

Homework 4

Assignment due:

All problems: 8:00 pm, Wednesday, May 7th.

Problems 1, 2: Turnin typed (not hand-written) answers to the non-programming problems. You can hand the assignment to me in class, during office hours, or slip it under my office door, prior to the turnin time. If using electronic turnin, provide a pdf file named **answers.pdf**, or a text file named **answers.txt**. See the instructions at the end regarding the use of turnin. Problems 1 and 2 are worth 36 points (18 points each).

Program: Submit the program electronically by 8:00 pm on Wednesday, May 7th. See the end of this assignment for information on electronic turnin. The programming exercise is worth 64 points.

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 8.6, page 417 (partial)

Consider a self-scheduling disk driver process as in Figure 7.9 (page 311). Suppose the process exports one operation: **request(cylinder, ...)**. Show (in pseudo-code) how to use rendezvous and an **in** statement to implement the shortest seek time scheduling algorithm. (Hint: Use scheduling expressions.) Note: the question in the book asks you to do circular scan and elevator as well; for this assignment you only need to implement the shortest seek time).

Problem 2:

Develop (in pseudo-code) a solution to the Dining Philosophers problem using RPC. Each philosopher, when ready to eat, will call:

```
chop1 = getChop(my_id);
chop2 = getChop(my_id);
... eat for a little while ...
call releaseChop(my_id, chop1);
call releaseChop(my_id, chops);
```

The server should only give philosophers the two chopsticks that are nearest to the philosopher (as in the standard description of the Dining Philosophers problem). In addition, the server should insure that deadlock will not happen.

Programming Exercise:

Due: Wednesday, May 7th, 8:00 pm.

Time Server and Rendezvous:

Write an MPD program that implements a time server, similar to the one shown in Figure 8.7, page 378, that uses rendezvous and the **in** statement. Use a *tick worker* to simulate the clock ticks; have this tick worker use the MPD **nap()** function to wake up every 100 msec (everytenth

of a second) and notify the server's `tick()` operation.

The one change from the description in Figure 8.7 is that the clients specify a delay time rather than a wakeup time. That is, the clients specify the number of seconds they want to sleep, rather than specifying the time to be awakened. This implies that the Client will need to invoke the appropriate `in` operation for delay and then do a receive operation. Each client makes `numRounds` calls to the server. Each call uses a random coin flip to choose the `get_time` or `delay` operations. Use a random number for the requested delay time, with a range of one to three seconds. In between calls to the server, the client should nap for up to two seconds to simulate doing other work before making another request to the server.

Use MPD's virtual machines to create workers processes on both `lectura` and `voltron`. Create half the workers on `lectura` and half on `voltron`. The tick worker and the server can be created on the machine where the command is entered.

Turnin a `readme` file that tells us what to type to compile your program, and several examples of simulations to run. You should have two command-line arguments: number of clients, number of rounds. You may have more command-line arguments, if needed.

Turnin:

Use the `turnin` program to turn in the code file(s) for your solution. Turnin a `README.txt` file that:

- states how to compile your program,
- gives a sample command-line to execute your program, and
- contains the output of your program for the sample command-line.

The assignment name for use in the `turnin` command is `422assign4`.