Fall 2025

CSE 321 Lab Assignment 2

xv6 Lottery Scheduler

In this project, you will modify the xv6-riscv kernel to replace its default round-robin scheduler with a lottery scheduler.

Lottery Scheduling

In xv6, the traditional round-robin scheduler treats all runnable processes equally, offering time slices in a fixed rotation. However, if we require flexibility in assigning priorities while maintaining proportional resource sharing, we can implement lottery scheduling by allowing processes to **hold tickets**, which represent their share of CPU time.

Core Idea

- Each runnable process owns a certain number of **tickets**.
- At each scheduling decision:
    - The scheduler computes the total number of tickets among the **runnable** processes.
    - A random number is drawn in the range [1, total_tickets].
    - The process whose ticket range contains the winning number is selected to run.

This ensures a process with **T** tickets receives approximately **T / total_tickets** fraction of the CPU over time.

Example

Suppose there are three runnable processes:

Process	Tickets
P1	10
P2	5
P3	5

Total tickets = 20

P1 runs ~ 50% of the time
P2 runs ~ 25% of the time
P3 runs ~ 25% of the time

[Link to Sample Simulation]

xv6 Scheduling Overview

In xv6-riscv:

- The scheduler is implemented in **kernel/proc.c**, in the function **scheduler()**
- Processes are stored in the global **proc[NPROC]** table
- Only processes in the state **RUNNABLE** are eligible to run.

Tasks

Task 1: Add Tickets and Rounds to Processes

- You should modify the process structure to make it hold a certain number of tickets.
- The initial process, init, should have 10 tickets.
- **Inheritance:** Whenever a new process is forked, it should inherit all the tickets of its parent.
- You should also add a field to hold the number of rounds that a process has run inside the CPU, i.e. the number of times the scheduler allotted the CPU to the particular process. Each new process should have zero rounds after initialization.

Task 2: Add a System Call to Set Tickets

User processes should be able to change their ticket count. You should add a new system call

```
int settickets(int n);
```

where, n must be a positive integer, and sets the number of tickets of the calling process.

Task 3: Random Number Generation

xv6-riscv does not provide a random generator. Instead, we will implement a pseudorandom generator in the kernel. You should use the functions and variables

provided below to generate the pseudorandom numbers. Simply call the rand() method to generate a new random number each time.

```
int
do_rand(unsigned long *ctx)
{
    long hi, lo, x;

    x = (*ctx % 0x7ffffffe) + 1;
    hi = x / 127773;
    lo = x % 127773;
    x = 16807 * lo - 2836 * hi;
    if (x < 0)
        x += 0x7fffffff;

    x--;
    *ctx = x;
    return (x);
}

unsigned long rand_next = 1;

int
rand(void)
{
    return (do_rand(&rand_next));
}
```

Task 4: Replace Round-Robin with Lottery Scheduling

You should modify the existing scheduler using the following algorithm:

- Generate a random number winning within **[0, total_tickets - 1].** This will be the number of the winning ticket.
- Iterative over all processes to find the process with the winning ticket.
- Switch to the selected process.
- Increase the number of rounds of the process by 1.

Task 5: Modifying Procdump

xv6-riscv provides a procdump method (in **kernel/proc.c**) to check the details of the running processes using the procdump function asynchronously. You can use **Ctrl + P** to see its output. Modify the output so that it also shows the number of rounds each process has run.

Task 6: Testing Using a User Program

To test your new scheduler, you should:

- Create a user program test_scheduler that takes in a single command line argument: its ticket count as input.
- Call the previously implemented settickets method with this ticket count and set its ticket count accordingly.
- Spin in an infinite loop to keep the process alive.
- Call multiple instances of this program as separate **background processes** using the following method:

```
$ test_scheduler 10 &
$ test_scheduler 5 &
$ test_scheduler 2 &
```

- **Confirm that the rounds of the test_scheduler processes are approximately proportional to their ticket counts using Ctrl + P multiple times,** allowing a short interval of execution time between each process to observe how the counts evolve. The respective counts should converge to their ticket ratios with time.

Instructions

- You **must** start with a clean version of xv6-riscv. You can check whether the version is clean using the git status command. If you are yet unsure, then it is a good idea to clone the repository again and start afresh.
- **IMPORTANT: You must set CPUS := 1 in the Makefile.** This is the single-CPU implementation of the lottery scheduler, and you will not observe proper behavior if you don't perform this step.
- Upon completion of the assignment, **do not commit** to this git repository. Instead:
    - Use **git add .** to stage all changes.
    - Use **git diff --staged > YOUR_ID.patch** to create a patch file of your changes.
- Submit this patch file **only** as your submission.
- If you feel you have broken the xv6 code and are unable to get back to a stable state, you should reset your working git repository using **git stash -u**.
  WARNING: This step removes all new files created inside the repository, so if you have a working patch inside the repository, safely copy it to a different location before using the command.

- To ensure your submission works, move the patch file to a clean xv6-riscv repository, and run **git apply YOUR_ID.patch**. The repository should switch to the state of your submission, so you can check whether everything is working. Note that we will check your work using this method, so you should ensure proper functionality here.