Advanced Operating Systems – Lab 1

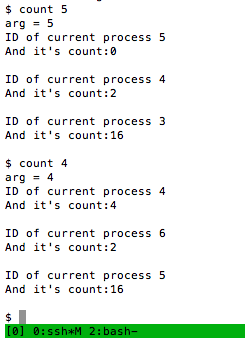
Team members:

Dennis Chen 861242096

Zhencong Li 861243465

**In part-1**, the files we changed including Makefile, proc.c, proc.h, syscall.c, syscall.h, sysproc.c, user.h and sys.S. Besides, we add some files including count.c, hash.c and hash.h.

The test file used in part-1 is count.c. In part-1, we implement a system call that can print accumulate counter of specific processor. For example, using “count 5” to print the number of system call that processor id 5 have been called. And the test results are as follow:



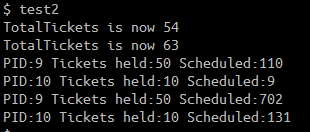
The test result shows that we have tested twice the counter of specific processor, currently processor and the child processor of currently processor. In the first time, processor 5 is not running at that time, so it’s total number of system call is 0. The processor 4 is current processor, which have run only 2 system calls, and processor 3 is the child processor, which has already run 16 system calls. Then, in test 2, we run count 4. Since processor 4 already run in the last test, so the final count of processor is 4 (2 more system calls at the end of test 1). For child processor, since it may experience many system calls that incurred form fork, so that it has much more system calls.

**In part 2** of the project, we’ve made changes to the files Makefile, defs.h, proc.c, proc.h, syscall.c, syscall.h, sysproc.c, user.h, usys.S

And added the files myRand.h, test.c, test2.c

Most of our modifications were in proc.c, where we changed proc to also hold an integer tickets, and the scheduler to then scan through the list of processes regardless of them being runnable or not, and finding the winner to run. proc.h and myRand.h were modified/added to facilitate this. syscall.c, syscall.h, sysproc.c, user.h, usys.S were modified to include a system call to set number of tickets, and to print out the status of a process.

Test2.c is our main test file, in which we create two child processes from a parent, and after setting both up, running them for a given amount of time before killing them and printing their scheduling characteristics. Below is a sample output. The first two lines prompt when the number of tickets printed by the kernel has changed. The next set of two are status system calls from the parent, inquiring about the status of its children. It shows the number of tickets that child holds, and the number of times the child has been scheduled up to this point. The last set of two is printed after a certain time has elapsed and both children are killed. Taking the difference between the first set and second, we can deduce the number of times each process was scheduled within that time frame.



We ran the test for different time intervals with 20 tickets for each process, and as the graph below shows, we found that at small timeframes, the scheduled ratios are further apart, but after 200ms, they tend to be similar.

Running the test with 10 tickets as the base amount, for 500ms produces the graph below. We can see that the error between targeted ratio and observed ratio is small.