

# 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 we 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 outputting 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.

```

Added: 0xffff80060651000(benchmark) tickets 2116
size 99 min 1024 max 2704 total 179756
Removed: 0xffff80060651000(benchmark) tickets 2116
size 100 min 1024 max 2704 total 181872
Added: 0xffff80060651000(benchmark) tickets 2116
size 99 min 1024 max 2704 total 179756
Removed: 0xffff80060651000(benchmark) tickets 2116
size 100 min 1024 max 2704 total 181872
Added: 0xffff80060651000(benchmark) tickets 2116
size 99 min 1024 max 2704 total 179756
Removed: 0xffff80060650b000(benchmark) tickets 2116
size 100 min 1024 max 2704 total 181872
Added: 0xffff80060650b000(benchmark) tickets 2601
size 99 min 1024 max 2704 total 179271
Removed: 0xffff80060650b000(benchmark) tickets 2116
size 100 min 1024 max 2704 total 181872
Added: 0xffff80060650b000(benchmark) tickets 2601
size 99 min 1024 max 2704 total 179271
Removed: 0xffff8006065e3620(benchmark) tickets 2601
size 100 min 1024 max 2704 total 181872
Added: 0xffff8006065e3620(benchmark) tickets 1521
size 99 min 1024 max 2704 total 180351
Removed: 0xffff8006065e3620(benchmark) tickets 2601
size 100 min 1024 max 2704 total 181872
Added: 0xffff8006065e3620(benchmark) tickets 1521
size 99 min 1024 max 2704 total 180351
Added: 0xffff80003979620(init) tickets 5184
size 0 min -1 max -1 total 0
Removed: 0xffff80003979620(init) tickets 5184
size 1 min 5184 max 5184 total 5184
Added: 0xffff80003979620(init) tickets 2704
size 0 min -1 max -1 total 0
Removed: 0xffff80003979620(init) tickets 2704
size 1 min 2704 max 2704 total 2704
Added: 0xffff80003979620(init) tickets 2704
size 0 min -1 max -1 total 0
Removed: 0xffff80003979620(init) tickets 2704
size 1 min 2704 max 2704 total 2704
$ █

```

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.

```

last pid: 1213; load averages: 66.43, 21.71, 8.96
124 processes: 102 running, 22 sleeping
CPU: 2.0% user, 50.4% nice, 39.5% system, 8.2% interrupt, 0.0% idle
Mem: 48M Active, 48M Inact, 307M Wired, 3537M Free
ARC: 81M Total, 8857K MFU, 69M MRU, 3104K Anon, 441K Header, 1432K Other
     42M Compressed, 136M Uncompressed, 3.25:1 Ratio
Swap: 2048M Total, 2048M Free

```

PID	USERNAME	THR	PRI	NICE	SIZE	RES	STATE	TIME	WCPU	COMMAND
421	root	1	34	0	6420K	2488K	RUN	0:10	9.90%	syslogd
1178	bwong20	1	84	12	4232K	2048K	RUN	0:00	1.40%	benchmark
1172	bwong20	1	90	18	4232K	2048K	RUN	0:00	1.36%	benchmark
1187	bwong20	1	75	3	4232K	2048K	RUN	0:00	1.24%	benchmark
1179	bwong20	1	83	11	4232K	2048K	RUN	0:00	1.21%	benchmark
1175	bwong20	1	87	15	4232K	2048K	RUN	0:00	1.18%	benchmark
1165	bwong20	1	76	4	4232K	2048K	RUN	0:00	1.17%	benchmark
1116	bwong20	1	82	10	4232K	2048K	RUN	0:01	1.16%	benchmark
1173	bwong20	1	89	17	4232K	2048K	RUN	0:00	1.11%	benchmark
1185	bwong20	1	77	5	4232K	2048K	RUN	0:00	1.10%	benchmark
1184	bwong20	1	78	6	4232K	2048K	RUN	0:00	1.10%	benchmark
1166	bwong20	1	75	3	4232K	2048K	RUN	0:00	1.08%	benchmark
1115	bwong20	1	83	11	4232K	2048K	RUN	0:01	1.02%	benchmark
1188	bwong20	1	74	2	4232K	2048K	RUN	0:00	0.96%	benchmark
1180	bwong20	1	82	10	4232K	2048K	RUN	0:00	0.93%	benchmark
1163	bwong20	1	78	6	4232K	2048K	RUN	0:00	0.92%	benchmark
1134	bwong20	1	85	13	4232K	2048K	RUN	0:01	0.91%	benchmark
1129	bwong20	1	90	18	4232K	2048K	RUN	0:01	0.90%	benchmark
1136	bwong20	1	83	11	4232K	2048K	RUN	0:01	0.88%	benchmark
1189	bwong20	1	73	1	4232K	2048K	RUN	0:00	0.87%	benchmark
1200	bwong20	1	83	11	4232K	2048K	RUN	0:00	0.84%	benchmark
1124	bwong20	1	74	2	4232K	2048K	RUN	0:02	0.80%	benchmark
1207	bwong20	1	76	4	4232K	2048K	RUN	0:01	0.78%	benchmark
1145	bwong20	1	74	2	4232K	2048K	RUN	0:02	0.78%	benchmark
1149	bwong20	1	91	19	4232K	2048K	RUN	0:01	0.77%	benchmark
1201	bwong20	1	82	10	4232K	2048K	RUN	0:00	0.75%	benchmark
1144	bwong20	1	75	3	4232K	2048K	RUN	0:02	0.74%	benchmark
1174	bwong20	1	88	16	4232K	2048K	RUN	0:00	0.74%	benchmark

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.

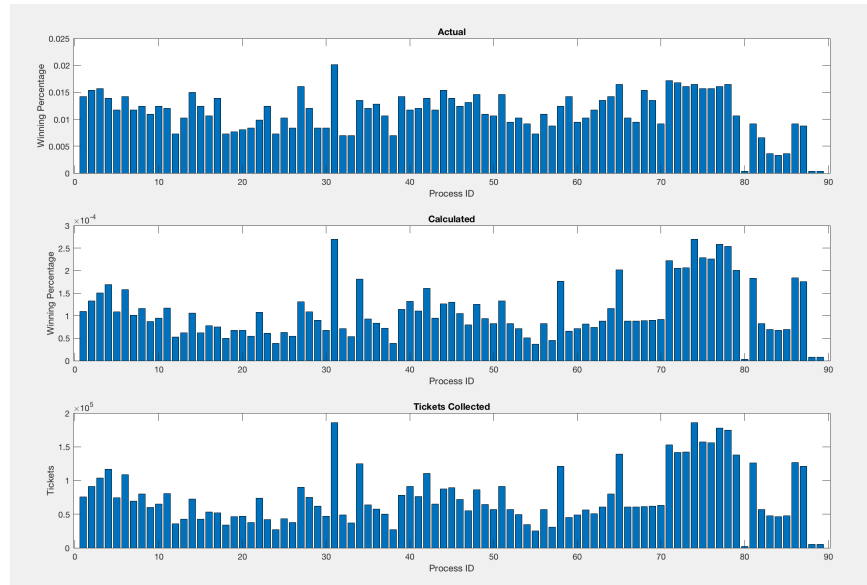


Figure 4: Lotto Stats

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 with 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.