CMPS111 Asgn 2

Barron Wong & John Runnberg
bwong20@ucsc.edu jrunnber@ucsc.edu

February 07, 2019

Introduction

Assignment 2 take a look at the FreeBSD kernel and its process scheduling algorithm. The gives exposure on how to understand small portions of a complex system. It shows that in order to modify an complex system, one does not need to fully understand all aspects of it. Other items that this assignment touches on how operating systems deal multi-threaded processes; this includes consumer and producer functions, resource sharing, and data locking.

1 Modifications

For this assignment we altered three files of the FreeBSD operating system source:

src/sys/kern/kern_switch.c
src/sys/kern/sched_ule.c
src/sys/sys/runq.h

The changes primarily occur in kern_switch.c, with the other changes being made to communicate this change. We add a function, called runq_choose_lotto. The function will implement the lotto algorithm which will be discussed fully, shortly and be used to replace the runq_choose_pri when called on the timeshare runq in the tdq_choose function in sched_ule.c. This swap will replace FreeBSD's algorithm for choosing a thread, which is implemented (in the case of timeshare) in the use of the runq_choose_pri function in tdq_choose.

1.1 Lottery Algorithm

Our algorithm will use FreeBSD's runq structure to hold threads and we will iterate across it up to 3 times:

Firstly, checking for non-empty queue lists, using FreeBSD's built-in status bit array. For each non-empty queue list we will, for each entry in that list,

calculate the tickets the thread should have and add them to a running sum to calculate the total number of tickets.

Next, we will take a random number less than the total number of tickets. With this number well will iterate through the runq counting the tickets of each thread. Once we reach a thread which puts this count above the aforementioned random number we will have our thread.

Finally, we will go through the entire runq again and calculate some statistics such as: min, max, and size.

In regards to adding and removing from the runq: For this we are able to use FreeBSD's built in runq_add and runq_remove. They function fully in cooperation with our new choosing function.

2 Logging

Logging all lottery events are done by outputing directly from the kernel. Each time an item is placed or removed from the lotto queue a message is logged showing which event it was (Add/Remove) along with the following stats: Tickets held by the thread, the thread name, the memory address of the thread structure, the highest amount of tickets being held by a thread, and the lowest amount of tickets being held by a thread. Below is a screenshot of the kernels output.

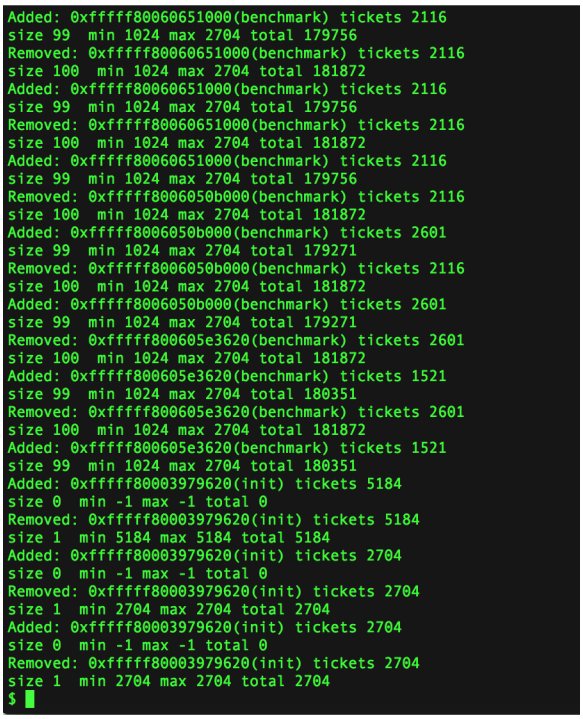

Figure 1: Logging

As it can be seen in the log, when a thread is added or removed from the queue, the queue stat that is printed, is a snapshot of the queue before the particular add or remove is processed. This design choice was selected, so that it would be easy to see the total amount of tickets in the queue at the time of a lottery selection. Our code also has a mode to produce a comma separated output, this is done by changing a defining a macro called "LOG_OUTPUT".

3 Benchmarking

Two program were written for bench marking purposes and a script was written in MatLab to analyze the data. The first program, called benchmark, simply runs a while loop indefinitely to keep the processor performing a task. The main stress program, called stress, creates multiple forks of the benchmark and tries to load the schedulers run queues. Stress can take in a parameter which will be the number of forks that it creates; if no parameter is set, the default value of 100 is set.

