CS 202: Advanced Operating Systems
University of California, Riverside

Lab #2: Modifying xv6 Scheduler

Due: 05/11/2022, Wednesday, 11:59 p.m. (Pacific time)

In this assignment, you will implement the lottery scheduler and the stride scheduler. The goal is to implement only **basic** operations of these schedulers (i.e., no ticket transfers, no compensation tickets, etc). Assume a **single CPU core** for xv6.

Background: xv6 scheduler

To get started, look at the file "kernel/proc.c". In there you will find `scheduler()`, a simple round-robin CPU scheduler implemented. Xv6 runs a periodic timer interrupt, and whenever the timer interrupt arrives (at every clock tick), the scheduler function is called to decide which process to run next. The default scheduler simply selects the next ready (runnable) process from the process list, achieving a round-robin scheduling policy with a time slice length equal to one tick. To implement other scheduling policies, you should change the logic there and add appropriate information in the process control block as needed.

Part 1: System Calls

Add two system calls:

```
int sched_statistics(void);
```

- This system call prints, for each process, 1) PID, 2) name in a parenthesis, 3) the ticket value, and 4) the number of times it has been scheduled to run (i.e., a rough estimation of the number of ticks used by each process).
- Below shows an example output format:

```
1(init): tickets: xxx, ticks: 21
2(sh): tickets: xxx, ticks: 15
```

```
int sched_tickets(int);
```

- This system call sets the caller process's ticket value to the given parameter. The maximum number of tickets for each process cannot exceed 5000.

Both system calls always return 0.

Part 2: Scheduler Implementation

Implement two scheduling policies: lottery and stride scheduling. The lottery scheduler needs a random number generator, which is not included in the xv6 code base. So use the following code for your implementation. Note: the output of `rand()` is in an unsigned short range.

```c
// pseudo random generator (https://stackoverflow.com/a/7603688)
unsigned short lfsr = 0xACE1u;
unsigned short bit;

unsigned short rand()
{
  bit = ((lfsr >> 0) ^ (lfsr >> 2) ^ (lfsr >> 3) ^ (lfsr >> 5)) & 1;
  return lfsr = (lfsr >> 1) | (bit << 15);
}
```

In case of the stride scheduler, use 5000 as a large constant ($stride_1$ in the paper and slides) to compute a stride value. Our goal is to implement the "Basic" stride scheduling algorithm in the paper (Sec 2.1). No need to implement the "Dynamic" and "Hierarchical" stride scheduling algorithms.

Assign a default ticket value to each process so that it can be scheduled even if its ticket was not explicitly assigned by `sched_tickets()`. Decide what default value to use by yourself.

How to compile:

Use preprocessor directives to switch between lottery and stride schedulers. For example:

```
......xv6 code......

#ifdef LOTTERY

your code for lottery scheduler

#endif

#ifdef STRIDE

Your code for strider scheduler

#endif

......xv6 code......
```

You need to modify the Makefile to let you can toggle it on/off in command line:

After lines 72-73 "`CFLAGS = -fno-pie ...... endif`", insert 2 new lines:

```
LAB2 = LOTTERY

CFLAGS += -D$(LAB2)
```

So your code should be compiled for the lottery scheduler by "`make clean; make LAB2=LOTTERY`" and for the stride scheduler by "`make clean; make LAB2=STRIDE`".

How to test:

To test your scheduler implementation, add the following user-level program as `user/lab2.c`.

```c
#include "kernel/types.h"
#include "kernel/stat.h"
#include "user/user.h"

#define MAX_PROC 10
int main(int argc, char *argv[])
{
    int sleep_ticks, n_proc, ret, proc_pid[MAX_PROC];
    if (argc < 4) {
        printf("Usage: %s [SLEEP] [N_PROC] [TICKET1] [TICKET2]...\n", argv[0]);
        exit(-1);
    }
    sleep_ticks = atoi(argv[1]);
    n_proc = atoi(argv[2]);
    if (n_proc > MAX_PROC) {
        printf("Cannot test with more than %d processes\n", MAX_PROC);
        exit(-1);
    }
    for (int i = 0; i < n_proc; i++) {
        int n_tickets = atoi(argv[3+i]);
        ret = fork();
        if (ret == 0) { // child process
            sched_tickets(n_tickets);
            while(1);
        }
        else { // parent
            proc_pid[i] = ret;
            continue;
        }
    }
    sleep(sleep_ticks);
    sched_statistics();
    for (int i = 0; i < n_proc; i++) kill(proc_pid[i]);
    exit(0);
}
```

This user program creates N child processes, assigns a different number of tickets to each process, waits for a specific amount of time, and then prints scheduling results of the child processes.

For example, if you want to test 3 processes with 30, 20, and 10 tickets respectively during 100 ticks, type:

```
$ lab2 100 3 30 20 10
```

(the first argument is the test time in ticks, the second argument is the number of processes to create, and the rest arguments are the number of ticks for each process).

This will print like below:

```
1(init): tickets: xxx, ticks: 21
2(sh): tickets: xxx, ticks: 15
7(lab2): tickets: xxx, ticks: 101
8(lab2): tickets: 30, ticks: 50
9(lab2): tickets: 20, ticks: 33
10(lab2): tickets: 10, ticks: 17
```

Since 3 processes were tested, the last three lines are what you have to focus on. Use the number of ticks to calculate the allocated ratio per process: (number of ticks per program) / (total number of ticks by all 3 processes). Ideally, the ratio of the first process is 1/2, the ratio of the second process is 1/3, and the ratio of third process is 1/6.

Compare the outputs from the lottery and the stride schedulers and check if they perform as expected.

Part 3: Experiments

See **Section 5** of the stride scheduling paper (http://dl.acm.org/citation.cfm?id=889650 or http://publications.csail.mit.edu/lcs/pubs/pdf/MIT-LCS-TM-528.pdf) on the evaluation of the two schedulers. Your goal is to reproduce the plots like **Figure 8** using your xv6 implementation (not simulation).

Run **two** experiments: (1) three processes with 30, 20, and 10 tickets (3:2:1 allocation as in Figure 8), and (2) two processes with 19 and 1 tickets (19:1 allocation). For each experiment, measure cumulative ticks (quanta) allocated to each process and draw plots for lottery and strider scheduling. Give a short discussion on how far they differ from the ideal allocation.

What to submit:

You need to submit the following:

(1) The **entire XV6 source code** with your modifications ('make clean' to reduce the size before submission)

(2) **A report (in PDF)** with detailed explanation what changes you have made (Part 1 & 2), the experiment figures and discussion (Part 3). Also, give a brief summary of the contributions of each member.

(3) A **demo video** showing that all the functionalities you implemented can work as expected, as if you were demonstrating your work in-person.
You don't need to demonstrate Part 1 and Part 3. Only show **Part 2** with "lab2 100 3 30 20 10" and "lab2 100 2 19 1" under both lottery and stride schedulers.

Grades breakdown:

- System calls (with an explanation in the report): 10 pts
- Working lottery scheduler (with an explanation in the report): 35 pts
- Working strider scheduler (with an explanation in the report): 35 pts
- Experiment figures and discussion: 20 pts

Total: 100 pts