School of Computer Engineering

Operating Systems Course Project: Comparative Analysis of Scheduling Algorithms in xv6

Instructor: Dr. Keyvan RahimiZadeh
TA: MohammadJavad Jalilvand

Release Date: 1404/03/21
Due Date: 1404/04/06

1 Project Overview

This project introduces CPU scheduling concepts by implementing and comparing different scheduling algorithms in the `xv6` operating system. Students will modify the `xv6` kernel to implement First-Come-First-Served (FCFS) and Lottery scheduling, then compare their performance with the default Round Robin scheduler.

2 Learning Objectives

- Understand fundamental CPU scheduling concepts
- Gain experience modifying a real operating system kernel
- Implement two scheduling algorithms in `xv6` (FCFS and Lottery)
- Add necessary system calls to support the new schedulers, such as setting tickets or priorities, and retrieving process statistics.
- Design and run test workloads to evaluate scheduler performance.

3 Installation

- First, make sure to have all the dependencies :

  ```
  sudo apt-get update && sudo apt-get install --yes \
          build-essential git qemu-system-x86
  ```

- After successfully installing the required programs, clone the Github repository of xv6 source code:

  ```
  git clone https://github.com/mit-pdos/xv6-public
  ```

- Now just compile the kernel and run:

  ```
  make qemu
  ```

4 Deliverables

- Modified XV6 source code with both schedulers implemented
- Test programs demonstrating each scheduler
- Report containing:
  - Design decisions and implementation challenges
  - Performance comparison tables and graphs
  - Analysis of scheduler strengths/weaknesses
  - Discussion of fairness and starvation

5 Implementation Steps

These are key files you have to modify:

```
proc.c        # Scheduler implementation
proc.h        # Process structure modifications
sysproc.c     # System call implementations
syscall.c     # System call numbers
user.h        # User-space interfaces
usys.S        # System call stubs
```

Also study XV6's default Round Robin scheduler befor starting.

5.1 Implementing FCFS Scheduling

To implement First-Come-First-Served scheduling in XV6:

1. **Create a ready queue**:
   - Modify the process table in `proc.h` to add:
     ```
     uint arrival_time;  // Timestamp when process became ready
     ```
   - Maintain all ready processes in a FIFO queue ordered by their arrival time
2. **Modify the scheduler** in `proc.c`:
   - Select the process that has been waiting longest (head of queue)
   - Remove preemption - processes run until they block or terminate
   - Update arrival time whenever:
     - Process is created (`fork()`)
     - Process unblocks from I/O
3. **Handle edge cases**:
   - When no processes are ready, keep idle loop
   - Maintain compatibility with existing process switching

5.2 Implementing Lottery Scheduling

For the Lottery scheduling implementation:

1. **Add ticket management**:
   - Add to `proc.h`:
     ```
     int tickets;          // Number of lottery tickets
     int original_tickets;// Initial ticket allocation
     ```

- Implement `settickets()` system call:
    - Validate ticket count (minimum 1 ticket)
    - Store in both `tickets` and `original_tickets`

2. **Modify scheduling logic**:

    - Calculate total tickets of all runnable processes
    - Generate random number in [0, total_tickets) range
    - Select process using weighted random choice:

    ```
    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
            if(p->state != RUNNABLE) continue;
            if(random_val < p->tickets) {
                    // Select this process
                    break;
            }
            random_val -= p->tickets;
    }
    ```

3. **Handle default cases**:

    - Assign 10 tickets to processes without explicit assignment
    - Reset tickets to original values after process completion

5.3 Scheduling Policy Selection

Add mechanism to switch between schedulers:

1. **Add policy flag**:

    - Add global variable in `proc.c`:

    ```
    int scheduling_policy = 0; // 0=RR, 1=FCFS, 2=LOTTERY
    ```

2. **Implement policy system call**:

    - Create `setpolicy()` in `sysproc.c`:

    ```
    int sys_setpolicy(void) {
            int policy;
            if(argint(0, &policy) < 0) return -1;
            if(policy < 0 || policy > 2) return -1;
            scheduling_policy = policy;
            return 0;
    }
    ```

    - Add wrapper in `user.h` and syscall numbers

3. **Modify scheduler dispatch**:
    - In scheduler main loop:

      ```
      switch(scheduling_policy) {
              case 1: fcfs_schedule(); break;
              case 2: lottery_schedule(); break;
              default: rr_schedule(); // Original RR
      }
      ```

5.4 Performance Comparison

Students should develop comparison methodology:

1. **Create test workloads**:
    - CPU-bound processes (e.g., number crunching)
    - I/O-bound processes (frequent `sleep()` calls)
    - Mixed workloads
2. **Measure metrics**:
    - Turnaround time (completion - arrival)
    - Waiting time (ready queue time)
    - Response time (first run - arrival)
    - Throughput (processes completed per time unit)
    - Fairness (tickets vs. CPU share in Lottery)
3. **Analysis report**:
    - Present results in tables/graphs
    - Compare performance under different loads
    - Discuss tradeoffs between schedulers
    - Analyze fairness properties

Metric	**Round Robin**	**Lottery**	**MLFQ**
Average Waiting Time (ticks)	120	80	65
Maximum Response Time (ticks)	40	25	15
Throughput (jobs/sec)	8	10	12

Table 1: Sample scheduler performance metrics