CS 503 Lab 2

Experimenting with Mutual Exclusion

*Shuheng Guo*

Content

1. Introduction 3

2. Implementation and Explanation 4

2.1 variables 4

2.2 Three Different Exclusive Functions 5

2.3 Extra Credit 6

2.4 Start\_tests 6

3. Results 9

4. Conclusion and Discussion 11

# 1. Introduction

This lab is a follow up of lab 1. In lab 1 I used semaphores to enforce exclusive access to the shared resource. In this lab, there will be three different method used to realize mutual exclusive. They are as follows:

* Semaphores
* Spin lock, using the compare and exchange (CMPXCHG) instruction
* Disabling/Restoring interrupts.

Even though all of the three methods will do the job, but they may differ with each other in performance.

* Semaphores - Consider a semaphore with an initial value of 1, when a process waits on the semaphore the value is decremented and the process is allowed to continue executing if the value is >= 0. If the value is less than zero, the process is put to sleep until another process signals the semaphore.
* Disabling/Restoring interrupts - The XINU scheduler uses the clock interrupt to give each ready process a time slice of the CPU. When interrupts are disabled (using the disable() call), the clock interrupt is never sent so the XINU scheduler is never executed. As such, the process is allowed exclusive access to the CPU and any resources it wishes to use until interrupts are re-enabled.
* Spin lock using the CMPXCHG instruction - The compare and exchange instruction (CMPXCHG) is a special instruction in the x86 architecture that compares a register with a memory location and if the two are equal it sets the value of memory to a new value. This is done as an atomic operation.

# 2. Implementation and Explanation

## 2.1 variables

Figure 1. Variables in Lab2

|  |  |
| --- | --- |
| Variables | Explanation |
| global | The variable to be added by three methods in every process |
| general | Semaphore used to start all |
| Sem | Semaphore for method use semaphore to realize mutual exclusive |
| Lost | time lost in problem extra credit 1 |
| mutex | Mutex used for method spin lock |
| Rate | Iterate rate to measure performance of the three method |
| Count[] | Array of count by each process in problem extra credit 2 |
| Func\_array | Function array to restore different functions |
| Func\_name | Array of function name |
| Pid[] | Array of process ID |

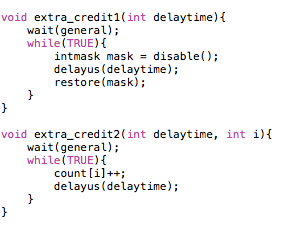
Table 1. Variables in Lab2

## 2.2 Three Different Exclusive Functions

Figure 2. Three different Functions in Mutual Exclusive

The first function I used here is diable\_restore. When interrupts are disabled (using the disable() call), the clock interrupt is never sent so the XINU scheduler is never executed. As such, the process is allowed exclusive access to the CPU and any resources it wishes to use until interrupts are re-enabled.

The second function I used here is semaphore. When a process waits on the semaphore the value is decremented and the process is allowed to continue executing if the value is >= 0. If the value is less than zero, the process is put to sleep until another process signals the semaphore.

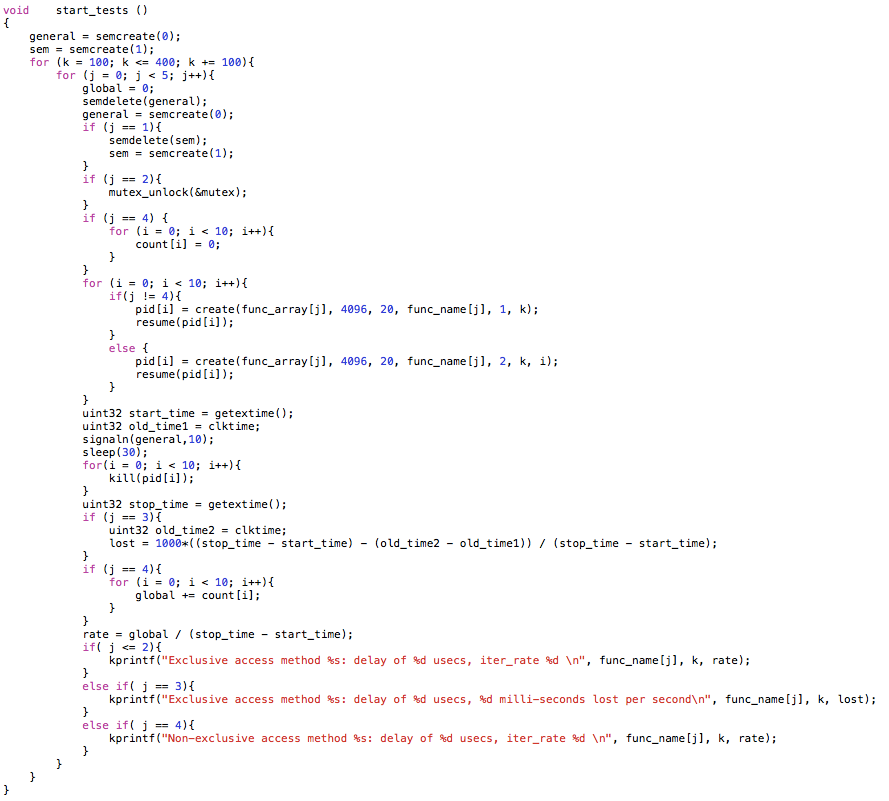
The third function I used here is Spin lock. The compare and exchange instruction (CMPXCHG) is a special instruction in the x86 architecture that compares a register with a memory location and if the two are equal it sets the value of memory to a new value. This is done as an atomic operation.

