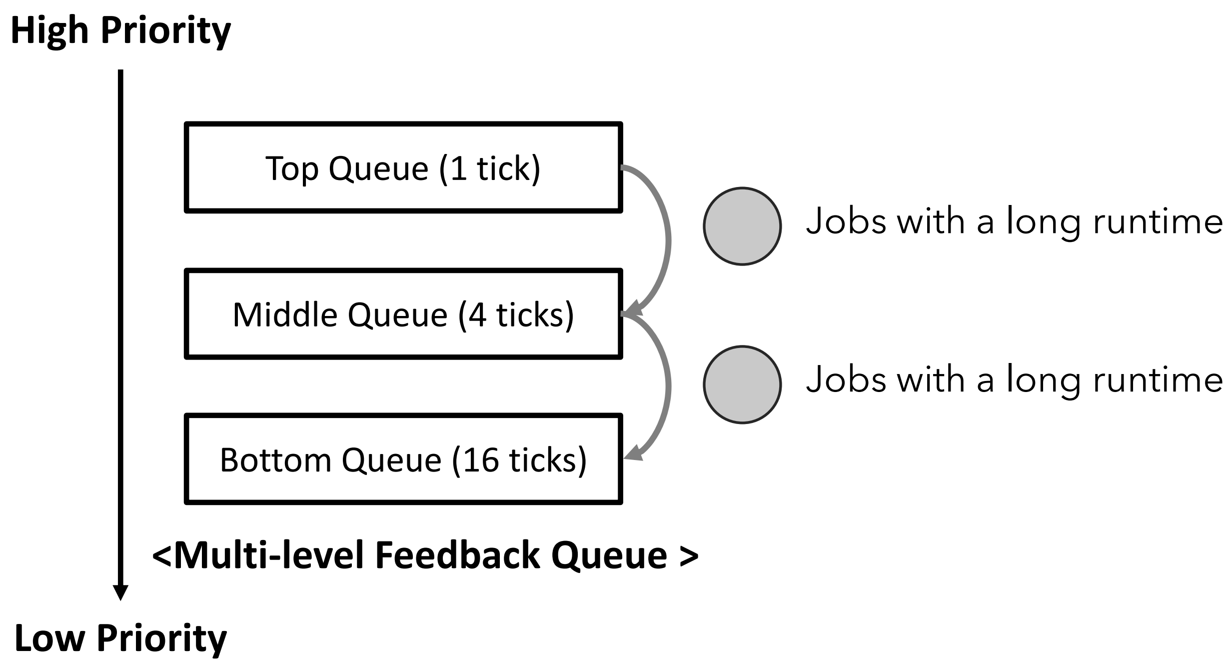
Operating Systems

Project 1 - Scheduler
Minjae Kim
jwya2149@dgist.ac.kr

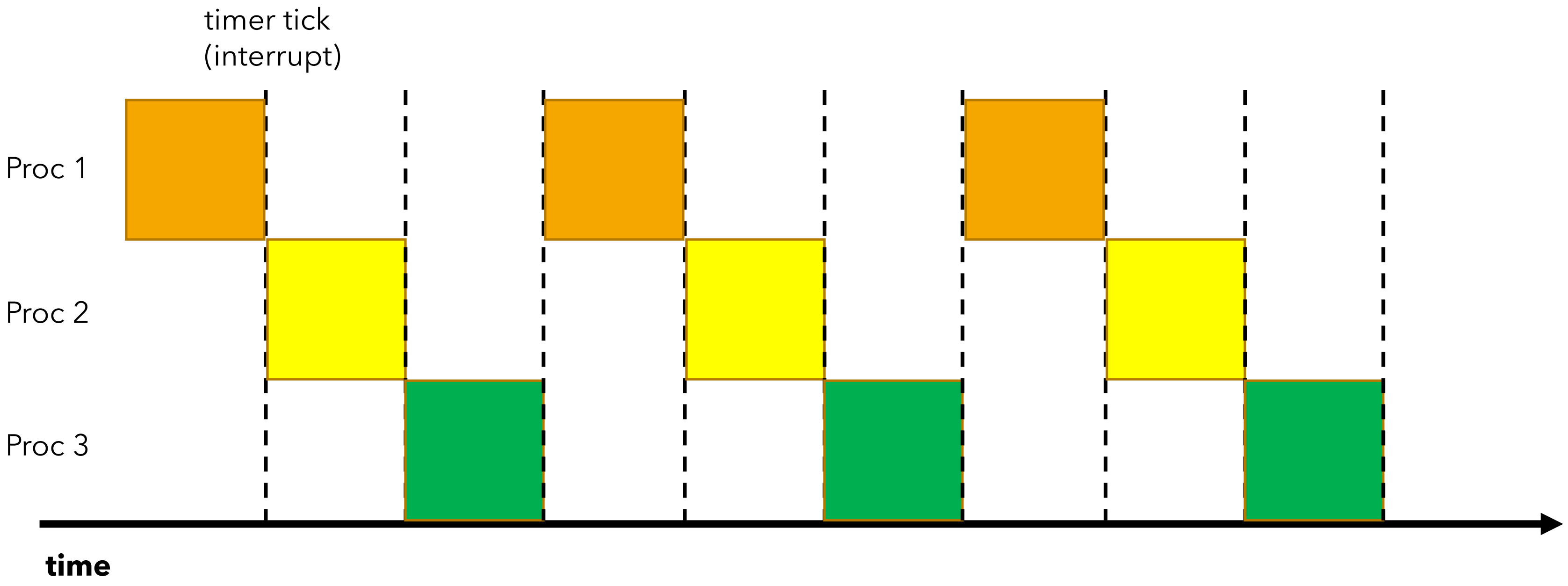
Project #1 - Implement xv6 MLFQ scheduler

