing

- Job files show:
  - Job ID
  - Priority
  - Time of Arrival
  - Time to Complete

Basic Aging Test

Thread 1:
Priority: 3, Time of Arrival: 0, Time to Completion: 5

Thread 2:
Priority: 2, Time of Arrival: 1, Time to Completion: 3

```
==============================
Statistics
------------------------------
Thread: 2    Turn around time: 3    Response time: 0
Thread: 1    Turn around time: 8    Response time: 0
==============================
The average response time was 0 and the average turn around was 5.5
==============================
```

```
==============================
Time 0
------------------------------
Running next thread
Thread 1 taking the CPU at time 0.
Thread 1 doing work on the CPU. Has 4 left.
Thread 1 giving up the CPU.
ID: 1 Priority: 3 Wait: 0
==============================
Time 1
------------------------------
Running next thread
Thread 2 taking the CPU at time 1.
Thread 2 doing work on the CPU. Has 2 left.
Thread 2 giving up the CPU.
ID: 2 Priority: 2 Wait: 0
ID: 1 Priority: 3 Wait: 1
==============================
Time 2
------------------------------
Running next thread
Thread 2 taking the CPU at time 2.
Thread 2 doing work on the CPU. Has 1 left.
Thread 2 giving up the CPU.
ID: 2 Priority: 2 Wait: 0
ID: 1 Priority: 3 Wait: 2
==============================
Time 3
------------------------------
Running next thread
Thread 2 taking the CPU at time 3.
Thread 2 doing work on the CPU. Has 0 left.
Thread 2 is done.
ID: 1 Priority: 3 Wait: 3
Thread 1's priority has increased to 2
==============================
```

```
==============================
Time 4
------------------------------
Running next thread
Thread 1 taking the CPU at time 4.
Thread 1 doing work on the CPU. Has 3 left
Thread 1 giving up the CPU.
ID: 1 Priority: 2 Wait: 0
==============================
Time 5
------------------------------
Running next thread
Thread 1 taking the CPU at time 5.
Thread 1 doing work on the CPU. Has 2 left
Thread 1 giving up the CPU.
ID: 1 Priority: 2 Wait: 0
==============================
Time 6
------------------------------
Running next thread
Thread 1 taking the CPU at time 6.
Thread 1 doing work on the CPU. Has 1 left
Thread 1 giving up the CPU.
ID: 1 Priority: 2 Wait: 0
==============================
Time 7
------------------------------
Running next thread
Thread 1 taking the CPU at time 7.
Thread 1 doing work on the CPU. Has 0 left
Thread 1 is done.
==============================
```

Starvation Test

Thread 1:
Priority: 3, Time of Arrival: 0, Time to Completion: 4

Thread 2:
Priority: 2, Time of Arrival: 2, Time to Completion: 5

Thread 3:
Priority: 1, Time of Arrival: 7, Time to Completion: 5

```
==============================
Statistics
------------------------------
Thread: 1    Turn around time: 7    Response time: 0
Thread: 2    Turn around time: 8    Response time: 0
Thread: 3    Turn around time: 7    Response time: 0
==============================
The average response time was 0 and the average turn around was 7.33333
==============================
```

Starvation Test

Hoarding Test

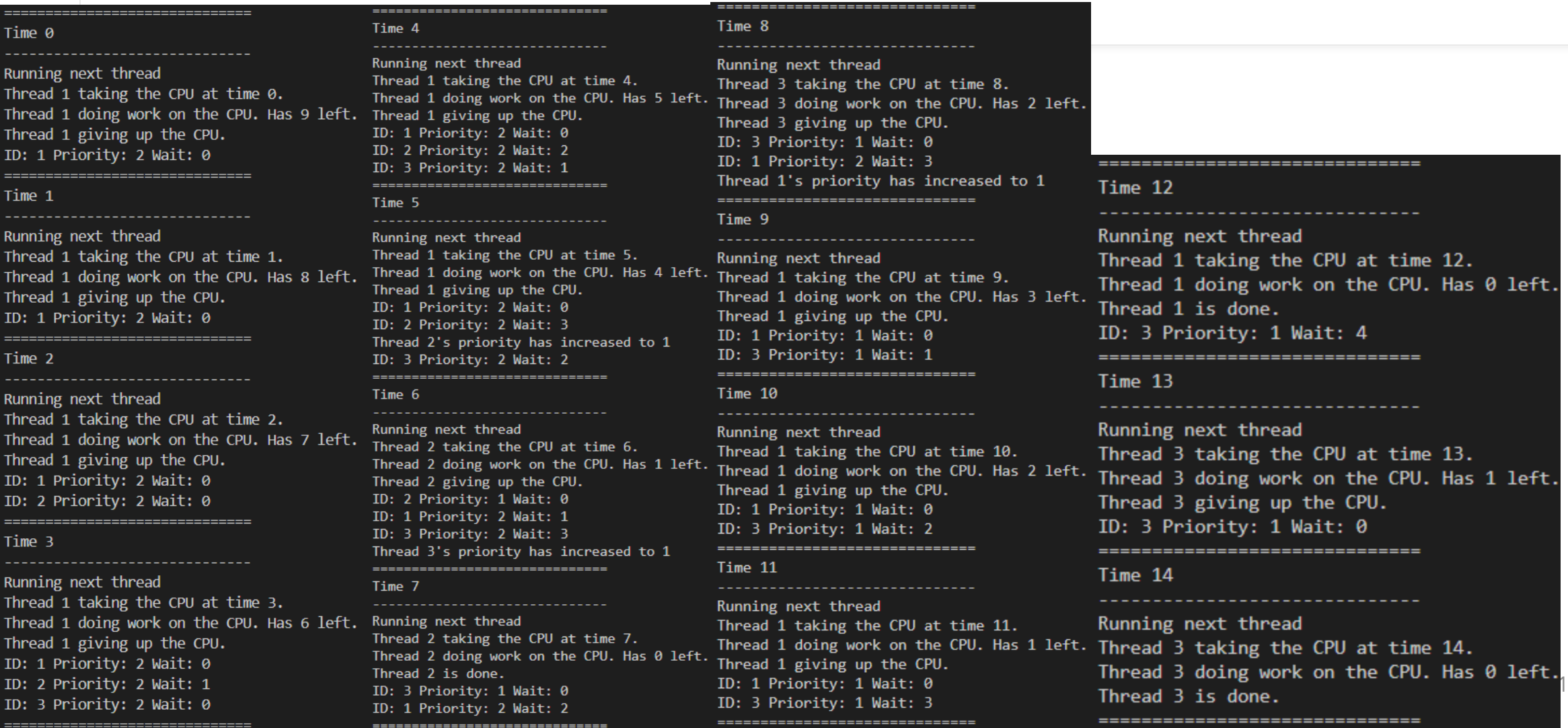
Thread 1:
Priority: 2, Time of Arrival: 0, Time to Completion: 10

Thread 2:
Priority: 2, Time of Arrival: 2, Time to Completion: 2

Thread 3:
Priority: 2, Time of Arrival: 3, Time to Completion: 3

```
==============================
Statistics
------------------------------
Thread: 2    Turn around time: 6    Response time: 4
Thread: 1    Turn around time: 13    Response time: 0
Thread: 3    Turn around time: 12    Response time: 5
==============================
The average response time was 3 and the average turn around was 10.3333
==============================
```

Hoarding Test



Conclusion

Key Achievements:

- Successfully implemented a priority-based CPU scheduler with support for priority-based preemption.
- Effectively integrated aging to mitigate starvation and ensure fair scheduling.
- Developed and ran various test cases to validate performance and functionality.