Part 2

February 1, 2016

.

3.1 Lottery Scheduler

In the second part of the lab we had to implement the lottery scheduler. In the lottery scheduler, the user assigns to each process a number of tickets. We decided to assign one ticket by default to each process. For a process to win the lottery, it should have the ticket that the lottery generates. We generate the winning ticket using a random number generator function. To find the winner, we iterate through the tickets of all the processes until we find the winning ticket.

Now that we described the procedure of the lottery scheduler, we need to explain which parts of the code we changed to implement it. Firstly, we assigned the default numbers of tickets (to one) in the function allocproc of the proc.c file. All the other changes were implemented in the function scheduler of the proc.c file. At first, we iterated through all the processes, to find the total of all the tickets (getTotalTickets function in the proc.c). Then, we implemented a second iteration in which we look for the winner process. We find the number of tickets accumulated till the process that we currently are and we check if this number is greater or equal to the random number that the random number generator has produced (randomGen function in the proc.c). If this is true, then we found the winner process, otherwise we continue our search.

3.2 Ticket Inflation

In this part of the assignment we implemented the ticket inflation. In the ticket inflation, if the user notices that a process that she needs does not win as much as she wants, she could increase increase the number of its tickets. We do that by adding a system call which increases the number of tickets of the process that invokes it.

Our system call is called sys_numtickets and it is implemented in the sysproc.c file. We increase by one the number of tickets that the process already has and we return its current ticket numbers. We also had to make this system call visible to the syscall.c file. Then, we added it in the syscall.h, user.h, defs.h and usys.S files.

4 Tests

We design two tests to evaluate the validity of our implementation. The main idea is to run processes which perform long loops, allocate different amount of ticket to each of them and output their performance in fixed intervals of time. We implement a number of system calls in the sysproc.c file to achive these goals.

4.1 System calls

A first system call (top) was implemented to run the procdump function every 5 seconds. We modify the default procdump function (at the end of proc.c) to output information about the running and runnable processes. Specifically, it shows the pid, name, tickets and number of lottery wins of each of the process. This system call receives as parameters the number of requested outputs and its number of tickets. We use the first parameter to set how many samples we want to collect during the test.

A second system call (secs) will return the instant in which this system call was invoked. It uses the cmostime function and structs provide by the date.h and lapic.c files. For convinience, it just returns the number of seconds during the current hour.

A third system call (work) will recieve two parameters: a number of iterations for the loop and a number of tickets to be assigned for the current process. It just will performs a simple asignation during the number of specified iterations.

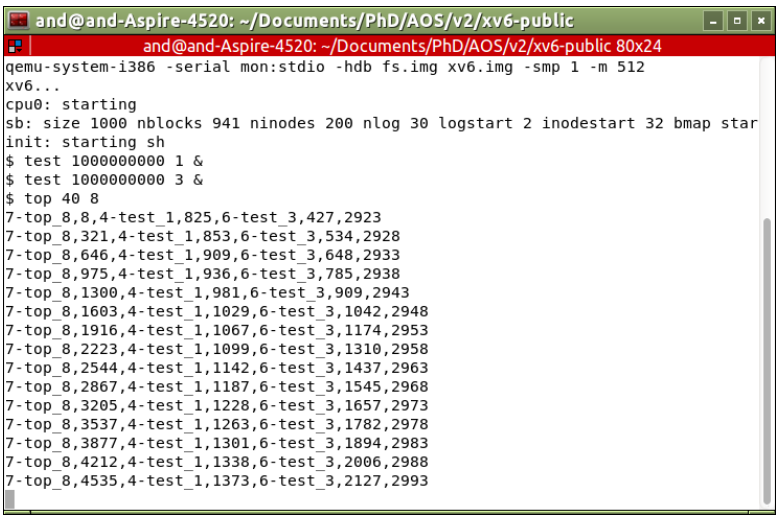

Figure 3: Details of the settings for test 1.