- Implement a basic MLFQ scheduler in the xv6
  - Modify the default xv6 scheduler to Multi-level Feedback Queue



Project #1 - Implement xv6 MLFQ scheduler

- What is the default xv6 scheduler?
  - Round-robin scheduler
  - For each timer tick (~10ms), a timer interrupt occurs to incur a context switch
  - Change this round robin to the multi-level feedback!



Project #1 - Implement xv6 MLFQ scheduler

- Core function: *void scheduler() in proc.c*
- First *for loop* is looping forever
  - *This function never returns*
  - *Find a new process to be scheduled, run it until it yields*

Infinite loop (1 tick)

- Scan ptable to find RUNNABLE process

```
struct {
  struct spinlock lock;
  struct proc proc[NPROC];
} ptable;
```

- Switch from scheduler to the process
  - After the process yields by timer interrupt, come back to swtch()

<default round-robin scheduler code>

```
void scheduler(void) {
  struct proc *p;
  struct cpu *c = mycpu();
  c->proc = 0;

  for(;;){
    // Enable interrupts on this processor.
    sti();

    // Loop over process table looking for process to run.
    acquire(&ptable.lock);
    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
      if(p->state != RUNNABLE)
        continue;

      // Switch to chosen process.  It is the process's job
      // to release ptable.lock and then reacquire it
      // before jumping back to us.
      c->proc = p;
      switchuvm(p);
      p->state = RUNNING;

      swtch(&(c->scheduler), p->context);
      switchkvm();

      // Process is done running for now.
      // It should have changed its p->state before coming back.
      c->proc = 0;
    }
    release(&ptable.lock);
  }
}
```

Project #1 - Implement xv6 MLFQ scheduler

- Objectives of this project
  - Understand how context-switches are performed in xv6 code
  - Implement a basic MLFQ scheduler

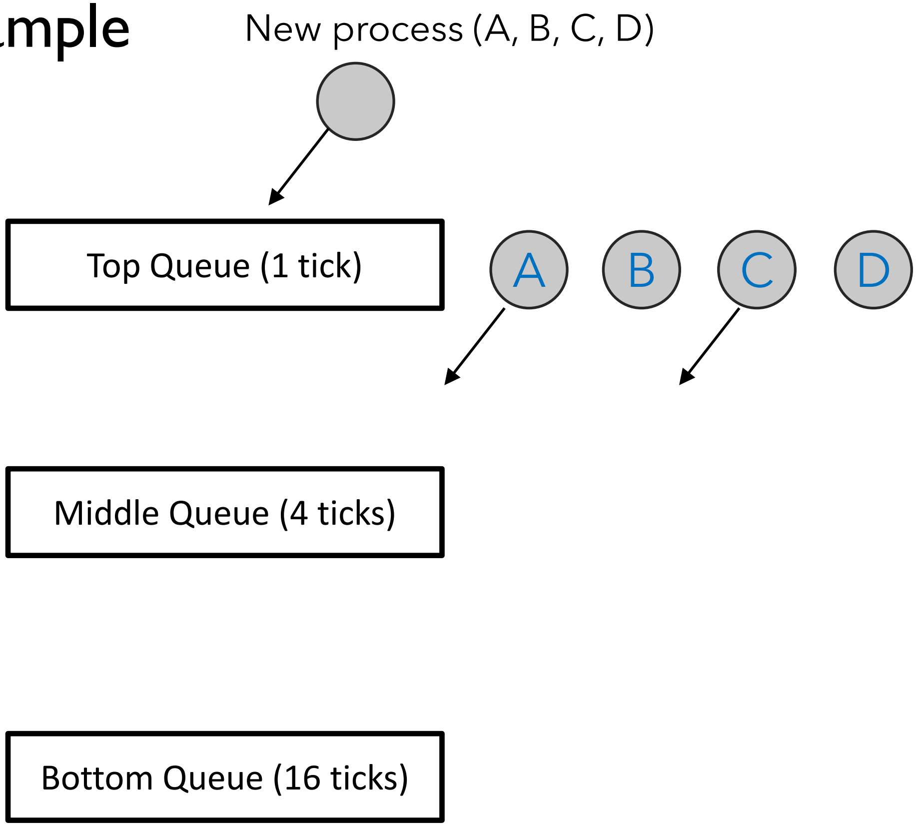
- Where to look and write code:
  - proc.c, proc.h (+ etc)

