XV6 Report - Scheduler

Yash Bhaskar - 2021114012

27th September 2023

1 First-Come-First-Serve (FCFS)

In our project, we also implemented the First-Come-First-Serve (FCFS) scheduling algorithm, which prioritizes processes based on their creation time. This algorithm ensures that the process that arrives first gets executed first.

1.1 Modified Process Structure

To support the FCFS algorithm, we made a minor modification to the `struct proc` data structure. We introduced the following additional variable:

- `ctime`: This variable stores the creation time of each process, indicating the time (in ticks) when the process was created.

1.2 Scheduler Modifications

In the scheduler function, we implemented the following changes to enforce the FCFS scheduling discipline:

1. Process Selection: We traverse all processes and select the process with the minimum creation time (ticks). This process represents the oldest process in the queue and is the next to be executed.
2. Execution: We execute the selected process using the `swtch` call, allowing it to run until completion or until it explicitly yields the CPU.

The FCFS scheduling algorithm ensures that processes are executed in the order in which they were created, making it suitable for scenarios where fairness and strict ordering are essential.

Our FCFS implementation enhances the operating system's ability to prioritize processes based on their arrival time, ensuring that older processes get executed before newer ones.

2 Multi-Level Feedback Queue (MLFQ)

In our project, we also implemented the Multi-Level Feedback Queue (MLFQ) scheduling algorithm to efficiently manage and prioritize processes. This scheduling algorithm is designed to handle varying priorities based on the process's behavior.

2.1 Modified Process Structure

To support the MLFQ algorithm, we made modifications to the `struct proc` data structure. We introduced the following additional variables:

- `level`: This variable stores the queue number of the process, indicating its priority level.
- `in_queue`: A binary flag (1 if the process belongs to a queue, 0 otherwise).
- `change_queue`: The number of ticks after which the process should be preempted and moved to the next queue.
- `enter_ticks`: Records the timestamp when the process entered its current queue.

2.2 Scheduler Modifications

In the scheduler function, we implemented the following changes for MLFQ:

1. Aging: We regularly check if processes need to be aged and moved to a higher-priority queue. If a process is aged, we remove it from all queues, set `in_queue` to 0, and decrement the `level` by 1 if it is not already at the lowest priority (level 0).
2. Queue Assignment: Processes are assigned to queues based on their `level`. We push processes into the queue corresponding to their current level.
3. Queue Execution: We iterate through the queues, starting from the highest priority. If we find a runnable process in a queue, we select that process for execution in the next time slice.
4. Process Execution: We run the chosen process in the scheduled time slice.

2.3 Trap Handling

In the `trap.c` file, we also incorporated MLFQ-related logic. Specifically, we check if a process has aged, and if so, we preempt it and move it to the next queue with a higher priority.

2.4 Additional Configuration Parameters

We introduced two new configuration parameters in the `param.c` file to fine-tune the MLFQ algorithm:

- Number of Queues: We set the number of queues to 5 in our implementation, allowing us to manage processes across multiple priority levels.
- Aging Time: We configured the aging time to 30 ticks in our implementation, determining the frequency at which processes are checked for aging.

2.5 Queue Implementation

To manage the queues effectively, we created a new data structure and defined it within the `proc.h` header file. We implemented essential queue operations like `popfront`, `pushback`, `front`, `size`, and `delete` in a separate file named `queue.c`.

Our implementation of the Multi-Level Feedback Queue (MLFQ) scheduling algorithm enhances the operating system's ability to manage processes with varying execution requirements efficiently.

