CS 333: Operating Systems Lab
Autumn 2017

Lab #8 Game of Processes

Goal

The current scheduler in xv6 is a very simple round robin scheduler. In this lab, you will modify the scheduler to take into account user-defined process priorities.

Before you begin

- Download the source code for xv6 from the following url:
  `http://www.cse.iitb.ac.in/~cs347/labs/xv6-public.tar.gz`
- Download the formatted xv6 source code here.
  All the sheet and line numbers mentioned in this lab refer to this.
  url: `http://www.cse.iitb.ac.in/~cs347/labs/xv6-rev8.pdf`
- The xv6 book is at: `http://www.cse.iitb.ac.in/~cs347/labs/xv6-book-rev8.pdf`

You must use C for this lab.

Task 0: Time logging

1. Add the following variables related to time logging in `struct proc` and update them at appropriate places. Time is calculated in terms of clock ticks. Look `uptime` system call implementation to understand how to use system clock ticks.

   (a) `arrival_time`: Time at which the process is forked/created.

   (b) `first_scheduling_time`: Time at which the process is scheduled for the first time.

   (c) `run_time`: Duration for which the process runs on the CPU. Note that the a process can be RUNNABLE for a long time, but its actual time on the CPU can be lesser. This variable captures total time spent on the CPU.

   (d) `exit_time`: Time at which the process completes the execution and calls the `exit()` function. Note that process can languish in `ZOMBIE` state for a non-deterministic amount of time, update this variable somewhere in the *exit* function before the non-determinism kicks-in.

   (e) `priority` The priority associated with the process, to be used by the scheduler. Default value is 1.

2. Print the following information when a process exit (in the `exit` function).
   `Exiting. pid=5, prio=5, AT=363, IWT=6, FST=369, ET=1136, RT=84, TWT=690.`

   Here, `AT` = arrival time, `IWT` = Initial wait time, `FST` = First schedule time, `ET` = Exiting time, `RT` = Run time, and `TWT` = Total wait time.
   `IWT` is the delay to schedule a process after it is created. `TWT` is the time for which the process was not running in the cpu, but was present.

Task 1: Attaching priority to processes

1. Add a new system call to xv6 to set process priorities - `setprio()`.
   When a process calls `setprio(n)`, the priority of the process should be set to the specified value. The priority can be any positive integer value, with higher values denoting higher priority.

2. Add another system call `getprio()` to read back the priority of calling process, in order to verify that it has worked.

3. **fork2(int priority):**
   Add a new system call `fork` which sets the priority of the child process. The implementation of `fork2` system call should be the same as existing `fork` function, and in addition initializes the priority of child process as part of implementation. Copy existing fork for this.

Test your implementations using the given test program and verify that the logging works correctly. Note we have not use changed the scheduler yet to use the priority.

Task 2: A simple priority based scheduler

Modify the xv6 scheduler to use the priority of each process to choose the next process to schedule. The scheduling policy picks the highest priority process which can be scheduled and schedules it to run. If there two or more process with same priority, select any of them. Once a process is scheduled, the scheduler should schedule the process again and again till the process completes, or blocks, or a higher priority process arrives.
If all processes have same priority, this policy roughly approximates FCFS.
Note: For implementation of this task, only update the `scheduler` function.

Task 3: A weighted round robin scheduler

1. This is another priority based scheduler which uses the priority of each process to implement a round robin scheduler. A round is one iteration over all `RUNNABLE` processes.
2. The scheduler schedules each process for number of ticks in proportion to its priority. Set the CPU ticks for each process to be equal to its priority, default ticks are 1, in each round.
3. Once a process completes its allocated number of CPU ticks or it blocks, the next `RUNNABLE` process in the process array is scheduled.
4. Note: If a process blocks before completing its share, assume that the process consumed its share in that round.

`Note:` Use the above test program to verify correctness of task2 and task3 implementations. Also change the priorities assigned for child processes in different order like increasing, decreasing, constant, etc. and verify outputs.

Outputs and report.txt :
Submit your outputs on running the sample test program in each of the above two tasks with increasing (priority = `i+1`), decreasing (priority = `N-i`) and constant (priority = 5) priority distributions. Explain your observations in the report. Also, run the test program on task3 with 20 child processes having priority in increasing order (ie. priority = `i+1`). Explain the output obtained.

Submission Guidelines

- All submissions via moodle. Name your submission as: <**rollno_lab8**>**.tar.gz**
- The tar should contain the following files in the following directory structure: <`rollno_lab8`>/
  |__task1/
  |__<all modified files in xv6>
  |__task2/
  |____`proc.c`
  |__task3/
  |____`proc.c`
  |__outputs/
  |____task2/
  |____task3/
  |__report.txt

- We will overwrite `task2/proc.c` and `task3/proc.c` on `task1` to evaluate task2 and task3 respectively. Also we will execute your submission over several test cases whcih are not given as part of sample test cases.

- **Deadline: Friday, 13th October 2017 - 05:00 PM**.

4. extra time.

- In the Makefile, try changing the `CPUS` variable to change the number of CPUs in xv6. Test your implementation of scheduler with more than one CPU and understand how it performs.
- Instead of the simple weighted round robin technique stated above, implement a deficit round robin scheduler. The allotted ticks to a process that are not consumed due to blocking are maintained as deficit and added to the credits of a process in the next round. Maintain a maximum credit limit for each process to accumulate credits.