CMPS 111 Assignment 2: Lottery Scheduling in FreeBSD

David Trang, Thomas Wong
cruzid: `datrang,twong40`

CMPS 111, Winter 2019

1. Goal

The goal of the program is to implement a lottery scheduling for all times-haring processes in the FreeBSD operating system. Furthermore, whenever a processes is added or removed from the lottery queue, statistics of the events are printed. This includes (1) the event type, add or remove, (2) the size of the queue, (3) the smallest number of tickets of a process in the lottery queue, (4) the largest number of tickets of a process in the lottery queue, (5) the total number of tickets of all processes, (6) the number of tickets of the process added to or removed from the lottery queue.

2. Assumptions

We are assuming that we have to only worry about adding and choosing processes to the queue. Any subroutines and processes removal can be taken care by the original code. We are also assuming that putting our lottery adding in the following conditional in tdq_runq_add:

else if (pri <= PRI_MAX_BATCH)

This will take care of only user timeshare processes and not worry about root processes and other. Also in adding, we're assuming adding all the processes at td_rqindex of 30 will keep all the processes together.

We're assuming that changing the runq_choose_from to our lottery_choose will still run the same.

3. Design

The general approach we're taking is that from where the original scheduling add or removes time-sharing processes, we implement our lottery scheduling code. This will be the functions, tdq_runq_add and tdq_choose. We will use the original timeshare_runq and add the variable tickets to process struct which will hold the number of tickets a process has.

When we add a process, we will check if the process already been added and if not, we will give it an initial 500 tickets and call lottery_add to add it the queue. We will discuss lottery_add later on. When we remove a process, we call lottery_choose which we will also discuss later on.

The function lottery_add will take in a thread and add it to the runq at priority rqindex, 30. The function lottery_choose will just take in the runq. From there it will get the total numbers of tickets by recurring through the list. From there we will use (random() % (total_tickets + 1)) to randomly choose a thread. If there's no processes in the queue, it'll return NULL.

For the number of tickets given to each process, we give each one an initial 500 as we said before. Afterwards we use the process nice value and interactive score to adjust the number of tickets given. We take the respective values from sched_nice and sched_priority. Each process can have minimum of 1 ticket and maximum of 5000 tickets.

4. Pseudocode

```
void tdq_runq_add
    if(process hasn't initialized with initial 500 tickets)
        Give process initial 500 seconds
    /* Original Code */
    if(thread is timeshare)
        CALL lottery_add
        return
    /* Original Code */
    end

thread tdq_choose
    /* Original Code */
    thread = lottery_choose(timeshare runq)
    if (thread isn't NULL)
        Removing thread status print statement
        return thread
    /* Original Code */

void lottery_add
    set thread rq_index to 30
    insert thread to timeshare runq
    Adding thread status print statement

thread lottery_choose
    Get the total number of tickets in queue
    Generate random number
    win ticket = random number % (total_tickets+1)
    get winning process
    return winning process
```

5. Results

On running two different C programs with different priorities but same code, one with higher and one with lower. On average the higher priority process finishes faster than the slower process. The process with the higher priority usually finishes 17000 nanoseconds faster