Department of Computer Science, Hanyang University

MLFQ and Stride Scheduling

Operating Systems: Three Easy Pieces

Programming Project with xv6

Due date: April 19, 2022

1 Process scheduler

CPU scheduling is the basis of operating systems that supports multi-programming. The CPU scheduler chooses a candidate among available processes, and this leads the operating system to make use of resources efficiently, thus being more productive. For such use, the operating system has a module that selects the next process to be admitted into the system. We call the module a process scheduler. Our ultimate goal of this project is to improve the xv6 process scheduler by implementing a new process scheduler, combining the policies of both the multilevel feedback queue and stride scheduling.

1.1 Multilevel feedback queue scheduling

Unlike the multilevel queue scheduling algorithm where processes are permanently assigned to a queue, multilevel feedback queue scheduling allows a process to be moved between the queues. This movement is facilitated by the characteristic of the CPU burst of a process. If a process uses too much CPU time, it will be moved to a lower-priority queue. Such scheme allows I/O-bound and interactive processes to remain 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 the unfairness of a lottery algorithm which is a basic proportional-share scheduling method.

2 Project Milestones

1. **The First Milestone**. Design a new scheduler with MLFQ and Stride on an abstraction level. Follow the steps below.
   a) First step: MLFQ
      - 3-level feedback queue
      - Each level of queue adopts a Round Robin policy with different time quantum
      - Each queue has a different time allotment
      - To prevent starvation, priority boost is required

   b) Second step: Combine the *stride scheduling* algorithm with MLFQ
      - Make a system call (i.e. set_cpu_share) that requests a portion of CPU and guarantees the calling process to be allocated that much of a CPU time.
      - The total amount of stride processes are able to get at most 80% of the CPU time. Exception handling is required for exceeding requests.
      - The rest of the CPU time (20%) should run for the MLFQ scheduling which is the default scheduling policy in this project

   - **Due Date**: April 5
2. **The Second Milestone**. Implement the newly designed scheduler. Observe its behaviour and make a report.

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

   - **Due Date**: April 19

3 Implementation specification

1. The original xv6 scheduler (i.e., default scheduler in xv6)
   xv6 uses RR (Round Robin) scheduling as a default scheduling algorithm. When a timer interrupt occurs, the 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. MLFQ (Multilevel feedback queue) scheduling
    - MLFQ consists of three queues with each queue applying the round robin scheduling.
    - The scheduler chooses the next process that's ready from the 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 adopts the Round Robin policy with a different time quantum.
        - The highest priority queue: 1 tick
        - Middle priority queue: 2 ticks
        - The lowest priority queue: 4 ticks
    - Each queue has a different time allotment.
        - The highest priority queue: 5 ticks
        - Middle priority queue: 10 ticks
    - To prevent starvation, priority boosting needs to be performed periodically.
        - Priority boosting is the only way to move the process upward.
        - Frequency of priority boosting: 100 ticks of MLFQ scheduling
    - MLFQ should always occupy at least 20% of the CPU share.
    - Unless stated otherwise, please follow the algorithm described in the textbook, or mechanisms you've learnt in the lecture.

3. Stride scheduling
    - If a process wants to get a certain amount of CPU share, then it should invoke a new system call to request the desired amount of CPU share.
    - When a process is newly created, it initially enters the MLFQ. The process will be managed by the stride scheduler only if the set_cpu_share() system call has been invoked.
    - The total sum of CPU share requested from the processes in the stride queue can not exceed 80% of the total CPU time. Exception handling needs to be properly implemented to handle oversubscribed requests.
    - Do not allocate CPU share if the request induces surpass of the CPU share limit.

4. Required system calls
   The following system calls should be newly implemented for this project, and TAs will assume that all these system calls are implemented in your xv6 kernel when testing your project:
    - yield: yield the cpu to the next process
        - int sys_yield(void) - wrapper

    - int yield(void) - system call
    - Return 0.
    - **Gaming the scheduler**: Calling the system call yield before a tick is done will allow a process to stay in the high level of a MLFQ for as long as the programmer wants. In this project, we will forbid this by incrementing a tick for the process in such cases.
  - getlev: get the level of the current process in the MLFQ.
    - int sys_getlev(void) - wrapper
    - int getlev(void) - system call
    - Return one of the levels of MLFQ (0/1/2), otherwise a negative number.
  - set_cpu_share: inquires to obtain a cpu share (%).
    - int sys_set_cpu_share(void) - wrapper
    - int set_cpu_share(int) - system call
    - Return 0 if successful, otherwise a negative number.

5. MLFQ Stride scheduling scenario
   - We will give you a test program code that allows you to check the behavior of your scheduler.
   - 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 (e.g., graph illustration, well-formed figures, etc.).
   - We will give you more detail about this with our test program code.

4 General Requirements

1. You should upload your code to hconnect (note that email submission is not accepted !!)
2. You must upload a detailed documentation describing your design and implementation on your Gitlab Wiki.
   - Your wiki including analysis report, design documents and all other related resources should be placed here.
   - We will evaluate your project depending on your wiki, and the ones that do not provide an appropriate document will receive a huge penalty on your project score.
3. Follow the appropriate coding convention.

- Within the 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 your thoughts with your friends.
5. Do **NOT** share/use the code on public.
    - **Plagiarism issue will not be forgiven under any circumstances as mentioned before. Be careful not to upload any of your code on a public community since both the contributor and the plagiarist will be treated as equal.**
6. The last version (commit) **before** the deadline will be evaluated. Any updates after the deadline are invisible on evaluation.