Xv6 Scheduling

Updates: argptr() is now deprecated, refer to argaddr() for passing pointers to the kernel.

In this assignment, you will be implementing a new scheduler for the xv6 operating system. This scheduler will implement a *lottery scheduling* algorithm. The basic idea is quite simple. Each process will be assigned a fixed number of tickets (an integer number). A process will be probabilistically assigned a time slice based on its number of tickets.

More specifically, if there are n processes $p_1, p_2, \cdots, p_n$ and they have $t_1, t_2, \cdots, t_n$ tickets respectively at the beginning of a time slice, then a process p_i has a probability of $\frac{t_i}{\sum_{j=1}^{n} t_j}$ of being scheduled at that time slice. You will basically sample p_i based on the probability distribution derived from the ticket counts.

At the end of that time slice, t_i will be reduced by 1 (i.e., the process will have used that ticket). Hence, the probabilities will have to be recomputed at the beginning of each time slice using the updated ticket counts. Once the ticket counts of **all** processes become 0, their original ticket counts will be reinstated.

System Calls

You will need two system calls for your implementation:

- `settickets`

The first system call is `int settickets(int number)`, which sets the number of tickets of the calling process. By default, each process should get 1 ticket; calling this routine makes it such that a process can raise the number of tickets it receives and thus receive a higher proportion of CPU cycles. This routine should return 0 if successful and -1 otherwise (if, for example, the caller passes in a number less than one).

- `getpinfo`

The second system call is `int getpinfo(struct pstat *)`. This routine returns some information about all running processes, including how many time slices each has been scheduled to run and the process ID of each. You can use this system call to build a variant of the command line program `ps`, which can then be called to see what is going on. The structure `pstat` is defined below; note that you cannot change this structure and must use it exactly as is. This routine should return 0 if successful and -1 otherwise (if, for example, a bad or `NULL` pointer is passed into the kernel).

You will need to understand how to fill in the structure `pstat` in the kernel and pass the results to userspace. The structure should look like what you see here in a file, you will

README.md

```
#ifndef _PSTAT_H_

#define _PSTAT_H_

#include "param.h"

struct pstat {
    int pid[NPROC]; // the process ID of each process

    int inuse[NPROC]; // whether this slot of the process table is being used (1 or

    int tickets_original[NPROC]; // the number of tickets each process  originally h

    int tickets_current[NPROC]; // the number of tickets each process currently has

    int time_slices[NPROC]; // the number of time slices each process has been sched

};
```

```
#endif // _PSTAT_H_
```

Files

Most of the code for the scheduler is quite localized and can be found in `proc.c`; the associated header file, `proc.h`, is also quite useful to examine. To change the scheduler, not much needs to be done; study its control flow and then try some small changes.

Ticket Assignment

You will need to assign tickets to a process when it is created. Specifically, you will need to ensure a child inherits the same number of tickets as its parents. Thus, if the parent has 42 tickets and calls `fork()` to create a child process, the child should also get 42 tickets.

Argument Passing

Good examples of how to pass arguments into the kernel are found in existing system calls. In particular, follow the path of `read()`, which will lead you to `sys_read()`, which will show you how to use `argaddr()` (and related calls) to obtain a pointer that has been passed into the kernel. Note how careful the kernel is with pointers passed from userspace -- they are a security threat(!) and thus must be checked very carefully before usage.

For this particular offline set CPUS = 1 in the makefile of xv6.

Random Number Generation

You will also need to figure out how to generate (pseudo)random numbers in the kernel; you can implement your own random number generator or use any off-the-shelf implementation from the web. You must make sure that the random number generator uses a deterministic seed (so that the results will be reproducible) and is implemented as a kernel-level module.

Testing

You need to write two user-level programs, `testticket.c` and `testprocinfo.c` to test out the ticket assignment and scheduling, respectively. The `testticket.c` program should also have provisions to test forked processes.

Executing `testticket.c` followed by `testprocinfo.c` should print the relevant statistics defined in `pstat.h` like

```
PID | In Use | Original Tickets | Current Tickets | Time Slices
5     1        200000             149750            50250
3     1        70000              52521             17479
4     1        160000             118543            41457
6     1        400000             301103            98897
1     1        100000             76198             23802
```

Submission Guidelines (Deadline: January 17, 8:00 AM)

Start with a fresh copy of xv6 from the original repository. Make necessary changes for this offline. Like the previous assignment, you will submit just the patch file.

Don't commit. Modify and create files that you need to. Then create a patch using the following command:

```
git diff HEAD > studentID.patch
```

Where studentID = your own six-digit student ID (e.g., 1405004). Just submit the patch file, do not zip it. In the lab, during evaluation, we will start with a fresh copy of xv6 and apply your patch using the command:

```
git apply <studentID>.patch
```

Make sure to test your patch file after submission in the same way we will run it during the evaluation.

Please DO NOT COPY solutions from anywhere (your friends, seniors, internet, etc.). Any form of plagiarism (irrespective of source or destination) will result in getting -100% marks in this assignment. It is your responsibility to protect your code.

Releases

No releases published

Packages

No packages published