



# Project #1

Dept. of Computer Science  
Hanyang University





Implement simple scheduler on xv6



- **스케줄링은** 다중프로그램 운영체제의 기본이 되며, 실행 가능한 프로세스 중 가장 적합한 프로세스를 선택하여 실행합니다. 이를 통해 운영체제가 주어진 자원을 더 효율적으로 사용할 수 있도록 합니다.
- **xv6에서는** 기본적으로 매우 간단한 버전의 Round-Robin 방식의 스케줄러를 사용합니다.
- **이번** 과제에서는 새로운 스케줄러를 xv6에 구현해보는 것을 목적으로 하며, 이러한 과정에서 실제 동작하는 스케줄러를 디자인하고 구현, 테스트, 분석해봅시다.



## Default Round-Robin Scheduler

- xv6의 스케줄러는 기본적으로 Round-Robin 정책을 사용하고 있으며, 기본 동작은 다음과 같습니다.
  - **타이머** 인터럽트가 발생하면  
현재 실행중인 프로세스를 RUNNABLE 상태로 전환시키고,  
프로세스의 배열에서 다음 인덱스의 프로세스를 실행시킵니다.  
배열의 마지막 프로세스까지 실행시켰다면 배열의 처음 프로세스부터 다시 시작합니다.
  - **타이머** 인터럽트가 발생하는 주기를 tick이라 하며, 기본 값은 약 10ms입니다.  
즉, 약 10ms마다 한 번씩 프로세스 간의 context switch가 발생합니다.



- xv6 내에 구현되어야 할 스케줄러는 다음과 같습니다.
- **MLFQ scheduler**
  - 3-level feedback queue
- xv6 내의 모든 프로세스는 기존의 Round-Robin 스케줄러가 아닌, MLFQ 스케줄러를 통해 스케줄링됩니다.
- MLFQ 스케줄러에 의해 스케줄링되는 프로세스보다 항상 우선적으로 처리되어야 하는 프로세스가 있을 수 있습니다.



- 3-level feedback queue
- L0 ~ L2, 총 3개의 큐로 이루어져 있고, 숫자가 작을수록 우선순위가 더 높습니다.
- 각 큐는 각각 다른 time quantum을 가집니다.
  - Ln :  $2n+4$  ticks
- 처음 실행된 프로세스는 가장 높은 레벨의 큐(L0)에 들어갑니다.
- L0 큐와 L1 큐는 기본 Round-Robin 정책을 따릅니다.
  - 스케줄러는 기본적으로 L0 큐의 RUNNABLE한 process를 스케줄링합니다.
  - L0 큐의 RUNNABLE한 프로세스가 없을 경우, L1의 process를 스케줄링합니다.
  - L1 큐의 RUNNABLE한 프로세스가 없을 경우, L2의 process를 스케줄링합니다.



## Specification

- L2 큐는 priority 스케줄링을 합니다.
  - 우선순위를 설정할 수 있는 시스템 콜인 setPriority() 시스템 콜이 추가되어야 합니다.
  - Priority는 프로세스가 처음 실행될 때 3으로 설정되며, setPriority() 시스템 콜을 통해 0~3 사이의 값을 설정할 수 있습니다. Priority 값이 작을수록 우선순위가 높습니다.
  - 우선순위가 같은 프로세스끼리는 FCFS로 스케줄링 됩니다.
  - 우선순위는 L2 큐에서만 스케줄링에 영향을 줍니다.
  - L2 큐에서 실행된 프로세스가 L2에서의 time quantum을 모두 사용한 경우, 해당 프로세스의 priority 값이 1 감소하고 time quantum은 초기화됩니다.
  - Priority 값이 0일 경우, 더 이상 값을 감소시키지 않고 0으로 유지합니다.
- Starvation을 막기 위해 priority boosting이 구현되어야 합니다.
- Priority boosting
  - Global tick이 100 ticks가 될 때마다 모든 프로세스들은 L0 큐로 재조정됩니다.
  - Priority boosting이 될 때, 모든 프로세스들의 priority 값은 3으로 재설정됩니다.
  - Priority boosting이 될 때, 모든 프로세스들의 time quantum은 초기화됩니다.



- **MLFQ** 스케줄러에 의해 스케줄링 되는 프로세스보다 **항상 우선적으로** 처리되어야 하는 프로세스가 있을 수 있습니다.
- 이를 위해 스케줄러 lock/unlock을 가능하게 하는 **schedulerLock()**, **schedulerUnlock()** 시스템 콜이 구현되어야 합니다.
- 스케줄러를 lock하는 프로세스가 존재할 경우 MLFQ 스케줄러는 동작하지 않고, 해당 프로세스가 최우선적으로 스케줄링 되어야 합니다.



