Design Document

CMPS 111, Winter 2019

1. Goal

The primary goal for this project is to modify the FreeBSD scheduler and implement a new scheduler called the Lottery Scheduler. We also aim to understand how to build and compile the FreeBSD kernel.

2. Assumptions

We assume that the ‘nice’ value for a user time-sharing process is more than or equal to -20, where a bigger value means a higher priority. We also assume that the nice values will not be changed while running the scheduler.

3. Design

We will create a queue for all user time-sharing processes with each process having a certain number of ‘tickets’ (depending on its ‘nice’ value). The number of ‘tickets’ is then used to decide which process runs next. When the kernel chooses a process to run, we will select a process based on the probability corresponding to the number of ‘tickets’ that a process has out of the total number of tickets for all processes in the queue. We also want to intercept the kernel when it wants to add a process to the run queue -- instead we will place the process to our lottery queue.

4. Pseudocode

Function to choose a process from the lottery queue:

```
runq_choose(runq){
if (runq isn’t lottery queue)
  do old code
else
  select the single lottery queue
  x = random number from 1 to # of tickets in runq
  sum=0
  For each process in lottery queue
    sum+= tickets for process
    if sum > x
      return current process
```

}

Function to add elements to the lottery queue:

```
runq_add(runq, thread, pri){
  if runq is not lottery queue
    run old code
  else
    insert thread into lottery queue, sorted from lowest to highest nice value
    runq.tickets += thread.tickets
    runq.lotterysize += 1
}
```

Function to remove elements from the lottery queue:

```
runq_remove(runq, thread){
  if runq is not lottery queue
    run old cold
  else
    remove thread from lottery queue
    runq.tickets -= thread.tickets
    runq.lotterysize -= 1
  }
}
```

5. Benchmark

Our benchmark program to test whether or not the lottery queue scheduler was implemented properly is fairly straightforward. It consists of forking a process multiple times and having a loop run for a predetermined amount of time within each fork. We set the priority for each process to be a different number. This should show whether we implemented the lottery scheduler properly since all the processes are set to run the same loop for the same amount of time so the process that finishes first should be the one corresponding to the process with the highest number of tickets, which is the one with the highest priority. To know which program process finishes first the benchmark ideally would print out statements which would illustrate when a fork is done running. That along with the statistics printed by our scheduler would illustrate that our implementation of the scheduler is correct less the bugs mentioned below. The benchmark program is named HelloWorldLoop.c. We include a Makefile to compile it.

Unfortunately, in the current state of our scheduler we cannot properly use our benchmark to obtain meaningful data.

6. Bugs/Unfinished Work

The assignment instructions specified that lottery scheduling should only be implemented for non-root user processes. We were unable to separate the root and non-root processes, so the scheduler doesn't work quite as intended. As a result, it was not possible for us to properly test our benchmark code.