ECE/CS 5510 Multiprocessor Programming
Homework 2

Mincheol Sung

September 24, 2018

Part I

Problem 1.
(a) It violates safety property because threads with same value of label can enter critical section at the same time.
(b) It can cause deadlock because threads with same value of label will be block by each other.

Problem 2.
(a) No. Even though the process crushes in the critical section, other processes still can't enter the critical section. However, this is not the case when OS interferes. If OS terminates the crashed process, safety can be broken.
(b) Yes. The crashed process keeps occupying the critical section. Unless OS terminates it, others will starve.
(c) No. Definition of bounded waiting: there exists a bound, or limit, on the number of times **other processes** are allowed to enter their critical sections after a process has made request to enter its critical section and before that request is granted. When the process crashes, no other process can enter critical section.

Problem 3.
Doorway section is the atomic inclement operation. In the doorway section, every threads will get unique number ($\because$atomic operation). Waiting section is while loop. threads are waiting for its turn. It is starvation-free. Two points make it work. First, each threads gets unique, continuous number thanks to the atomic operation. Second, variable "nowServing" is guaranteed to increase 1 because it is in critical section.

Problem 4.
Figure 2:
(a) No. Both $p0$ and $p1$ can check the flags simultaneously and break the while loop.
(b) Yes. It is because the algorithm doesn't provide mutual exclusion. The mutual exclusion is requirement of deadlock.
(c) No. One can always enter the critical section during the other is scheduled out.

Figure 3:
(a) Yes. Both $p0$ and $p1$ can't be critical section at the same time because they are looking a single shared variable.
(b) Yes. Both $p0$ and $p1$ is sharing a single flag $turn$ which must be either 0 or 1. At least one can enter the critical section.
(c) Yes. They will eventually run in turn like (b).

Problem 5.
(a) It satisfies the safety property. $p0$ can enter the critical section with initial value of a and b. Let's assume $p0$ successfully enters the critical section. When $p0$ enters the critical section, $a = 1$ and $b = 0$. That is, $p1$ can be at #7, #10, or #11 ($\because b = 1$ at #8,#9). In all cases of these, $p1$ can't enter the critical section as long as $a = 1$ (due to the while loop).
(b) It doesn't satisfies the liveness property. A schedule like $1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 7 \Rightarrow 8 \Rightarrow 9 \Rightarrow 4 \Rightarrow 10 \Rightarrow 5 \Rightarrow 11$ is valid and can keep looping infinitely. So neither $p0$ or $p1$ can enter the critical section.

Problem 6.
(a) Mutual-exclusion: Yes. A thread can enter critical section only if x is itself. And x can only be one of thread IDs. So only one thread can enter critical section. It is deadlock-free as well. Cases of deadlock are:
• both await $x = 0$
• both await $y = 0$
• one awaits $y = 0$ and another $x = 0$
Those never happens.

(b) Starvation-Freedom with 2 threads: No. Let's assume that thread 1 is in the critical section and thread 2 is in line #9 waiting $x = 0$. At this point, thread 2 is somehow scheduled out. While thread 2 is scheduled out, thread 1 exits the critical section and reset $x = 1$ in line #1. Thread 2 is scheduled in and check x is still not 0.

(c) Deadlock freedom with 3 threads: Yes. Same reason with (a)

(d) Mutual-exclusion with 3 threads: No. If thread 1 is in critical section and thread 2 is in line #9. New thread 3 starts and set $x = 3$ and skip line #2 because thread 2 has already set $y = 0$ at line #8. Now thread 3 can enter critical section even though thread 1 is still in the critical section.

Part II

Problem 1.
I ran experiments on a local machine (Table 1):
AMD FX(tm)-8350 Eight-Core Processor (8 cores with hyper threading)

Table 1: Filter vs Bakery on local machine

Iteration	2 threads	4 threads	8 threads
Filter	23ms	88ms	317ms
Bakery	30ms	54ms	136ms

Filter algorithm: To enter critical action, threads compete at each level. Even though a thread wins at a level, it competes against other threads at the next level.
Bakery algorithm: To enter critical action, threads compete once. Once a thread wins, it can enter the critical section.

Problem 2.
(1) LockOne
• Mutual-exclusion: Yes. I can see deadlock.
• Fairness: I can't say its fairness due to deadlock.
• Deadlock-freedom: No. It always goes to deadlock. So we can say it satisfies mutual-exclusion, not deadlock-free.
• Starvation-freedom: No. I can't say it is starvation-free because it is not deadlock-free.

