DEPARTMENT OF COMPUTER SCIENCE, HANYANG UNIVERSITY

MLFQ and Stride scheduling

Operating Systems: Three Easy Pieces

Programming Project with xv6

Due date: April 28, 2019

1 PROCESS SCHEDULER

CPU scheduling is the basis of multiprogrammed operating systems. The CPU scheduler chooses the best candidate among available processes, and this leads the operating system to make best use of resources efficiently, thus being more productive. There is an operating system module that selects the next jobs to be admitted into the system and the next process to run. We call the module a process scheduler.
In this project, our goal is improving xv6 process scheduler by implementing the process scheduler as a combination of multi-level feedback queue and stride scheduling method.

1.1 MULTILEVEL FEEDBACK QUEUE SCHEDULING

Unlike multilevel queue scheduling algorithm where processes are permanently assigned to a queue, multilevel feedback queue scheduling allows a process to move between queues. This movement is facilitated by the characteristic of the CPU burst of the process. If a process uses too much CPU time, it will be moved to a lower-priority queue. This scheme leaves I/O-bound and interactive processes in the higher priority queues. In addition, a process that waits too long in a lower-priority queue may be moved to a higher priority queue. This form of aging (or priority boosting) also helps to prevent starvation of certain lower priority processes.

1.2 STRIDE SCHEDULING

Stride scheduling is a deterministic resource proportional-share algorithm. By interpreting the stake of a resource using the concept of stride the algorithm overcomes unfairness of lottery algorithm which is the basic proportional share scheduling method.

2 PROJECT MILESTONES

1. **The First Milestone**. Design a new scheduler with MLFQ and Stride on the abstraction level. Follow steps below, because it should be implemented incrementally.
   a) First step: Stride scheduling
      - Stride scheduler should be run default scheduler of xv6.
      - Make a system call (i.e. cpu_share) that requests the portion of CPU and guarantees the calling process to be allocated that CPU time.
      - Total CPU portion of using cpu_share API is 20%. Exception handling is needed for exceeding request.
      - Processes that doesn't call any other APIs (e.g. cpu_share, run_MLFQ) should run with equal CPU portion(100% / number of processes).

   b) Second step: Combine the MLFQ algorithm with MLFQ
      - 3-level feedback queue
      - Each level of queue adopts Round Robin policy with different time quantum.
      - Each queue has different time allotment.
      - To prevent starvation, priority boost is required
      - MLFQ only start to run when user API(run_MLFQ) is called.
      - MLFQ should run with fixed portion of CPU (20%).

   - **Due Date**: April 14

2. **The Second Milestone**. Implement a newly designed scheduler. Observe its behaviour and make a report.

   IMPORTANT: Never proceed to this milestone unless your design gets confirmed by TAs or the professor.

   - **Due Date**: April 28

3 IMPLEMENTATION SPECIFICATION

1. Original xv6 scheduler (default scheduler in xv6)
   Xv6 uses RR (Round Robin) scheduling as a default scheduling algorithm. When a timer interrupt occurs, a currently running process is switched over to the next runnable process. The time interval between consecutive timer interrupts is called a *tick*. The default value set for the tick in xv6 is about 10ms.

2. Stride scheduling
   - Stride scheduler should be a default scheduler of xv6.
   - If a process wants to get a certain amount of CPU share, then it invokes a new system call cpu_share() to set the amount of CPU share.
   - The total sum of CPU share requested from processes can not exceed 20% of CPU time. Exception handling needs to be properly implemented to handle oversubscribed requests.

3. MLFQ (Multi-level feedback queue) scheduling
   - MLFQ consists of three queues with each queue applying the round robin scheduling.
   - The scheduler chooses a next ready process from MLFQ. If any process is found in the higher priority queue, a process in the lower queue cannot be selected until the upper level queue becomes empty.
   - Each level of queue adopts Round Robin policy with different time quantum.
     - The highest priority queue: 1 tick
     - Middle priority queue: 2 ticks
     - The lowest priority queue: 4 ticks
   - Each queue has different time allotment.
     - The highest priority queue: 5 ticks
     - Middle priority queue: 10 ticks
   - To prevent starvation, priority boosting needs to be performed periodically.
     - The priority boosting is the only way to move the process upward.
     - Frequency of the priority boosting: 100 ticks
   - If a process wants to run with MLFQ, then it invokes a new system call to change scheduler.
   - When a process is newly created, it initially enters Stride scheduler. The process will be managed by the MLFQ only if the run_MLFQ() system call has been invoked.
   - Unless stated otherwise, please follow the algorithm described in the textbook, or mechanisms you learnt in the lecture.

4. Required system calls
   Following system calls should be newly implemented for this project, and TAs will assume all these system calls are implemented in your xv6 kernel when testing your kernel
   - yield: yield the cpu to the next process.
     - int yield(void)
   - cpu_share: inquires to obtain cpu share(%).
     - int cpu_share(int)
   - run_MLFQ: be involved in MLFQ.
     - void run_MLFQ(void)
   - getlev: get the level of current process ready queue of MLFQ. Returns one of the level of MLFQ (0/1/2).
     - int getlev(void)

5. MLFQ & Stride scheduling scenario
   - We will give you a test program code that allows you to check the behavior of scheduling.
   - To verify your scheduling algorithm, analyze the scheduling behavior of the process by writing scheduling scenarios.
   - Write your analysis in the report. There will be an extra point if the results are well organized(ex, using graph).
   - We will give you more detail about this with our test program code.

4 General Requirements

1. You should upload your code to hanyang gitlab repository. (note that email submission is not accepted!!)

2. Every documentation should be written in your Gitlab Wiki.
   - Wiki: documentation and explanation storage. For example, analysis report, design documents, background information, dev note, etc. can be here.

3. Follow the appropriate coding convention.
   - In xv6 kernel source code, follow the xv6 style.

4. Do brainstorming with your class mates via on/offline.
   - Piazza can be a good dev community. Share what you think with your friends.

5. Do NOT share the code.

- Please keep the programming ethics. Your code will be kept in GitLab. Take responsibility for your code.

6. The last version(commit and push) before deadline will be pulled and evaluated. Any updates after the deadline are invisible to the test program.