Assignment 2 Design Document

Charles Tzou, David Lung, Payton Javete
CruzID: chtzou, dflung, pjavete

1 Goal

The goal of this assignment was to implement a lottery queue in FreeBSD by modifying and adding onto preexisting files and have it take care of user timeshare processes.

2 Assumptions

The program should only work with FreeBSD and is not meant to work as a replacement for the Mac or Windows kernel.

3 Design

Our approach to implementing Lottery Queue was to keep as much of the original code as possible, because although we had studied the different functions, we weren't sure if the deletion of a piece of code could break the system. We only deleted sections of the code if we were sure that they were not necessary. The only functions we added to the program were lotteryq_choose( ) and lotteryq_init( ) which had to be implemented differently from the original code.

4 Explanation

1. **tdq_runq_add( ):** We kept the majority of this function the same, but in the section adding processes to the timeshare run queue, we first check to see if the process is called by user or root. We do this by first calling sys_getuid(td, NULL) and then check to see if td->td_retval[0] != 0 (0 means that it is a root process which we handle in else). In the conditional, we assign our lottery queue to ts->ts_runq, and then call runq_add_pri with pri being 0. We assign the thread the number of tickets (20 + nice value) and then check whether the number of tickets assigned is the new min or max value for statistics. We increment the number of processes and the total number of tickets, and then at the end, call runq_add( ).
2. **tdq_choose( ):** We added another section to tdq_choose( ) for lottery queue where we first do td = lotteryq_choose(&tdq->tdq_idle) and then check to make sure td != NULL. If td != NULL, we return td. The rest of the function is the same.
3. **lotteryq_choose( ):** We started by copying the function of runq_choose( ), but we had to modify it to choose from the tickets rather than priorities. In order to choose a random number, we created a function called rng which uses random( ) to generate a random uint64_t modulos the total number tickets and then adds 1. Afterwards, we use

TAILQ_FOREACH( ) to go through the lottery queue and adds the number of tickets to each process to a counter (which starts at 0) until it is greater than or equal to the generated number. This generates our chosen process which we then subtract the length of the lottery queue by 1 and the total number of tickets by the number of tickets held by the process.

4. **runq.h:** We added four values to the struct runq: num_tickets, total_proc, max_tickets and min_tickets. These were used to help us keep track of the different values we need to print when we add or remove a process.
5. **Proc.h:** We added a uint64_t td_tickets to the struct threads so that we could assign tickets to processes.

4 Benchmarking

Although lottery scheduling is a fair algorithm as all processes are assigned tickets based on their priority (nice value), this does not guarantee that it is optimal. This is because although higher priority processes technically have a higher probability of running,

Probability of Running = Number of tickets held/ Total number of tickets

there is a possibility that the higher priority process simply is not chosen by the scheduler due to the ticket number being randomly generated. Therefore, the default scheduling algorithm in FreeBSD, round robin, provides a more optimal scheduling.