

Homework 2

Assignment due:

Problems 1, 2, 3: Hand in at the start of class (9:30 am) on Tuesday, March 4th, or turnin electronically by the same time. Turnin typed (not hand-written) answers to the non-programming problems. If using electronic turnin, provide a pdf file named **answers.pdf**. Use the command: `turnin 422assign2 answers.pdf Problems 1-3` are worth 36 points (12 points ea.).

A reminder: Homework assignments are individual efforts. The two projects can be done in teams of two, not the homework. You may discuss the meanings of questions with classmates, but the answers and programs you turn in are to be yours alone. For the exercises from the book, explain your answers clearly and succinctly.

Problem 1: Exercise 3.2, page 142

Suppose a computer has atomic decrement **DEC** and increment **INC** instructions that also return the value of the sign bit of the result. In particular, the decrement instruction has the following effect:

```
DEC(var, sign):
  < var = var - 1;
    if (var >= 0)
      sign = 0;
    else
      sign = 1; >
```

INC is similar, the only difference being that it adds 1 to var.

Using **DEC** and/or **INC**, 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.

Problem 2: Exercise 4.13, page 194

Consider the following proposal for implementing **await** statements using semaphores:

```
sem e = 1, d = 0; # entry and delay semaphores
int nd = 0;      # delay counter
# implementation of <await (B) S;>
P(e);
while (!B) {
  nd = nd + 1;
  V(e);
  P(d);
  P(e);
}
S;
while (nd > 0) {
  nd = nd - 1;
  V(d);
}
V(e);
```

Does this code ensure that the **await** statement is executed atomically? Does it avoid deadlock? Does it guarantee that **B** is true before **S** is executed? For each of these questions, either give a convincing argument why the answer is “yes”, or give an execution sequence that illustrates why the answer is “no”.

Problem 3: Exercise 4.21, pages 197-8

Consider the following solution to the readers/writers problem. It employs the same counters and semaphores as in Figure 4.13 (page 176), but uses them differently.

<pre>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</pre>	
<pre>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); # await read permission read the database P(e); nr = nr - 1; if (nr == 0 and dw > 0) { dw = dw - 1; nw = nw + 1; V(w); } V(e); } }</pre>	<pre>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); } }</pre>

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 solution have? Readers preference? Writers preference? Alternating preference?

c.) Compare this solution to the one in Figure 4.13 (p. 176). How many P and V operations are executed by each process in each solution in the best case? In the worst case?