UFC

Universidade Federal do Ceará
Centro de Ciências/Departamento de Computação
CK0227 - Programação Concorrente
Prof. Dr. Francisco Heron De Carvalho Junior

DC

3ª Lista de Exercícios   Data de Entrega: 20/05/2019

Nome:
Matrícula:

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:

$$\begin{tikzcd}
 & T_2 \arrow[r] & T_4 \arrow[dr] & \\
T_1 \arrow[ur] \arrow[dr] & & & T_5 \\
 & T_3 \arrow[urr] & &
\end{tikzcd}$$

Minimize the number of semaphores that you use, and do not impose constraints not specified in the graph. For example, `T2` and `T3` can execute concurrently after `T1` completes.

(b) Describe how to synchronize processes, given an arbitrary precedence graph. In particular, devise a general method for assigning semaphores to edges or processes and for using them. Do not try to use the absolute minimum number of semaphores since determining that is an NP-hard problem for an arbitrary precedence graph!

4.8 Give all possible final values of variable **x** in the following program. Explain how you got your answer.

```
int x = 0; sem s1 = 1, s2 = 0;
co P(s2); P(s1); x = x*2; V(s1);
// P(s1); x = x*x; V(s1);
// P(s1); x = x+3; V(s2); V(s1);
oc
```

4.10 Consider the butterfly and dissemination barriers in Figures 3.15 and 3.16.

(a) Using semaphores for synchronization, give complete details for a butterfly barrier for eight processes. Show the code that each process would execute. The barrier should be reusable.

(b) Repeat (a) for a dissemination barrier for eight processes.

(c) Compare your answers to (a) and (b). How many variables are required for each kind of barrier? If each semaphore operation takes one unit of time, what is the total time required for barrier synchronization in each algorithm?

(d) Repeat (a), (b), and (c) for a seven-process barrier.

(e) Repeat (a), (b), and (c) for a 12-process barrier.

Chapter 3 Locks and Barriers

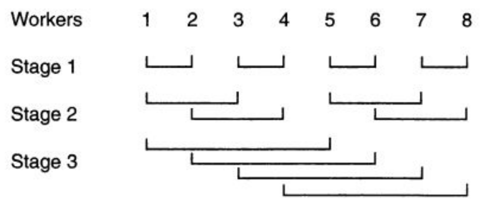

Figure 3.15 Butterfly barrier for eight processes.

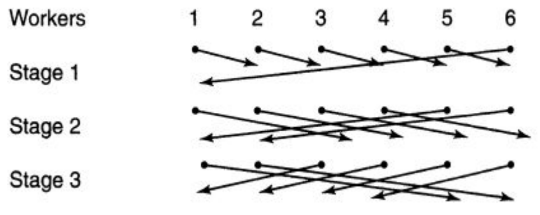

Figure 3.16 Dissemination barrier for six processes.

4.15 Another way to solve the bounded buffer problem is as follows. Let `count` be an integer between `0` and `n`. Then `deposit` and `fetch` can be programmed as follows:

```
deposit:
  ⟨ await (count < n)
        buf[rear] = data;
        rear = (rear+1) % n; count = count+1; ⟩
fetch:
  ⟨ await (count > 0)
        result = buf[front];
        front = (front+1) % n; count = count-1; ⟩
```

Implement these `await` statements using semaphores. (*Hint:* Use a variation of passing the baton.)

4.21 Consider the following solution to the readers/writers problem. It employs the same counters and semaphores as in Figure 4.13, but uses them differently.

```
int nr = 0, nw = 0;   # numbers of readers and writers
sem e = 1;            # mutual exclusion semaphore
sem r = 0, w = 0;     # delay semaphores
int dr = 0, dw = 0;   # delay counters

process Reader[i = 1 to M] {
  while (true) {
    P(e);
      if (nw == 0) { nr = nr+1; V(r); }
      else  dr = dr+1;
    V(e);
    P(r);   # wait for permission to read
    read the database;
    P(e);
      nr = nr-1;
      if (nr == 0 and dw > 0)
        { dw = dw-1; nw = nw+1; V(w); }
    V(e);
  }
}

process Writer[j = 1 to N] {
  while (true) {
    P(e);
      if (nr == 0 and nw == 0) { nw = nw+1; V(w); }
      else  dw = dw+1;
    V(e);
    P(w);
    write the database;
    P(e);
      nw = nw-1;
      if (dw > 0) { dw = dw-1; nw = nw+1; V(w); }
      else
        while (dr > 0) { dr = dr-1; nr = nr+1; V(r); }
    V(e);
  }
}
```

(a) Carefully explain how this solution works. What is the role of each semaphore? Show that the solution ensures that writers have exclusive access to the database and a writer excludes readers.

(b) What kind of preference does the above solution have? Readers preference? Writers preference? Alternating preference?

(c) Compare this solution to the one in Figure 4.13. How many `P` and `V` operations are executed by each process in each solution in the best case? In the worst case? Which program do you find easier to understand, and why?

4.27 *Cigarette Smokers Problem* [Patil 1971; Parnas 1975]. Suppose there are three smoker processes and one agent process. Each smoker continuously makes a cigarette and smokes it. Making a cigarette requires three ingredients: tobacco, paper, and a match. One smoker process has tobacco, the second paper, and the third matches. Each has an infinite supply of these ingredients. The agent places a random two ingredients on the table. The smoker who has the third ingredient picks up the other two, makes a cigarette, then smokes it. The agent waits for the smoker to finish. The cycle then repeats.

Develop a solution to this problem using semaphores for synchronization. You may also need to use other variables.

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.

(b) Modify your answer to (b) to ensure that any car that is waiting to cross the bridge eventually gets to do so. You may want to solve the problem differently. (*Hint:* Use the technique of passing the baton.)