4.2 Test 1. Ticket allocation and fairness share

The first test evaluates the ratio of the ticket allocation. We implement a command line function (file test.c) that runs the system call work. Three processes were launched with different number of tickets running at the same time (using the & wildcard). The selected ratio was 8:3:1. Figure 3 shows the details of the setting for this test. Figure 4 shows the number of lotteries each process wins during a 200 seconds test.

In order to test the fairness share of the allocation, we plot the proportion of lottery wins for each process. For a ratio of 8:3:1 the expected proportions are 0.666, 0.250 and 0.083. Figure 5 illustrates the results. We can see that at the beginning of the test the proportions suffer some fluctuations but, as the number of lotteries increase, the proportions remains constant around the expected values.

4.3 Test 2. Ticket inflation

To test ticket inflation we implement a command line function (file test2.c) which runs a long loop. We track the number of iterations inside the loop and call the numtickets system call (section XX) every 1000000 iterations to increase its number of tickets. Then, we launch two process: test2 (starting with 2 tickets) and top, which remains with a fixed number of tickets. Figure 6 shows the settings for this test.

Figure 7 illustrate the number of lottery wins for the two processes. We can see that the process with ticket inflation increases faster its number of wins according to the increase in its number of tickets. The probability of the fixed-ticket process of winning lotteries start to decrease compared with the increase number of tickets of the inflation process.

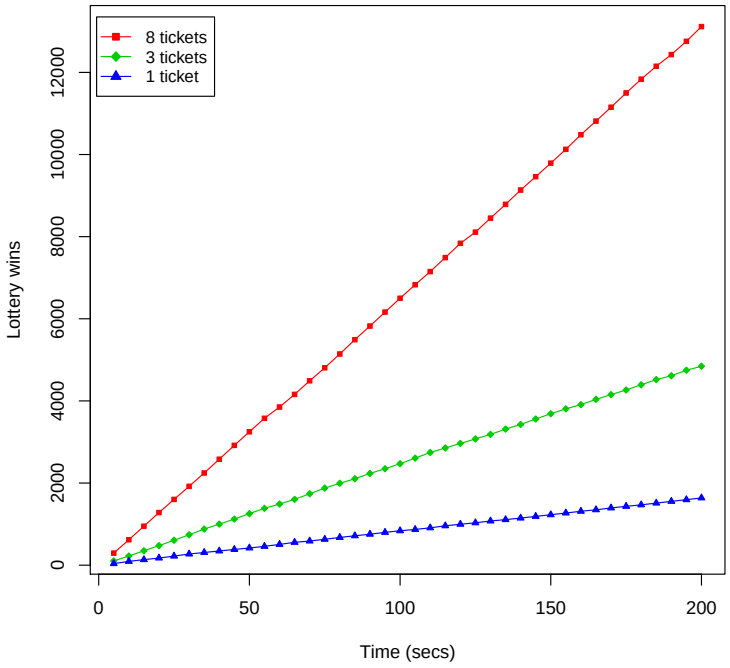

Figure 4: Lottery wins for three different processes. The proportional ration was 8:3:1.