Project #1 - Implement xv6 MLFQ scheduler

- Rules for MLFQ scheduler
    1. There are 3 priority levels – top, middle, bottom
    2. Whenever a timer tick interrupt occurs, a process in the top queue is scheduled
        1. Similarly, a process in the top queue is scheduled after previously running process yields the CPU or exits
        2. When a new process is created, it starts from top queue
        3. When scheduling, find a process in the lower queue if there are no processes in the upper queue
    3. Processes in the same queue are scheduled by a round-robin policy
        1. However, the time slice of each queue should be considered
        2. Time-slice: # of ticks allowed to each process according to the priority level
    4. Time-slice of top/middle/bottom queue is 1 tick / 4 ticks / 16 ticks
    5. If a process consumes its time-slice, the process goes to down (except for bottom queue)

Project #1 - Implement xv6 MLFQ scheduler

- Example



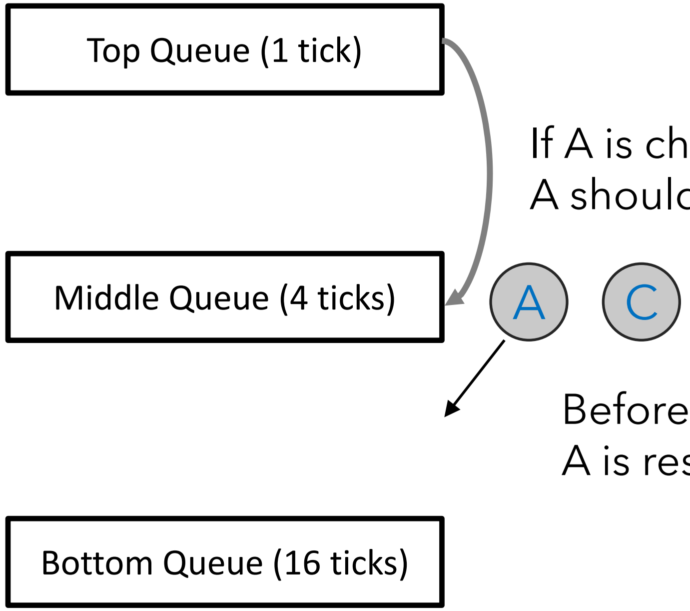
In the top queue, all processes should be scheduled at each timer tick