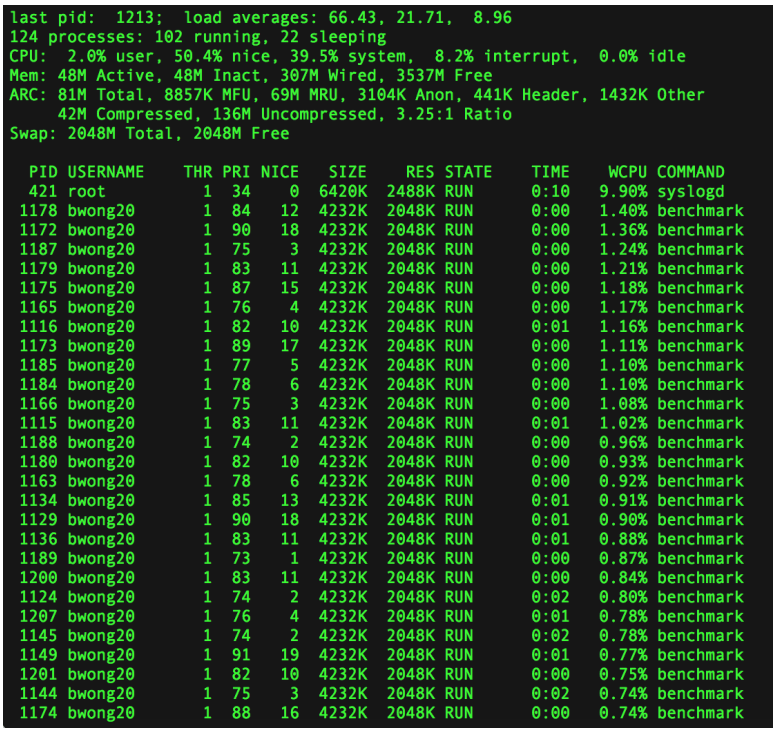

Figure 2: Benchmarking: Top

The figure above shows the output of top after the stress program is ran with the default settings. Over a short period of time its hard to determine how much CPU time each process is getting, to get a better snapshot, we ran our kernel in the LOG_OUTPUT mode as described above in the logging section; when benchmarking our kernel we decided to only look at the removed items from the queue, since we are only interested in the threads the lotto chooses.

messages.csv 〉 No Selection

```
1  992339488,490,10,410,490,4520
2  992339488,490,10,410,490,4520
3  992339488,490,10,410,490,4520
4  992339488,490,10,410,490,4520
5  993109536,430,10,410,490,4520
6  993109536,430,10,410,490,4520
7  993558528,470,10,410,490,4520
8  992337920,460,10,410,490,4520
9  993564192,410,10,410,490,4520
10 993564192,410,10,410,490,4520
11 993560096,490,10,410,490,4520
12 228492832,440,10,410,490,4520
13 228492832,440,10,410,490,4520
14 992342016,410,10,410,490,4520
15 993259520,440,10,410,490,4520
16 993562624,480,10,410,490,4520
17 993562624,480,10,410,490,4520
18 992339488,490,10,410,490,4520
19 993109536,430,10,410,490,4520
20 993558528,470,10,410,490,4520
21 993558528,470,10,410,490,4520
22 992337920,460,10,410,490,4520
```

Figure 3: Benchmarking: CSV

The above output is produced after the stress test runs for 4 minutes. All PIDs for child processes are killed by the parent process in the stress program and a CSV file is created using the output of dmesg. Once the CSV file is created, it can be imported into a MatLab script for analyzing and processing.

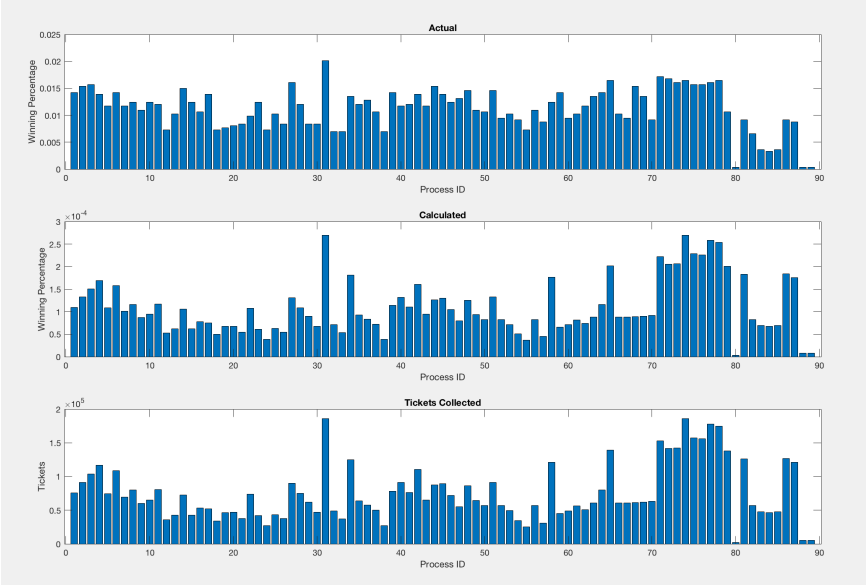

Figure 4: Lotto Stats

Above is the output produced by the MatLab script. The graph shows the

data for 90 different threads running at the same time. This is display data from roughly 3000 instances of a process getting pick by the lottery algorithm.

The first graph shows the actual percentage of each thread getting chosen. This is determined by taking the total number of times a particular thread is chosen and dividing it by the total number of lotteries that have taken place in the sample size.

The calculated graph is determined by the total number of tickets collected by each process divided by the total number of tickets distributed over the sample size. This graph closely resembles our actual graph will small differences, given enough time and samples it seems as though both graphs should converge.

The last graph is just display the amount of tickets collected by each of the processes. It can be seen that the amount of tickets collect is directly correlated to the probability of it being selected.

Conclusion

This assignment has given us exposure on the intricacies of the FreeBSD kernel, more specifically with its scheduling algorithm. Being able to understand the code has given us insight on where to modify certain variables and functions, so that they don't interfere with other functions in critical sections. Not only did we modify the FreeBSD kernel to implement a lottery scheduler, we also took statistics data from our scheduler and are able to confirm that it works as expected.