Assignment 5

Tweaking the Xv6 Scheduler

Deadline: **November 5th, Sunday, 9pm.**

This assignment is supposed to be done individually.

The goal of this assignment is to better understand the concepts of scheduling by making changes to the Xv6 scheduler. You will also learn how to implement system calls.

IMPLEMENTING A SYSTEM CALL – this will **not** be a part of the evaluation

Task: Learn how to implement a basic system call

Here we try to create a basic system-call that can be used when you try to implement your scheduler functions. The first step is to **extend the current proc structure** and add new fields **ctime, etime and rtime** for creation time, end-time and total time respectively of a process. When a new process gets created the kernel code should update the process creation time. The run-time should get updated after every **clock tick** for the process. To extract this information from the kernel add a new system call which **extends wait**. The new call will be

int waitx(int* wtime, int* rtime)

The two arguments are pointers to integers to which waitx will assign the total number of clock ticks during which process was waiting and total number of clock ticks when the process was running. The return values for waitx should be same as that of wait system-call. **Create a test program** which utilises the waitx system-call by creating a 'time' like command for the same.

TWEAKING THE SCHEDULER – implement **any one** of the follwing scheduling techniques:

Alternative 1: The Priority Based Scheduler

Now that you know how to add system-calls to Xv6, let's extend the idea. Replace the **current round robin scheduler** for Xv6 and replace it with a **priority based scheduler**. A priority based scheduler selects the process with highest priority for execution. In case two or more processes have same priority, we choose them in a round robin fashion. The priority of a process can be in the range **[0,100]**, **smaller value will represent higher priority**. Set the default priority of a process as **60**. To change the default priority add a new system-call *set_priority* which can change the priority of a process.

int set_priority(int)

The system-call returns the old-priority value of the process. In case the the priority of the process increases (the value is lower than before), then rescheduling should be done.
Hint : Think along the lines of using ***yield* system call**.
Submit a report with a small example which demonstrates the working of your scheduler, the report should include comparison of your current (priority based) scheduling policy and round robin approach.

Alternative 2: The Lottery Based Scheduler

Implement a lottery based scheduler for Xv6. The idea behind a lottery based scheduler is simple : Assign each running process **a slice** of the processor based in **proportion to the number of tickets it has**; the more tickets a process has, the more it runs. Each time slice, a randomized lottery determines the winner of the lottery; that winning process is the one that runs for that time slice.
For this the following system call:

int settickets(int num)

which sets the number of tickets of the calling process, could be used. By default, each process should get one 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 user passes in a number less than one).
Output : Submit the results in a graph like format for two process, where
numtickets(process_1) = 4*numtickets(process_2)
indicating number of time-slices a set of two process receive over time.

Alternative 3: The Multi-Level Queue Scheduler

Implement a Multi-Level Queue Scheduling which will have 3 queues:
First Queue : The first queue will hold the **high priority processes**. The processes in this queue will be scheduled according to the guaranteed scheduling policy (Implement any fair scheduling policy of your choice).
Second Queue : The second queue will hold **medium priority processes**. The processes in this queue will be scheduled according to the FIFO round robin scheduling policy.
Third Queue : The third queue will hold **low priority processes**. The processes in this queue will be scheduled according to the round robin scheduling policy.
A process with a higher priority will be preferred and run before a process with a lower priority. Add a system-call

int nice()

which can **decrease** the priority of a process (returns 0 on success, else -1) and can be used to demonstrate the results.
Hint : Modify the proc structure for storing the priority of every process.
Output : Submit the results in the form of a report for a small example which demonstrates the correct working of the scheduler. You may utilise the output of waitx system-call, and report the average turn-around time for each of the process.

General Notes

- You could fork https://github.com/Ras-al-Ghul/xv6-Assignment to look at how a system call (*waitx*) is implemented. The readme therein has a few important links. Try running '*time ls*'. Note that the repo was created quite some time back. The Xv6 code may changed in the meantime – so instead of directly replacing (copy-paste) the files in your Xv6 directory, make changes to them accordingly.
- Most distros include **qemu** as part of their repositories – hence an apt-get should do the job :-) Else, build from source :-(Then clone the **Xv6 repo**. In most cases, *cd Xv6* --> *make* --> *make qemu* should start the emulator with Xv6 running.
- Other useful links: <The PDF version of the Xv6 code> <The Xv6 book – read the Scheduling chapter> <A repo which might prove useful>
- In order to test your scheduler, try to create small code examples which forks a new process and assigns a unique number to each process created. For every process with say unique_id % 3 == x (can have some value), do a I/O intensive job, or decrement the nice value, or increase the priority. The type of jobs that you decide to do in each process should be fairly simple so that you can easily derive conclusions from them.
- Whenever you add new files do not forget to add them to the Makefile so that they get included in the build.

- The evaluation will be based on two criteria – i) working code which shows the scheduler in action – as in Xv6 must use your scheduler ii) the report and conclusions that you draw from the results of your experiments – and a few questions based on this.

Submission Guidelines

- Upload format **rollnum_assgn5.tar.gz**. Only the files that you change should be a part of the tar archive (including the Makefile) Make sure you write a **readme** which briefly describes the implementation for the alternative that you choose.
- **Do not copy**. Try to get a better understanding of how system calls are implemented and how scheduling works in OSes.

OSTAs, ICS231