Assignment 4 (Xv6-Project)

Deadline- 18th November 2015

Question 1:

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.

Question 2:

Now that you know how to add system-calls to Xv6, lets 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.

Challenge Problem:

* 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. 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.

OR

* 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 sys-call 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. Utilise the output of waitx system-call, and report the average turn-around time for each of the process.

General Comments :

1) 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.

2) In order to debug the kernel - Try the following link (http://zoo.cs.yale.edu/classes/cs422/2011/lec/l2-hw) and use “make qemu-gdb” and “gdb kernel” commands to start debugging. Set breakpoints in your gdb and to run the code type ‘c’ or ‘continue’ in your gdb.
3) Whenever you add new files do not forget to add them to the Makefile so that they get included in the build.
4) Make sure to have fun and gain some experience and insights on how an actual operating system works ;)