(2) LockTwo
The last thread in LockTwo is a victim that we can only see the final value of the counter is 1999 rather than 2000.

Iteration 1:
Thread 0 value 1992
Thread 1 value 1993
Thread 0 value 1994
Thread 1 value 1995
Thread 0 value 1996
Thread 1 value 1997
Thread 0 value 1998
Thread 0 DONE.. Counter:1999

Iteration 2:
Thread 0 value 1992
Thread 1 value 1993
Thread 0 value 1994
Thread 1 value 1995
Thread 0 value 1996
Thread 1 value 1997
Thread 0 value 1998
Thread 0 DONE.. Counter:1999

Iteration 3:
Thread 0 value 1992

Thread 1 value 1993
Thread 0 value 1994
Thread 1 value 1995
Thread 0 value 1996
Thread 1 value 1997
Thread 0 value 1998
Thread 0 DONE.. Counter:1999

Iteration 4:
Thread 0 value 1992
Thread 1 value 1993
Thread 0 value 1994
Thread 1 value 1995
Thread 0 value 1996
Thread 1 value 1997
Thread 0 value 1998
Thread 0 DONE.. Counter:1999

Iteration 5:
Thread 0 value 1992
Thread 1 value 1993
Thread 0 value 1994
Thread 1 value 1995
Thread 0 value 1996
Thread 1 value 1997
Thread 0 value 1998
Thread 0 DONE.. Counter:1999

• Mutual-exclusion: Yes. Thanks to mutual-exclusion, we can see correct counter value of 1999.
• Fairness: Yes. The fact that each threads enters the critical section fairly is observed.
• Deadlock-freedom: No. We can always observe the last counter is 1999, not 2000 meaning that last thread is blocked and waiting for someone to unlock. This causes deadlock.
• Starvation-freedom: NO. The algorithm without deadlock-freedom doesn't satisfy starvation-freedom.

(3) Peterson
Iteration 1:
Thread 0 value 1994
Thread 1 value 1995
Thread 0 value 1996
Thread 1 value 1997
Thread 0 value 1998
Thread 0 value 1999
Thread 1 DONE.. Counter:1999

Thread 0 DONE.. Counter:2000

Iteration 2:
Thread 0 value 1940
Thread 1 value 1941
Thread 0 value 1942
Thread 1 value 1943
Thread 1 value 1944
Thread 1 value 1945
Thread 1 value 1946
Thread 1 value 1947
Thread 0 DONE.. Counter:1945
...
Thread 1 value 1997
Thread 1 value 1998
Thread 1 value 1999
Thread 1 DONE.. Counter:2000

Iteration 3:
Thread 0 value 1994
Thread 1 value 1995
Thread 0 value 1996
Thread 1 value 1997
Thread 0 value 1998
Thread 0 value 1999
Thread 1 DONE.. Counter:1999
Thread 0 DONE.. Counter:2000

Iteration 4:
Thread 0 value 1996
Thread 1 value 1997
Thread 0 value 1998
Thread 1 value 1999
Thread 0 DONE.. Counter:2000
Thread 1 DONE.. Counter:2000

Iteration 5:
Thread 0 value 1996
Thread 1 value 1997
Thread 0 value 1998
Thread 0 value 1999
Thread 1 DONE.. Counter:1999
Thread 0 DONE.. Counter:2000

• Mutual-exclusion: Yes. As we can see, the last thread is done with counter 2000.

- Fairness: Yes. I ran extra execution with small counter.

Iteration 6:
Thread 1 waiting
Thread 1 waiting
Thread 1 waiting
Thread 0 entering
Thread 1 waiting
Thread 0 waiting
Thread 1 entering
Thread 0 waiting
Thread 1 waiting
Thread 0 entering
Thread 1 waiting
Thread 0 waiting
Thread 1 entering
Thread 0 waiting
Thread 1 waiting
Thread 0 entering
Thread 1 waiting
Thread 0 waiting
Thread 1 entering
Thread 0 waiting
Thread 1 waiting
Thread 0 entering
Thread 1 waiting
Thread 0 waiting
Thread 1 entering
Thread 0 waiting
Thread 1 waiting
Thread 0 entering
Thread 1 waiting
Thread 0 waiting
Thread 1 entering
Thread 0 waiting
Thread 1 waiting
Thread 0 entering
Thread 1 waiting
Thread 0 waiting
Thread 1 entering
Thread 0 waiting
Thread 1 waiting
Thread 1 waiting
Thread 1 waiting
Thread 1 waiting

```
Thread 1 waiting
Thread 1 waiting
Thread 1 waiting
Thread 1 waiting
Thread 0 entering
Thread 1 waiting
Thread 0 waiting
Thread 1 entering
Thread 0 waiting
Thread 1 waiting
Thread 0 entering
Thread 1 waiting
Thread 0 waiting
Thread 1 entering
Thread 0 waiting
Thread 1 waiting
Thread 0 entering
Thread 1 waiting
Thread 0 waiting
Thread 1 entering
Thread 0 waiting
Thread 1 waiting
Thread 1 waiting
Thread 1 waiting
Thread 1 waiting
Thread 1 waiting
Thread 1 waiting
Thread 1 waiting
Thread 1 waiting
Thread 1 waiting
Thread 1 waiting
Thread 1 waiting
Thread 1 waiting
Thread 1 waiting
Thread 1 waiting
Thread 0 entering
Thread 1 waiting
Thread 1 entering
Thread 0 DONE.. Counter:19
Thread 1 DONE.. Counter:20
```