3 Graphs

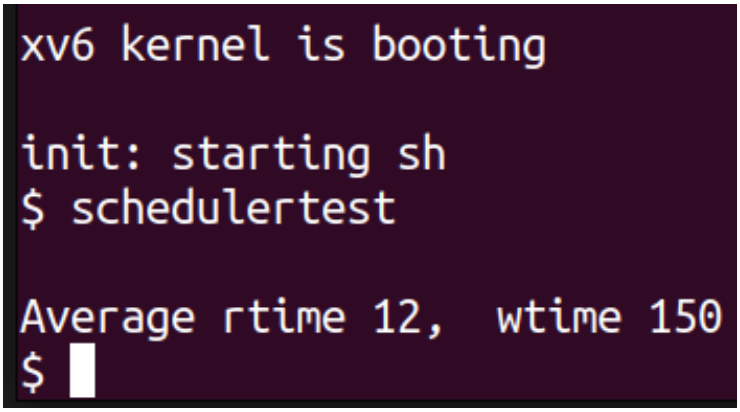

Figure 1: Image 1: Round Robin Scheduling in Action

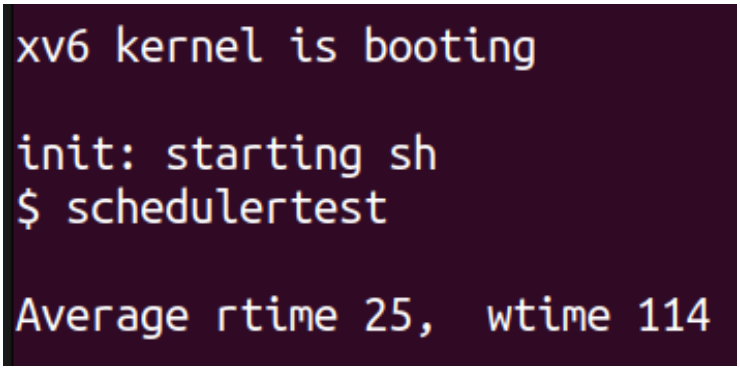

Figure 2: Image 1: FCFS Scheduling in Action

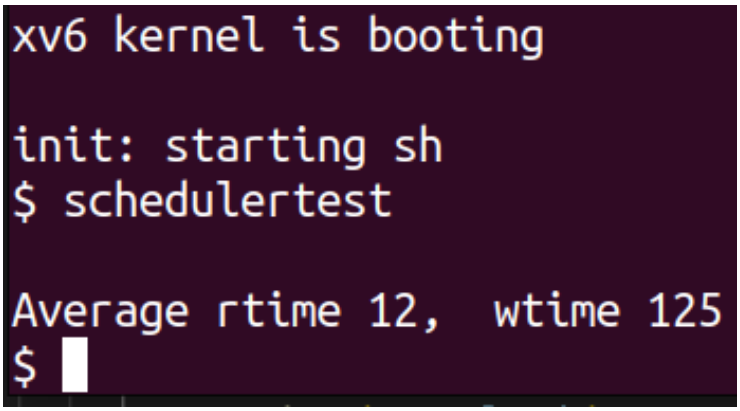

Figure 3: Image 1: MLFQ Scheduling in Action

4 Comparison

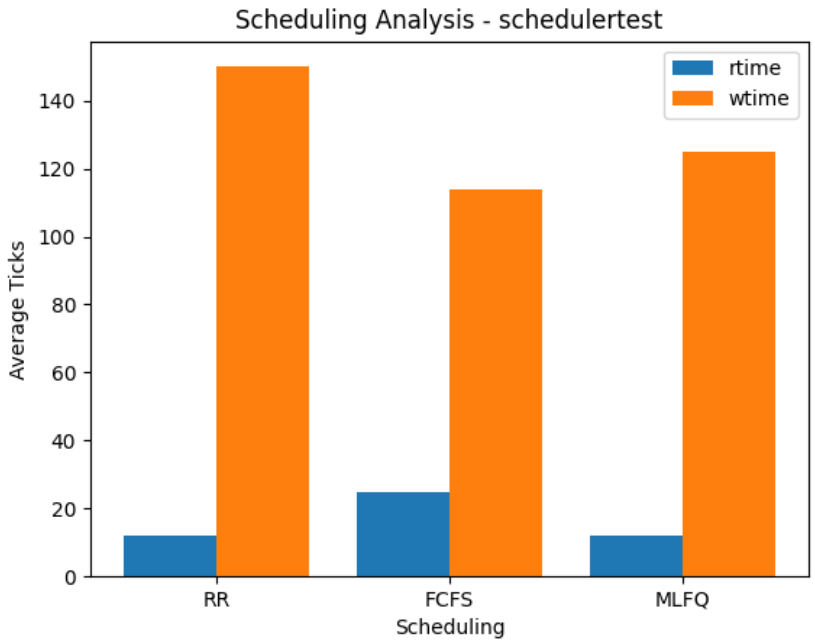

Figure 4: Image 1: MLFQ Scheduling in Action