Operating Systems
Project 02

- Implementing simple schedulers on xv6-

Due date : 11:59 pm May 16

Xv6 Scheduler

- 스케줄링은 다중프로그램 운영체제의 기본이 되며 실행가능한 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 사용하고 있습니다.
- 이번 과제에서는 이론시간에 배운 스케줄러들을 xv6에 구현해보는 것을 목적으로 하며 이러한 과정에서 실제 동작하는 스케줄러들을 디자인하고 구현, 테스트, 분석해봅니다.

Default RR scheduler

- Xv6의 스케줄러는 기본적으로 Round-Robin 정책을 사용하고 있으며 기본 동작은 다음과 같습니다.
  - 타이머 인터럽트가 발생하면 현재 실행중인 프로세스를 RUNNABLE 상태로 전환시키고, 프로세스 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.

Schedulers

- 구현할 스케줄러들
  1. FCFS scheduler
  2. Multilevel queue scheduler
     - Round-robin
     - FCFS
  3. MLFQ scheduler
     - Round-robin scheduler
     - Priority scheduler
- 각 스케줄러에 대한 자세한 내용은 책과 이론수업자료를 참조

Specification - FCFS

- 먼저 생성(fork())된 프로세스가 먼저 스케줄링 되어야 합니다.
- 기본적으로 스케줄링된 프로세스는 종료되기 전까지는 switch-out 되지 않습니다.
- 그러나 만일 프로세스가 스케줄링 된 이후 200 ticks가 지날 때까지 종료되거나 SLEEPING 상태로 전환되지 않으면 강제 종료시켜야 합니다.
- 만약 실행 중인 프로세스가 SLEEPING 상태로 전환되면(IO, sleep 등) 다음으로 생성된 프로세스가 스케줄링됩니다.
  - SLEEPING상태이면서 먼저 생성된 프로세스가 깨어나면 다시 그 프로세스가 스케줄링 됩니다.

Specification – Multilevel Queue

- 스케줄러는 2개의 큐로 이루어집니다.
  - Round Robin: pid가 짝수인 프로세스들은 round robin으로 스케줄링됩니다.
  - FCFS: pid가 홀수인 프로세스들은 FCFS로 스케줄링됩니다.
- Round Robin 큐가 FCFS 큐보다 우선순위가 높습니다.
  - 즉, pid가 짝수인 프로세스가 항상 먼저 스케줄링되어야 합니다.
- FCFS 큐는 먼저 생성된 프로세스를 우선적으로 처리합니다.
  - 즉, pid가 홀수인 프로세스들끼리는 pid가 더 작은 것부터 먼저 스케줄링되어야 합니다.
- SLEEPING 상태에 있는 프로세스는 무시해야 합니다.
  - i.e. RR 큐에 있는 프로세스 중에 RUNNABLE인 프로세스가 없다면 FCFS 큐를 확인해야 합니다.

Specification - MLFQ

- 2-level feedback queue
  - + Disabling / enabling preemption
- L0와 L1 두 개의 큐로 이루어져 있고, L0의 우선순위가 더 높습니다.
- 각 큐는 각각 다른 time quantum을 가집니다.
- 처음 실행된 프로세스는 모두 가장 높은 레벨의 큐(L0)에 들어갑니다.
- Starvation을 막기 위해 priority boosting이 구현되어야 합니다.

Specification - MLFQ

- L0 큐는 기본 RR 정책을 따릅니다.
  - L0 큐의 모든 프로세스가 SLEEPING 상태가 되면 L1 큐의 프로세스들을 실행합니다.

- L1 큐는 priority scheduling을 합니다.
  - 우선순위를 설정할 수 있는 시스템 콜인 setpriority() 시스템 콜이 추가되어야 합니다.
  - 프로세스가 처음 실행될 때의 priority는 0으로 설정되며, setpriority 시스템 콜을 통해 0~10 사이의 값을 설정할 수 있습니다. priority의 값이 클수록 우선순위가 높습니다.
  - 우선순위가 같은 프로세스끼리는 FCFS로 우선순위가 정해집니다.
  - Priority 값은 L1 큐에서만 유효하고, L0 큐에서는 아무런 역할이 없습니다.