I can say it is **fair**.

• Deadlock-freedom: Yes. With large number of counter (e.q., 2000000), no deadlock is observed.
• Starvation-freedom: Yes. Based on its fairness and deadlock-freedom, we can say is is starvation-free.

(4) Filter
Iteration 1:
Thread 2 value 3994
Thread 3 value 3995
Thread 1 value 3996
Thread 0 value 3997
Thread 2 value 3998
Thread 3 value 3999
Thread 1 DONE.. Counter:3999
Thread 3 DONE.. Counter:4000
Thread 2 DONE.. Counter:3999
Thread 0 DONE.. Counter:3999

Iteration 2:
Thread 2 value 3994
Thread 3 value 3995
Thread 0 value 3996
Thread 1 value 3997
Thread 2 value 3998
Thread 3 value 3999
Thread 2 DONE.. Counter:3999
Thread 1 DONE.. Counter:3999
Thread 3 DONE.. Counter:4000
Thread 0 DONE.. Counter:3999

Iteration 3:
Thread 2 value 3994
Thread 3 value 3995
Thread 0 value 3996
Thread 1 value 3997
Thread 2 value 3998
Thread 3 value 3999
Thread 0 DONE.. Counter:4000
Thread 2 DONE.. Counter:4000
Thread 3 DONE.. Counter:4000
Thread 1 DONE.. Counter:4000

Iteration 4:
Thread 2 value 3994
Thread 3 value 3995
Thread 0 value 3996
Thread 1 value 3997
Thread 2 value 3998
Thread 3 value 3999

Thread 0 DONE.. Counter:3999
Thread 3 DONE.. Counter:4000
Thread 2 DONE.. Counter:3999
Thread 1 DONE.. Counter:3999

Iteration 5:
Thread 2 value 3994
Thread 3 value 3995
Thread 1 value 3996
Thread 0 value 3997
Thread 2 value 3998
Thread 3 value 3999
Thread 1 DONE.. Counter:3999
Thread 3 DONE.. Counter:4000
Thread 2 DONE.. Counter:4000
Thread 0 DONE.. Counter:3999

• Mutual-exclusion: Yes. We can observe correct value of counter.
• Fairness: No. Threads can be overtake others. (Maybe we can say "weak fairness")
• Deadlock-freedom: Yes. I haven't seen deadlock.
• Starvation-freedom: Yes. It is hard to show whether it is starvation-free or not from this experiments because one can always argue the experiment is not comprehensive enough. However I would say this algorithm is starvation-free thanks to its fairness and deadlock-freedom.

(5) Bakery
Iteration 1:
Thread 0 value 1992
Thread 1 value 1993
Thread 0 value 1994
Thread 1 value 1995
Thread 0 value 1996
Thread 1 value 1997
Thread 0 value 1998
Thread 0 value 1999
Thread 1 DONE.. Counter:1999
Thread 0 DONE.. Counter:2000

Iteration 2:
Thread 0 value 1992
Thread 1 value 1993
Thread 0 value 1994
Thread 1 value 1995
Thread 0 value 1996
Thread 1 value 1997

Thread 0 value 1998
Thread 1 value 1999
Thread 1 DONE.. Counter:2000
Thread 0 DONE.. Counter:2000

Iteration 3:
Thread 0 value 1992
Thread 1 value 1993
Thread 0 value 1994
Thread 1 value 1995
Thread 0 value 1996
Thread 1 value 1997
Thread 0 value 1998
Thread 1 value 1999
Thread 1 DONE.. Counter:2000
Thread 0 DONE.. Counter:2000

