Name: ______________________________

CSc 422, Spring 2002 — Examination 1

You may use up to *two* pages of notes for this exam, but otherwise it is closed book. Please put your name on the top of your notes and turn them in with your examination. Do your work on these sheets, using additional sheets if necessary.

The exam is worth 60 points, divided as indicated. *You must show your work and/or explain your answers.* This is required for full credit and is helpful for partial credit.

1. [6 points] In the palindrome program in Homework 2, the bag contained 26 tasks. These were done in the order 'a', 'b', ..., 'z'. The tasks have different sizes because, for example, there are more words that begin with 'a' than begin with 'z'.

Suppose that the bag is represented as an ordered list. The first task in the bag (front of the list) is the one with the most dictionary words, the second task is the one with the next largest number of dictionary words, and so on with the shortest task last.

(a) Give brief pseudo-code to take a task from the bag.

(b) Would your parallel program most likely be faster, slower, or about the same? Why?

2. [6 points] Consider a parallel program with 12 worker processes, numbered from 1 to 12 as in `process Worker[i = 1 to 12]`.

(a) How many stages would be required for a dissemination barrier for the 12 workers?

(b) In the dissemination barrier, worker 5 would interact with which other workers?

3. [6 points] The critical section protocol below uses the Test-and-Set instruction. It is slow when there is contention.

```
while (TS(lock)) { skip; }     # CSenter
critical section
lock = true;                   # CSexit
```

(a) What is contention?

(b) Why is the above protocol slow when there is contention?

4. [12 points] Consider the following program:

```
int x = 0, y = 0, z = 0;
sem lock1 = 1, lock2 = 1;
```

```
process foo {                    process bar {
  z = z + 2;                       P(lock2);
  P(lock1);                        y = y + 1;
  x = x + 2;                       P(lock1);
  P(lock2);                        x = x + 1;
  V(lock1);                        V(lock1);
  y = y + 2;                       V(lock2);
  V(lock2);                        z = z + 1;
}                                }
```

(a) This program might deadlock. How?

(b) What are the possible final values of x, y, and z in the deadlock state?

(c) What are the possible final values of x, y, and z if the program terminates?

5. [15 points] It is possible to solve the critical section problem by using a separate coordinator process. A user process signals the coordinator when it wants to enter its critical section and then waits for permission to enter. A user process signals the coordinator again when it has exited its critical section. The coordinator checks to see which users want to enter and grants permission to one of them at a time.

Assume there are `n` user processes, numbered from 1 to `n`. Develop code that the user processes execute for CSenter and CSexit. Also develop code that the coordinator process executes.

Use flags and `await` statements (or `while` loops) for synchronization. Your solution must continue to work if a user process terminates outside its critical section. (You might find it helpful to draw a diagram of the process interaction.)

shared variables:

User code for CSenter and CSexit:

Coordinator process:

6. [15 points] Several processes share a resource that has U units. Processes request one unit at a time. They release one or more units at a time. Request and release are atomic operations as shown below.

Develop implementations of `request` and `release`. Use semaphores for synchronization. Be sure to declare and initialize additional variables that you need.

shared variables:

```
  int free = U      # initially all units are free
```

```
request():  # ⟨ await(free > 0) free = free - 1; ⟩
```

```
release(int number):  # ⟨ free = free + number; ⟩
```