If a top process does not end in 1 tick, the process goes to the middle queue

Project #1 - Implement xv6 MLFQ scheduler

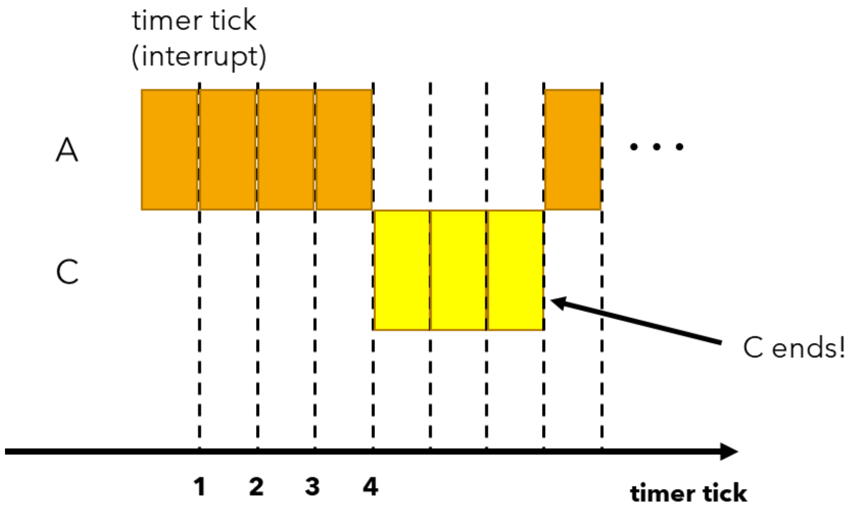
- Example

(Top queue is empty)



If A is chosen to be scheduled before C,
A should run for a complete time-slice (4 ticks)

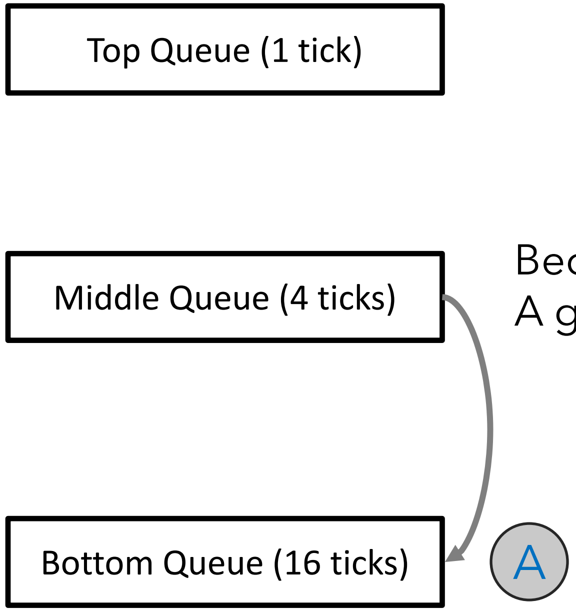
Before A uses its all time-slice,
A is rescheduled for each tick



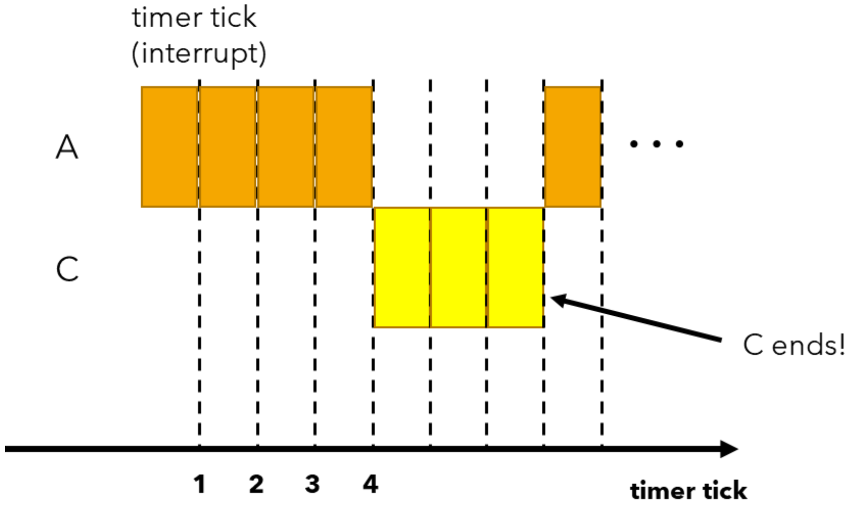
Project #1 - Implement xv6 MLFQ scheduler

- Example

(Top queue is empty)



Because A doesn't end after using its time-slice,
A goes to the bottom queue



Project #1 - Implement xv6 MLFQ scheduler

- Small tips
  - Don’t need too many changes in original code – understanding the scheduler flow is important
  - Don’t need linked-list queue – recommend to use fixed-sized arrays to represent each priority level
  - To understand more detail about xv6 scheduler, study Chapter 5 in the xv6 book (https://pdos.csail.mit.edu/6.828/2018/xv6/book-rev11.pdf)
  - To study MLFQ scheduler, see OSTEP Chapter 8 (https://pages.cs.wisc.edu/~remzi/OSTEP/cpu-sched.pdf)

- Build xv6 with ‘CPUS = 1’ flag to test easier (In Makefile)

```
ifndef CPUS
CPUS := 1
endif
```