Iteration 4:
Thread 0 value 1992
Thread 1 value 1993
Thread 0 value 1994
Thread 1 value 1995
Thread 0 value 1996
Thread 1 value 1997
Thread 0 value 1998
Thread 1 value 1999
Thread 1 DONE.. Counter:2000
Thread 0 DONE.. Counter:2000

Iteration 5:
Thread 0 value 1992
Thread 1 value 1993
Thread 0 value 1994
Thread 1 value 1995
Thread 0 value 1996
Thread 1 value 1997
Thread 0 value 1998
Thread 1 value 1999
Thread 1 DONE.. Counter:2000
Thread 0 DONE.. Counter:2000

(a) Mutual-exclusion: Yes. The results show correct values.
(b) Fairness: Yes. I ran extra execution with 2 threads, small counter.

Thread 1 waiting

Thread 1 waiting
Thread 1 waiting
Thread 0 waiting
Thread 1 entering
Thread 0 waiting
Thread 1 waiting
Thread 0 entering
Thread 1 waiting
Thread 0 waiting
Thread 1 entering
Thread 0 waiting
Thread 1 waiting
Thread 0 entering
Thread 1 waiting
Thread 0 waiting
Thread 1 entering
Thread 0 waiting
Thread 1 waiting
Thread 0 entering
Thread 1 waiting
Thread 0 waiting
Thread 1 entering
Thread 0 waiting
Thread 1 waiting
Thread 0 entering
Thread 1 waiting
Thread 0 waiting
Thread 1 entering
Thread 0 waiting
Thread 1 waiting
Thread 0 entering
Thread 1 waiting
Thread 0 waiting
Thread 1 entering
Thread 0 waiting
Thread 1 waiting
Thread 0 entering
Thread 1 waiting
Thread 0 waiting
Thread 1 entering
Thread 0 waiting
Thread 1 waiting
Thread 0 entering
Thread 1 waiting
Thread 0 waiting

Thread 1 entering
Thread 0 waiting
Thread 1 waiting
Thread 0 entering
Thread 1 waiting
Thread 0 waiting
Thread 1 entering
Thread 0 waiting
Thread 1 waiting
Thread 0 entering
Thread 1 waiting
Thread 0 waiting
Thread 1 entering
Thread 0 waiting
Thread 0 entering
Thread 1 DONE.. Counter:19
Thread 0 DONE.. Counter:20

I can say it is fair.

(c) Deadlock-freedom: Yes. I haven't seen deadlock.
(d) Starvation-freedom: Yes. Like Peterson and Filter, it is starvation-free thanks to its fairness and deadlock-freedom.

Problem 3.
(a) It satisfies all of four properties.
• Mutual-exclusion: Yes. I ran experiment with counter value of 1000000.
Thread 9 DONE.. Counter:15458695
Thread 10 DONE.. Counter:15458695
Thread 8 DONE.. Counter:15458696
Thread 11 DONE.. Counter:15458695
Thread 2 DONE.. Counter:15864752
Thread 3 DONE.. Counter:15864936
Thread 0 DONE.. Counter:15865317
Thread 1 DONE.. Counter:15865324
Thread 4 DONE.. Counter:15869102
Thread 5 DONE.. Counter:15869167
Thread 7 DONE.. Counter:15869199
Thread 6 DONE.. Counter:15869193
Thread 13 DONE.. Counter:16000000
Thread 15 DONE.. Counter:16000000
Thread 12 DONE.. Counter:16000000
Thread 14 DONE.. Counter:16000000
The counter value of 16000000 proves the correctness.

• Deadlock-freedom: Yes. I ran several iterations of experiments, I haven't see any deadlock.
• Fairness: Yes. I havent seen a thread which requests first was overtaken by other which requests later.
• Starvation-freedom: Yes. With the fairness and deadlock-freedom of this algorithm, I can say it is starvation-free.

(*b*) There is no upper bound. It is because a thread can take a nap anywhere, and the other side of the tree will have winners to get the lock. Therefore, there is no upper bound.

(*c*)
• Local machine
AMD FX(tm)-8350 Eight-Core Processor (8 cores with hyper threading)

Table 2: Local machine

Iteration	2 threads	4 threads	8 threads	16 threads	32 threads	64 threads
1	14ms	66ms	138ms	833ms	4334ms	45554ms
2	17ms	65ms	133ms	527ms	3458ms	44673ms
3	18ms	69ms	131ms	330ms	6196ms	39191ms
Average	16ms	67ms	134ms	563ms	4663ms	43139ms

• Rlogin machine (gum.rlogin)
Intel(R) Xeon(R) CPU E5-2470 v2 @ 2.40GHz (40 Cores)
Time in ms.