## 2.3 Extra Credit

Figure 3. Extra Credit

The function extra\_credit 1 is to cost the time lost in system clock. Every process just use disable-interrupt to contribute to the total time lost. The function extra\_credit 2 is to calculate the overhead of every exclusive function and nonexclusive function. I just made every process to increase their own counter every time and sum them together to get the iterate rate of nonexclusive method. The result will be shown in chapter 3.

## 2.4 Start\_tests

Figure 4. Function start\_tests

This function starts the test. First, I initialize all the variables and semaphores. Using three for loops to create 10 processes for each function with different delay time. If conditioning sentences are used to deal with different situations of corresponding specific functions. Clktime and getextime() are used to get system time without/with interrupt disable time, respectively. The total time lost on interrupt disable can be calculated out by a simple expression. After 30 seconds, I killed all processes and calculate iterate rate of every function to be printed out.

# 3. Results

After running my lab2 code on XINU, I got the result as follows:

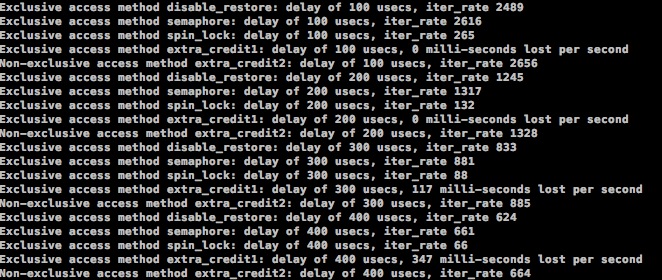


Figure 5. Lab 2 Result

From Figure 5 we can see that the result is reasonable. The iterate rate of method disable\_restore and semaphore are almost the same but the method spin\_lock is much lower than the first two method. As for extra\_credit1 method, milli-seconds with delay time of 100us and 200us are too small to be shown in milli-seconds level. Lost time with delay time of 300us and 400us are 117ms and 347ms respectively. Iterate rate of nonexclusive method extra\_credit2 should be larger than any exclusive method used before. And that is the case.

To calculate the overhead:

* Delay\_time = 100us

1. Disable\_restore: 6.29%
2. Semaphore: 1.51%
3. Spin\_lock: 90.02%

* Delay\_time = 200us

1. Disable\_restore: 6.25%
2. Semaphore: 0.83%
3. Spin\_lock: 90.06%

* Delay\_time = 300us

1. Disable\_restore: 5.88%
2. Semaphore: 0.45%
3. Spin\_lock: 90.06%

* Delay\_time = 400us

1. Disable\_restore: 6.02%
2. Semaphore: 0.45%
3. Spin\_lock: 90.06%

# 4. Conclusion and Discussion

After doing lab2, answer the following questions:

* Q: Which method for mutual exclusion provided the shortest wait time?
* Answer: From the lab result we can see that the performance of first two methods ( disable\_restore and semaphore) are almost the same. The method semaphore is a little bit better.
* Q: Can you think of other ways to measure the performance of mutual exclusion methods? Explain.
* Answer: To measure the performance of mutual exclusion we can get the variable global to a reasonable value and break the while loop whenever global reach to this preset value. Measure the total time spend in this process by function getextime(). The method with least time duration will have better performance.
* Q: Are there more methods for mutual exclusion that you can think of other than the three you tested? How would the performance compare to the three you tested?
* Answer: There are some other algorithms that can realize mutual exclusive:

1. Dekker’s algorithm: If one process is already in the critical section, the other process will busy wait for the first process to exit. This is done by the use of two flags, flag[0] and flag[1], which indicate an intention to enter the critical section and a turn variable that indicates who has priority between the two processes.
2. Peterson’s algorithm: Peterson's algorithm (AKA Peterson's solution) is a concurrent programming algorithm for mutual exclusion that allows two processes to share a single-use resource without conflict, using only shared memory for communication. It was formulated by Gary L. Peterson in 1981. While Peterson's original formulation worked with only two processes, the algorithm can be generalized for more than two.
3. Lamport's bakery algorithm: Lamport's bakery algorithm is a computer algorithm devised by computer scientist Leslie Lamport, which is intended to improve the safety in the usage of shared resources among multiple threads by means of mutual exclusion.

Reference : http://en.wikipedia.org/wiki/Mutual\_exclusion