Project #1 - Implement xv6 MLFQ scheduler

- Hints
  - per-process structure: struct proc in proc.h

```
// Per-process state
struct proc {
  uint sz;
  pde_t* pgdir;
  char *kstack;
rocess
  enum procstate state;
  int pid;
  struct proc *parent;
  struct trapframe *tf;
  struct context *context;
  void *chan;
  int killed;
  struct file *ofile[NOFILE];
  struct inode *cwd;
  char name[16];
};
```

- You can add member variables for per-process variable

  e.g.,) allowed ticks, level, …

Project #1 - Implement xv6 MLFQ scheduler

- Hints
  - New processes are initialized in allocproc()

```c
static struct proc*
allocproc(void)
{
  struct proc *p;
  char *sp;

  acquire(&ptable.lock);

  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++)
    if(p->state == UNUSED)
      goto found;

  release(&ptable.lock);
  return 0;

found:
  p->state = EMBRYO;
  p->pid = nextpid++;

  release(&ptable.lock);
```

- Scan ptable to find an empty space

  (Don't need to be modified)

- Good place to add codes for initialization
  - e.g.,) find empty space in top queue and add p

Project #1 - Implement xv6 MLFQ scheduler

- Hints

```c
void scheduler(void) {
  struct proc *p;
  struct cpu *c = mycpu();
  c->proc = 0;

  for(;;){
    // Enable interrupts on this processor.
    sti();

    // Loop over process table looking for process to run.
    acquire(&ptable.lock);
    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
      if(p->state != RUNNABLE)
        continue;

      // Switch to chosen process.  It is the process's job
      // to release ptable.lock and then reacquire it
      // before jumping back to us.
      c->proc = p;
      switchuvm(p);
      p->state = RUNNING;

      swtch(&(c->scheduler), p->context);
      switchkvm();

      // Process is done running for now.
      // It should have changed its p->state before coming back.
      c->proc = 0;
    }
    release(&ptable.lock);
  }
}
```

- In scheduler(), we need to find which process to schedule
  - Vanilla xv6 scans ptable, but we need to scan queues
  - When scanning queues, we should look at the top level to the bottom level in order
- After swtch(), we can distinguish if the process used just before is over or not
  - We need to modify # of ticks remaining in the process and manage queues according to the case

Project #1 - Implement xv6 MLFQ scheduler

- Testing
  - Run simple bench programs to test whether implemented scheduler works well
    - bench 1~3: fork 10 child processes with a fixed execution time (1/4/16 ticks)
    - bench 4: fork 10 child processes with different execution times ([0, 5, 10, 15, 20, 0, …] ticks)
    - bench 5: fork 50 child processes with 1 fixed + 49 different times (200 + [0,3,6,9,12,0, …] ticks)
  - To compile bench programs, you need to modify Makefile
    - e.g.,) add _bench1\, … under UPROGS=\
    - You can make & test any programs if you want
  - You need to modify param.h to test bench programs
    - #define NPROC 16 → 64

```
#define NPROC        64  // maximum number of processes
#define KSTACKSIZE 4096  // size of per-process kernel stack
#define NCPU          8  // maximum number of CPUs
```

```
UPROGS=\
	_cat\
	_echo\
	_forktest\
	_grep\
	_init\
	_kill\
	_ln\
	_ls\
	_mkdir\
	_rm\
	_sh\
	_stressfs\
	_usertests\
	_wc\
	_zombie\
	_mytest\
	_swtchtest\
	_realtest\
	_bench1\
	_bench2\
```

Project #1 - Implement xv6 MLFQ scheduler

- Deadline
  - ~ 2022.11.02 (Wed) 23:59
- Hand-in procedure
  - proj1_201812345.patch
    - Run the following command and upload proj1_201812345.patch
      - `git diff > proj1_201812345.patch`
    - Check the patch file with Notepad and confirm your modifications are in the patch file
  - Report
    - Submit an 1-2 page report
      - Free format (Korean/English)
      - Description of your implementation
      - Analysis of benchmark programs including comparison with xv6 default scheduler

Finally…

<u>Do NOT hesitate</u> to ask questions!

Setup & Mini Projects	Juhyung Park	arter97@dgist.ac.kr
Project #1	Minjae Kim	jwya2149@dgist.ac.kr
Project #2	Seonggyun Oh	sungkyun123@dgist.ac.kr
Project #3	Soyoung Han	zsally@dgist.ac.kr

Thank you!