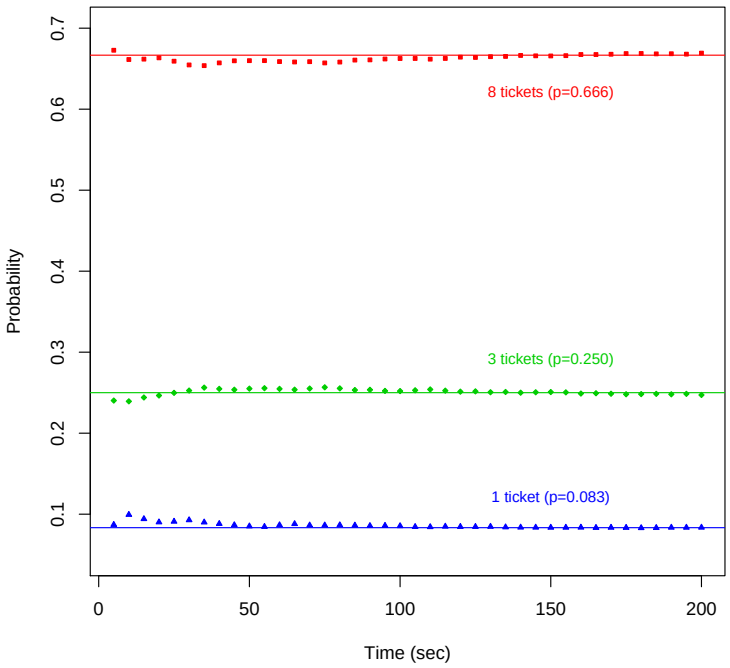

Figure 5: Lottery wins for three different processes. The proportional ratio was 8:3:1.

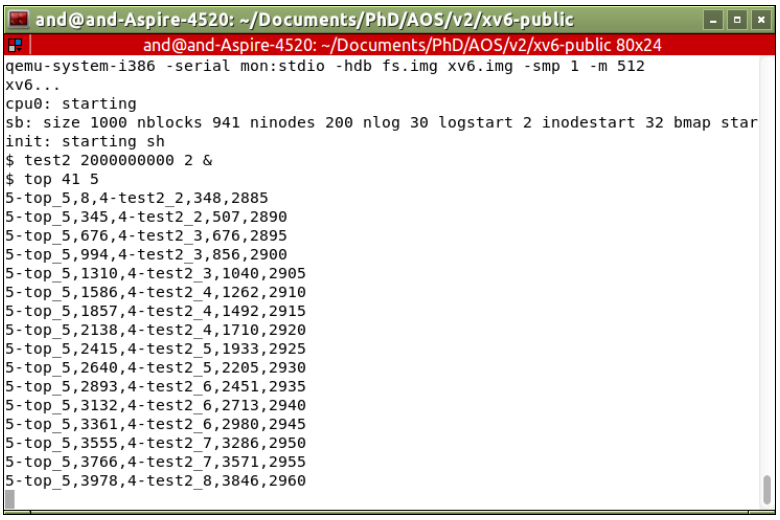

Figure 6: Details of the settings for test 2.

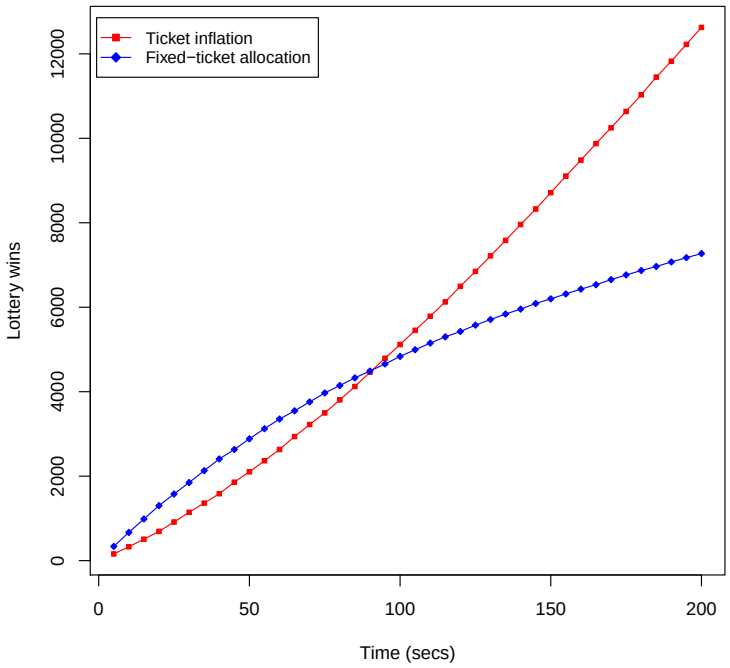

Figure 7: Response from two processes with different ticket allocation (Fixed vs Inflation).