Assignment 2

作业范围:

3-Critical sections, locks, and barriers.pdf

4-Semaphores.pdf

3.3 Suppose a computer has an atomic swap instruction, defined as follows:

```
Swap(var1, var2):
  ⟨ tmp = var1; var1 = var2; var2 = tmp; ⟩
```

In the above, `tmp` is an internal register.

(a) Using `Swap`, develop a solution to the critical section problem for `n` processes. Do not worry about the eventual entry property. Describe clearly how your solution works and why it is correct.

3.16 Consider the following implementation of a single `n`-process barrier:


```
int arrive[n] = {0, 0, ..., 0};

code executed by Worker[1]:
  arrive[1] = 1;
  ⟨await (arrive[n] == 1);⟩

code executed by Worker[i = 2 to n]:
  ⟨await (arrive[i-1] == 1);⟩
  arrive[i] = 1;
  ⟨await (arrive[n] == 1);⟩
```


(a) Explain how this barrier works.

(b) What is the time complexity of the barrier?

For 4.3, you are simulating P and V. The value of the semaphore (which should be maintained as an integer) can be negative, so long as P and V produce the semantically correct results.

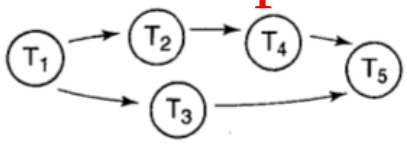
same time.

4.3 Recall that Fetch-and-Add, `FA(var, increment)`, is an atomic function that returns the old value of `var` and adds `increment` to it. Using `FA`, develop a simulation of the `P` and `V` operations on general semaphore `s`. Assume that memory reads and writes are atomic but that `FA` is the only more powerful atomic operation.

4.4 A precedence graph is a directed, acyclic graph. Nodes represent tasks, and arcs indicate the order in which tasks are to be accomplished. In particular, a task can execute as soon as all its predecessors have been completed. Assume that the tasks are processes and that each process has the following outline:


```
process T {
    wait for predecessors, if any;
    body of the task;
    signal successors, if any;
}
```


(a) Using semaphores, show how to synchronize five processes whose permissible execution order is specified by the following precedence graph:



For 4.31, do not worry about specifying a global invariant. However, in your solution, I am requiring you to use the passing the baton technique.

problem differently. (Hint: Use the technique of passing the baton.)

4.31 *The One-Lane Bridge.* Cars coming from the north and the south arrive at a one-lane bridge. Cars heading in the same direction can cross the bridge at the same time, but cars heading in opposite directions cannot.

(a) Develop a solution to this problem. First specify a global invariant, then develop a solution using semaphores for synchronization. Do not worry about fairness.

Last question :

- Consider the two-thread, busy waiting solution given in class to the critical section problem.


```
bool flag[2] := {false, false};
int turn := 0
entry(id) {
    flag[id] := true;
    turn := 1 − id;
    while (flag[1 − id] and turn == 1 − id) ;
}

exit(id) {
    flag[id] := false;
}
```

(Recall that each thread *repeatedly* does the following: first executes non-critical code, then calls **entry**(id), then executes its critical section, and then calls **exit**(id).)

What effect (if any) would making each of the following changes, *independently*, have on the (entire) critical section solution? Be specific in your answers, using at most a few sentences for justification.

A. Changing the *and* to an *or* in the **while** loop.

B. Changing $flag[id] := true$ to $flag[id] := false$ in **entry**.

C. Changing $flag[id] := false$ to $flag[id] := true$ in **exit**.