Specification - MLFQ

- 각 레벨의 time quantum
  - 기본적으로 각 레벨에서 주어진 quantum동안 실행한 후에 switch-out됩니다.
  - L0 큐에서 실행된 프로세스가 quantum을 모두 사용한 경우 L1 큐로 내려갑니다.
  - L1 큐에서 실행된 프로세스가 quantum을 모두 사용한 경우 해당 프로세스의 priority값이 1 감소합니다. (단, 0에서는 더 감소하지 않음)
  - L0: 4 ticks
  - L1: 8 ticks

- Priority boosting
  - 200 ticks마다 모든 프로세스들은 L0 큐로 올라가고, priority도 0으로 리셋됩니다.

Specification - MLFQ

- **MLFQ 스케줄링을 무시하고 프로세서를 독점해서 사용할 수 있게 하는 시스템 콜 monopolize가 추가되어야 합니다.**
- **monopolize 시스템 콜의 인자로 프로세서를 독점할 자격을 증명할 암호를 받습니다. 이 암호는 자신의 학번으로 합니다.**
  - 암호가 일치하지 않으면 프로세스를 강제로 종료시킵니다.
- **독점적으로 프로세스가 실행되는 중에는 MLFQ가 동작하지 않고 계속 그 프로세스만 실행되어야 합니다.**
- **독점적으로 실행되던 프로세스가 monopolize를 다시 호출하면 기존의 MLFQ 스케줄링으로 돌아갑니다. 그 프로세스는 L0 큐에 들어가고 priority가 0으로 설정됩니다.**
  - 여기서도 암호가 올바르지 않으면 해당 프로세스는 강제 종료되어야 하고, 그와는 무관하게 MLFQ 스케줄링으로는 돌아가야 합니다.

Specification - common

- **모든 preemption은 10ms(1tick)안으로 일어나면 됩니다.**
- **Coding Convention은 xv6코드의 기본적인 형태를 따릅니다. (들여쓰기, 중괄호의 위치 등)**

- **구현의 편의성을 위해 cpu의 개수는 하나임을 가정합니다.**
  - 테스트를 할 때는 make의 인자로 CPUS=1을 주거나, qemu의 옵션으로 -smp=1을 주고 테스트하시면 됩니다.
- **유저 프로세스가 잘못된 행동을 하여 강제 종료시키는 경우 강제로 종료시켰다는 메시지를 출력해야 하며, 그 이외의 문제가 발생해서는 안 됩니다. 또한 종료시킨 이후에는 다음 프로세스가 정상적으로 스케줄링 되어야 합니다.**

Required system calls

- 다음의 시스템 콜들은 반드시 정해진 명세대로 구현하여야 합니다.
- yield: 다음 프로세스에게 cpu를 양보합니다.
  - **void yield(void)**
- getlev: MLFQ 스케줄러일 때, 프로세스가 속한 큐의 레벨을 반환합니다(0,1).
  - **int getlev(void)**
- setpriority: MLFQ 스케줄러일 때, pid를 가지는 프로세스의 우선순위를 priority로 설정합니다.
  - **int setpriority(int pid, int priority)**
  - priority 설정에 성공한 경우 0을 반환합니다.
  - pid가 존재하지 않는 프로세스이거나, 자신의 자식 프로세스가 아닌 경우 -1을 반환합니다.
  - priority가 0 이상 10 이하의 정수가 아닌 경우 -2를 반환합니다.
- monopolize: MLFQ 스케줄러일 때, CPU를 독점하여 사용하도록 설정합니다. 인자로 독점 기능을 사용하기 위한 암호(자신의 학번)을 받습니다.
  - **void monopolize(int password)**

