Project 01

Implementing simple system schedulers

Due date
2025. 04. 30. 23:59

Overview

- Scheduling is fundamental to multiprogramming operating systems.
  - It selects the most appropriate process among the runnable ones for execution.
  - This enables the operating system to use the given resources more efficiently.
- In xv6, a very simple version of the Round-Robin scheduler is used by default.
  - The goal of this assignment is to implement schedulers learned during lecture in the xv6 system.
  - Through this process, you will design, implement, test, and analyze real working schedulers.

Default RR Scheduler

- The scheduler in xv6 uses a basic Round-Robin policy by default.
- Its default behavior is as follows:
  - When a timer interrupt occurs, the currently running process is switched to the RUNNABLE state, and the next process in the process array is selected for execution.
  - If the last process in the array has been run, the scheduler loops back to the first process.
  - The interval at which the timer interrupt occurs is called tick, and the default value is approximately 100 milliseconds.
  - In other words, a context switch between processes occurs roughly every 100 ms.

Scheduler

- The following scheduling policies are to be implemented in xv6 :
- FCFS(First-Come, First-Served) part
  - Implement basic FCFS scheduling
  - Understanding the fundamentals of non-preemptive scheduling
- MLFQ(Multi-Level Feedback Queue) part
  - Implement Round-Robin scheduling within levels
  - Implement priority scheduling for the lowest level

Specification – FCFS

- Modify the existing scheduler to implement FCFS scheduling.
- Use the existing PCB (Process Control Block, `struct proc` in kernel/proc.h) without adding new variables.
- The scheduler should select the process with the earliest creation time.
- Ensure that once a process is selected for execution, it runs to completion or until it voluntarily yields the CPU.
- Update necessary parts of the kernel to support this scheduling mechanism.

Specification - MLFQ

- 3-level feedback queue
- The scheduler consists of three queues : $L_0 \sim L_2$
  - with lower-numbered queues having higher priority.
- Each queue L_i has a time quantum of $2i + 1$ ticks.
- All newly created processes start in the highest priority queue (L_0).

Specification - MLFQ

- The L_0 and L_1 queues follow the basic Round-Robin(RR) scheduling policy.
  - By default, the scheduler selects RUNNABLE processes from L_0.
  - If there are no RUNNABLE processes in L_0, it checks L_1.
  - If L_1 also has no RUNNABLE processes, it checks L_2.
- When a process in L_0 uses up its entire time quantum,
  - It is demoted to L_1, and its time quantum is reset.
- Likewise, if a process in L_1 uses up its entire time quantum,
  - It is demoted to L_2, and its time quantum is reset.

Specification - MLFQ

- The L_2 queue uses priority scheduling.
  - If there are no RUNNABLE processes in L_1, the scheduler selects processes from L_2 .
  - Within the L_2 queue, processes with higher priority values are scheduled first.
  - A system call named setpriority must be implemented to allow changing a process's priority.
  - Each process has its own priority value, which is an integer from 0 to 3.
    - Higher values indicate higher priority (e.g., 3 is the highest, 0 is the lowest).
    - When a process is created, its default priority is set to 3.
    - If multiple processes have the same priority, any of them can be scheduled first.
  - Priority only affects processes in the L_2 queue and has no effect in L_0 or L_1.

Specification - MLFQ

- If a process in the L_2 queue uses up its entire time quantum, its priority is decreased by 1, and its time quantum is reset
  - However, the priority will not go below 0
- To prevent starvation, priority boosting must be implemented.
- **Priority boosting**
  - Every time the global tick count reaches 50, all processes are moved back to the L_0 queue.
  - During priority boosting, the time quantum and priority of all processes is reset.
- Whenever control is passed to the scheduler such as by a timer interrupt, yield, or sleep, the scheduler must select the highest-level queue that contains any RUNNABLE processes.

Specification – Mode Switch

- By default, processes are scheduled within the FCFS after the first xv6 boot.
  - While scheduling occurs in the FCFS mode, priority boosting does not take place.
  - If the mlfqmode system call is invoked, scheduling is switched to use the MLFQ.
- If the fcfsmode system call is invoked in MLFQ mode, processes will be scheduled within the FCFS again.
- When switching between modes, the global tick count is reset to 0.
- If a system call is invoked to switch to the current mode(e.g., calling fcfsmode while already in FCFS), a message is displayed indicating that the system is already in that mode, and no changes are made.

Specification – Common

- All preemptions must occur within 100ms(1 tick).
- The coding convention should follow the default style used in xv6.
  - ex) indentation, brace placement
- For simplicity, it is assumed that only one CPU is used.
  - use make CPUS=1 (already reflected) or add the –smp 1 option to QEMU.
- If a user process behaves incorrectly and needs to be forcefully terminated, a message indicating the forced termination must be printed, and no other unintended behavior should occur.
  - After termination, the next process must be scheduled normally.

