Writing Assignment 1:

Peichuan Wang 1147304

Jinyue Song 1234581

# Chapter 1:

## EX1:

## EX4:

The switch is initially off:

* One prisoner called C is designed to be the counter.

He will turn the switch on when entering the room. Every time when he turns the switch from “off” to “on”, counter++. If the switch is already on, he does nothing and leave, which means counter does not increase.

* Other prisoners will turn the switch off when entering the room. If the switch is on, he will turn it off. If the switch is off already, he will do nothing and leave.
* Prisoners can enter the room for N times, it means they can enter as many times as they want. At the same time, the number of prisoners is public, which is P.
* When the prisoner C finds (counter == P), he can declare all of them have visited the room.

The switch initial state is not known.

* + Still the prisoner C is designed to be the counter.

He will turn the switch on from off. If the switch is on, he does nothing and leave.

* + Others will turn the switch off from on. If the switch is off, they do nothing and leave. Everyone except the counter must turn the switch off twice at least.
  + The counter C will count the times of switching on and if (counter == 2\*P – 1), he can declare everyone has entered for at least once.

## EX6:

Amdahl’s Law:

speedup S = time of one processor to complete the job / time of n concurrent processors to complete the job

p is the fraction of the job that can be executed in parallel

1 - p: sequential part

overall time by parallelized computation: 1 – p + p/n

S = 1 / (1 – p + p / n)

a)

S = = = 2.5

if time of one processor execution is 1 unit, then the method M takes 0.4, which is 1 \* 40% = 0.4;

since “n” could be infinite large, could be 0. Then the result is 2.5

b)

Assume = the speedup on n processors and total computation time is unit 1. Since M accounts for 30% of the computation time, p is equal to 70%, which is 1 \* 70% = 0.7. Then, we have

We want to speed up M, so assume we have rate r, which is about to determine how much M is improved. Thus 0.3 \* r.

Since overall execution time is improved by a factor of 2, we have a = 2 \* .

Then we could have r = . When the number of processors is infinite, r = .

Thus, M has to use half of the original time in order to improve the execution time by 2.

c)

Assume and .

Speedup S = ; double the speedup => =>

Let’s simplify

Method M can be sped up three-fold => .

Based on the Amdahl’s Law (total execution time = sequential time + parallel time), we have

Then, we have .

Thus, the fraction is 0.75 of overall execution time.

# Chapter 2:

## EX:12

There are level 0, 1 and critical section. There are thread A, B, C.

In execution, thread A, B, C enter the level 0 and the victim is thread A.

Then thread B and C enter the level 1 and the victim is B.

So, thread C enters the critical section and leaves the critical section. At the same time, thread B finds thread C leaves, and thread B enters the critical section, which means thread A is still waiting.

However, at this moment, thread C comes back and wants to enter the critical section again. It will enter level 0, and then level 1 to wait for thread B leaving the critical section. After thread B leaves the critical section, thread C enters the critical section and thread B comes back to wait for entering the critical section again.

From the view of thread A, there is always a “level[k]” greater than its value i, which is 0. Thus, thread A could be overtaken many times.

## EX: 13

It satisfies mutual exclusion.

At the root, the lock only allows one thread to pass. So, only one thread could enter the critical section.

The tree lock is deadlock free.

First, the Peterson Lock is deadlock free. Then, locks in tree leaves and roots are actually Peterson Lock. So, those locks are deadlock free and this binary tree lock is deadlock free.

It is starvation free.

First, Peterson Lock is starvation free, which means at leaves and roots, there is no starvation at each local lock. Also, the order of requiring locks and releasing locks determines all threads can enter the critical section eventually.

There is an upper bound and it is 2 \* tree-length.

## EX: 14

1. class Filter implements Lock{
2. int[] level;
3. int[] victim;
4. public Filter(int n){
5. level = new int[n];
6. victim = new int[n-l]; //modify line 6; reduce the # of victim; there are no victims after level (n-l)
7. for (int i = 0; i < n; i++){
8. level[i] = 0;
9. }
10. }
11. public void lock(){
12. int me = ThreadID.get();
13. for (int i = 1; i < n-l; i++){ //
14. level[me] = i;
15. victim[i] = me;
16. while ((exist k != me) (|level[k] >= i| > (l-1) && victim[i] == me))
17. // “|level[k] >= i|” means the # of levels greater than i, or # of threads in the levels lower than level[me]; if this number is greater than (l – 1), current thread should wait.
18. }
19. }