Compile

- 컴파일은 기존의 xv6와 같이 make를 통해 이루어지며, 기존에 제공되던 옵션들은 동일하게 제공되어야 합니다.
- make의 인자로 어떠한 스케줄링 정책을 사용할지 결정됩니다.
  - SCHED_POLICY=MULTILEVEL_SCHED|MLFQ_SCHED
  - MLFQ인 경우 MLFQ_K의 값이 추가로 주어집니다.
  - Ex) make SCHED_POLICY=MLFQ_SCHED MLFQ_K=3 CPUS=1 -j
- make의 인자로 컴파일시 코드를 분기하는 예제를 참조하시기 바랍니다.

평가지표

평가 기준은 다음과 같으며, 각 스케줄러 명세의 일부분을 완성하였다면 부분점수가 주어집니다.

평가 기준	배점
FCFS	20
Multilevel Queue	30
MLFQ	30
Wiki	20
Total	100

평가

- **Completeness**: 명세의 요구조건에 맞게 xv6가 올바르게 동작해야 합니다.
- **Defensiveness**: 발생할 수 있는 예외 상황에 대처할 수 있어야 합니다.
- **Wiki&Comment**: 테스트 프로그램과 위키를 기준으로 채점이 진행되므로 위키는 최대한 상세히 작성되어야 합니다.
- **Deadline**: 데드라인을 반드시 지켜야 하며, 데드라인 이전 마지막으로 commit/push된 코드를 기준으로 채점됩니다.
- **Do NOT copy or share your code.**

Wiki

- Wiki에는 다음의 항목들이 서술되어야 합니다.
  - **디자인**: 각 스케줄러를 어떻게 구현해야 하는지, 추가/변경되는 컴포넌트들이 어떻게 상호작용하는지, 이를 위해 어떠한 자료구조들이 필요한지 등을 서술합니다.
  - **구현**: 실제 구현과정에서 변경하게 되는 코드영역이나 작성한 자료구조 등에 대하여 서술합니다.
  - **실행 결과**: 각 스케줄러가 올바르게 동작하고 있음을 확인할 수 있는 실행 결과와 이에 대한 설명을 서술합니다.
  - **트러블슈팅**: 코드 작성 중 생겼던 문제와 이에 대한 해결 과정을 서술합니다.
- 스케줄러에 대한 유의미한 그래프나 구조도 등 위키를 풍부하게 만들어주는 요소에는 추가 점수가 있을 수 있습니다.

테스트 프로그램

- 작성한 스케줄러가 올바르게 동작하는지 테스트하기 위한 테스트 프로그램이 추후에 업로드 될 것입니다.
- 이는 과제의 편의성을 위한 것으로 제공되는 테스트 프로그램 이외의 테스트가 존재할 수 있습니다.
- 테스트는 명세를 넘어가는 기능을 요구하지 않으며, 몇몇 예외상황에 대한 테스트는 있을 수 있습니다.

제출

- 제출은 Hanyang gitlab을 통해 이루어져야 합니다.
- 제출된 repository는 반드시 그림의 디렉토리 구조를 가져야 합니다. (os_practice가 repo폴더)
- xv6-public폴더의 이름이 지켜지면 되며, 파일이나 폴더가 추가되어도 상관없습니다.

```
os_practice/
├── extra
│   └── proj_shell
│       ├── Makefile
│       ├── shell
│       ├── shell.c
│       └── shell.o
└── xv6-public
    ├── append_this_to_your
    ├── asm.h
    ├── bio.c
    ├── bootasm.S
    ├── bootmain.c
    ├── buf.h
    ├── BUGS
    ├── cat.c
```

참고 자료

- Textbook
- 이론/실습 ppt
- MLFQ (http://pages.cs.wisc.edu/~remzi/OSTEP/cpu-sched-mlfq.pdf)