Required system calls

- The following system calls must be implemented according to the given specifications :
- void yield(void)
  - Give up current process’s own cpu.
- int getlev(void)
  - Returns the queue level to which the process belongs.
  - If the system is currently in FCFS mode, returns 99.

Required system calls

- int setpriority(int pid, int priority)
  - Sets the priority of the process with the given pid.
  - Returns :
    - 0 on success
    - -1 if no process with the given pid exists
    - -2 if the priority value is not between 0 and 3 (inclusive)

Required system calls

- int mlfqmode(void)
  - Switches the current scheduling mode from FCFS to MLFQ.
    - All ready processes that were being scheduled under FCFS are now placed in the L0 queue of MLFQ.
    - If multiple processes are ready, they will be inserted into the L0 queue while maintaining their relative order from FCFS selection.
    - During this transition, each process's priority is set to 3, time quantum to 0, and level to 0.
  - Returns 0 upon successful mode transition.
    - If the system is already in MLFQ mode when the system call is invoked, an error message is printed and -1 is returned; no changes are made.

Required system calls

- int fcfsmode(void)
  - Switches the current scheduling mode from MLFQ to FCFS
    - All processes previously managed by MLFQ will now be scheduled according to FCFS policy
    - After transition, processes will be scheduled based on their creation time, regardless of their previous level or priority in MLFQ. The earliest created process will be scheduled first.
    - During this transition, each process's priority, level, and time quantum are initialized to -1.
  - Returns 0 upon successful mode transition.
    - If the system is already in FCFS mode when the system call is invoked, an error message is printed and -1 is returned; no changes are made.

Evaluation

- The evaluation criteria are as follows, and partial credit will be given if parts of each scheduler's specification are correctly implemented.

Evaluation Criteria	Points
FCFS	20
MLFQ	50
Mode Switch	10
Wiki	20
Total	100

Evaluation

- **Completeness** The xv6 operating system must function correctly according to the specification requirements.
- **Wiki & Comment** Grading will be based on the wiki documentation, so the wiki should be written in as much detail as possible.
- **Deadline** The submission deadline must be strictly observed. After the deadline, your GitHub writing permissions will be revoked.
- **DO NOT SHARE AND COPY!!**

Wiki

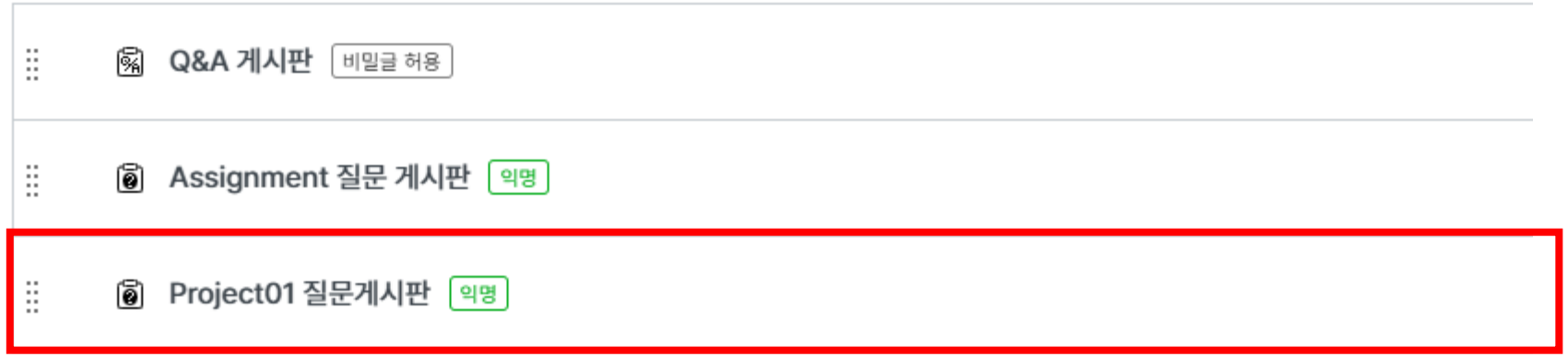
- **Design** Outline your implementation approach for meeting the assignment requirements.
- **Implementation** Explain key code modifications and their purpose, focusing on changes from the original code.
- **Results** Show evidence of successful implementation with:
  Compilation process, Screenshots of working code, Explanation of program flow
- **Troubleshooting** Describe any problems encountered, solutions applied, and any unresolved issues.
- Additional content may be included if relevant.

Submission

- Submit your implemented code and wiki through GitHub.
  - Refer to the announcement and create a new repository.
  - Rename the repository to “project01-[student ID]”
- The wiki file should be named "OS_project01_[class number]_[student ID].pdf".
- Submission deadline: **April 30, 2025, 23:59**
  - Late submissions will be accepted via **email** until **May 01, 2025, 23:59**, but will only receive **50%** of the possible score.

Q&A

- For questions related to the project, please use the question board (Project 01 Question Board) on the LMS.
- Questions sent by email will not be answered.
- For questions not related to the project, please use the Q&A board or send an email.



Q & A