Table 3: 2 threads

Iteration	12AM	6AM	9AM	12PM	6PM
1	52	37	42	45	48
2	47	45	44	44	44
3	47	43	47	44	45
Average	49	42	44	44	46

Table 4: 4 threads

Iteration	12AM	6AM	9AM	12PM	6PM
1	90	90	93	91	86
2	95	95	96	95	94
3	93	93	97	97	102
Average	93	93	95	94	94

Table 5: 8 threads

Iteration	12AM	6AM	9AM	12PM	6PM
1	240	239	243	241	241
2	243	251	238	242	239
3	241	252	233	225	239
Average	241	247	238	236	240

Table 6: 16 threads

Iteration	12AM	6AM	9AM	12PM	6PM
1	646	628	635	644	644
2	646	632	633	646	641
3	650	644	643	639	639
Average	647	635	637	643	641

Table 7: 32 threads

Iteration	12AM	6AM	9AM	12PM	6PM
1	1368	1332	1367	1372	1364
2	1389	1280	1391	1335	1351
3	1360	1362	1360	1351	1363
Average	1372	1325	1373	1353	1359

Table 8: 64 threads

Iteration	12AM	6AM	9AM	12PM	6PM
1	4867	4273	5406	5055	5052
2	4465	4421	5028	4942	4311
3	4725	4265	5038	4621	5070
Average	4686	4320	5157	4873	4811

- Local machine vs rlogin (Table 9, Figure 1)

Table 9: Local machine vs rlogin

Iteration	2 threads	4 threads	8 threads	16 threads	32 threads	64 threads
local	16	67	134	536	4663	43139
rlogin	45	95	238	643	1356	4737

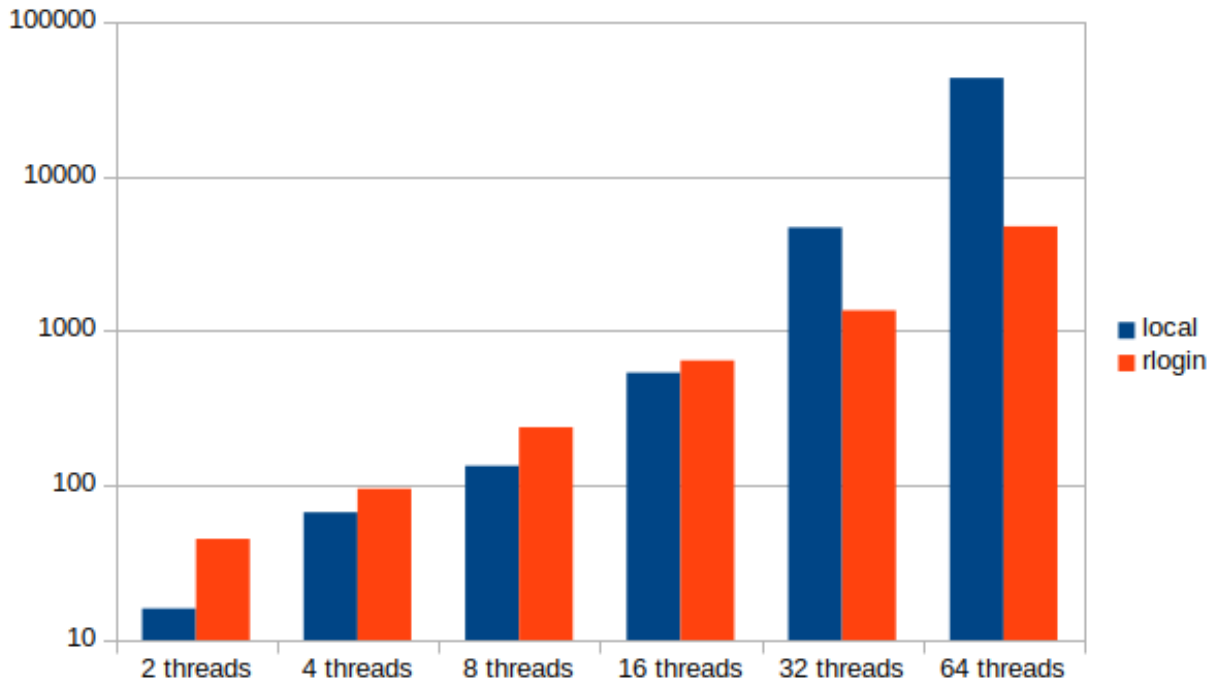

Figure 1: local machine vs rlogin (ms in log scale)