- **schedulerLock()** 시스템 콜은 '프로세스가 우선적으로 처리되어야 할 자격을 증명'하기 위한 암호를 인자로 받습니다. 암호는 자신의 학번으로 합니다.
  - 성공적으로 실행되었다면, global tick은 priority boosting 없이 0으로 초기화 됩니다.
  - **schedulerLock()** 시스템 콜은 기존에 존재하는 프로세스만 호출할 수 있습니다.
- **schedulerUnlock()** 시스템 콜은 '우선적으로 처리되어야 할 프로세스 자격을 해제'하기 위한 암호를 인자로 받습니다. 암호는 자신의 학번으로 합니다.
  - 성공적으로 실행되었다면, 기존의 MLFQ 스케줄러로 돌아갑니다.
  - **해당 프로세스는 L0 큐의 제일 앞으로 이동하고, priority는 3으로 설정합니다.**
  - 해당 프로세스의 time quantum은 초기화됩니다.
- **두 시스템** 콜 호출 시, 암호가 일치하지 않으면 해당 프로세스를 강제로 종료합니다.
  - 이때 프로세스의 pid, time quantum, 현재 위치한 큐의 level을 출력합니다.



- **schedulerLock()** 시스템 콜을 호출한 프로세스는 특정 조건을 만족할 경우 MLFQ로 돌아갑니다.
  - schedulerUnlock() 시스템 콜이 호출될 때
    - 해당 프로세스는 MLFQ 스케줄러의 L0 큐 맨 앞으로 이동합니다.
    - 해당 프로세스의 time quantum은 초기화됩니다.
    - 해당 프로세스의 priority는 3으로 재조정됩니다.
  - Global tick이 100 ticks가 될 때
    - 해당 프로세스는 MLFQ 스케줄러의 L0 큐 맨 앞으로 이동합니다.
    - Priority boosting이 발생합니다.
- **두 시스템** 콜은 인터럽트를 통해 실행될 수 있어야 합니다.
  - 129번 인터럽트 호출 시, schedulerLock() 시스템 콜을 실행합니다.
  - 130번 인터럽트 호출 시, schedulerUnlock() 시스템 콜을 실행합니다.



## Required system calls

- **다음의** 시스템 콜들은 정해진 명세대로 구현되어야 올바르게 테스트됩니다.
  - 유저모드에서 사용할 수 있어야 합니다.
- `yield()` : 다음 프로세스에게 프로세서를 양보합니다.
  - `void yield(void)`
- `getLevel()` : 프로세스가 속한 큐의 레벨을 반환합니다.
  - `int getLevel(void)`
- `setPriority()` : 해당 pid의 프로세스의 priority를 설정합니다.
  - `void setPriority(int pid, int priority)`
- `schedulerLock()` : 해당 프로세스가 우선적으로 스케줄링 되도록 합니다.
  - `void schedulerLock(int password)`
- `schedulerUnlock()` : 해당 프로세스가 우선적으로 스케줄링 되던 것을 중지합니다.
  - `void schedulerUnlock(int password)`



- 모든 action은 10ms(1tick) 안으로 일어나면 됩니다.
- Coding Convention은 xv6 코드의 기본적인 형태를 따릅니다.
  - 들여쓰기, 중괄호 위치 등
- 컴파일은 기존의 xv6와 같이 make를 통해 이루어지며, 기존에 제공되던 옵션들은 동일하게 제공되어야 합니다.
- 구현의 편의성을 위해 CPU의 개수는 하나임을 가정합니다.
  - 테스트를 할 때는 make의 인자로 CPUS=1을 주거나, qemu의 옵션으로 -smp=1 을 주고 테스트하시면 됩니다.



- **Code**

- 명세의 요구 조건을 모두 올바르게 구현해야 합니다.
- 코드에는 주석이 함께 작성되어야 합니다.

- **Wiki**

- 테스트 프로그램과 위키를 기준으로 채점됩니다.
- 위기는 실제로 동작하는 장면과 함께 본인의 디자인에 대해 상세히 작성되어야 합니다.

- **Submission**

- 데드라인을 반드시 지켜야 합니다.
- 데드라인 이전에 GitLab에 마지막으로 commit/push 된 코드를 기준으로 채점됩니다.



- **Completeness**
  - 명세의 요구 조건을 모두 올바르게 구현해야 합니다.
- **Defensiveness**
  - 발생할 수 있는 예외 상황에 대처할 수 있어야 합니다.
- **Comment**
  - 코드에는 반드시 주석이 있어야 합니다.
- **DO NOT ASK OR SHARE CODE !!**



## • Design

- 명세에서 요구하는 조건에 대해서 어떻게 구현할 계획인지, 어떤 자료구조와 알고리즘이 필요한지, 자신만의 디자인을 서술합니다.

## • Implement

- 본인이 새롭게 구현하거나 수정한 부분에 대해서 무엇이 기준과 어떻게 다른지, 해당 코드가 무엇을 목적으로 하는지에 대한 설명을 구체적으로 서술합니다.

## • Result

- 컴파일 및 실행 과정과, 해당 명세에서 요구한 부분이 정상적으로 동작하는 실행 결과를 첨부하고, 이에 대한 동작 과정에 대해 설명합니다.

## • Trouble shooting

- 과제를 수행하면서 마주하였던 문제와 이에 대한 해결 과정을 서술합니다. 혹여 문제를 해결하지 못하였다면 어떤 문제였고 어떻게 해결하려 하였는지에 대해서 서술합니다.

## • And whatever you want 😊



- You should upload your code to HYU GitLab repository.
- You should upload your wiki to HYU LMS (.pdf)
- Wiki file name is “ELE3021\_ project01\_[classNO]\_ [studentID].pdf”
- **Due date : 2023. 04. 23. 23:59 (No late submission allowed)**



Thank you :)