

Homework 3

Assignment due:

Problems 1, 2, 3: Hand in at the start of class (9:30 am) on Tuesday, April 15th, or turn in electronically by the same time. Turn in typed (not hand-written) answers to the non-programming problems. If using electronic turnin, provide a pdf file named **answers.pdf** or **answers.txt**. Use **422assign3** as the name of the assignment for electronic turnin.

Problems 1-3 are worth 36 points (12 points each).

Program: Submit the program electronically by 8:00 pm on Monday, April 21st. 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 5.8, page 256-257

The Savings Account Problem. A savings account is shared by several people (processes). Each person may deposit or withdraw funds from the account. The current balance in the account is the sum of all deposits to date minus the sum of all withdrawals to date. The balance must never become negative. A deposit never has to delay (except for mutual exclusion), but a withdrawal has to wait until there are sufficient funds.

- I asked this as a question on Test #3 last spring (see Sample Test #3). You are to do parts b.) and c.)
- Write a monitor that services withdrawals FCFS. For example, suppose the current balance is \$200, and one customer is waiting to withdraw \$300. If another customer arrives, he must wait, even if he wants to withdraw at most \$200. Assume there is a magic function **amount (cv)** that returns the value of the amount parameter of the first process delayed on **cv**. Use the Signal and Continue discipline.
- Suppose the magic **amount** function does not exist. Modify your answer to b.) to simulate it in your solution.

Problem 2: Exercise 5.10, page 257

Atomic Broadcast. Assume one producer process and **n** consumer processes share a bounded buffer having **b** slots. The producer deposits messages in the buffer; consumers fetch them. Every message deposited by the producer is to be received by all **n** consumers. Furthermore, each consumer is to receive the messages in the order they were deposited. However, consumers can receive messages at different times. For example, one consumer could receive up to **b** more messages than another if the second consumer is slow.

Develop a monitor that implements this kind of communication. Use the Signal and Continue discipline.

Problem 3: Exercise 7.7, page 354

Develop an implementation of a time-server process. The server provides two operations that can be called by client processes: one to get the time of day and one to delay for a specified interval. In addition, the time server receives periodic “tick” messages from a clock interrupt handler. Also show the client interface to the time server for the time of day and delay operations.

Programming Exercise:**Due:** Monday, April 21st, 8:00 pm.

Write two programs to the time-server problem (see Problem 3 above). One solution will be a monitor solution using threads on a shared memory machine. The second solution will be a distributed memory solution using message passing.

- a.) Write a monitor solution for a shared memory machine such as *lectura* or *voltron*. Create one thread to be the timer thread that sends “ticks” to the time-server. Create additional threads to be workers that want to sleep for varying amounts of time. Each worker thread should execute a loop that has it request a sleep period, then simulate a work session. Repeat this a number of times specified by a command-line parameter. The “simulate a work session” can be done by sleeping, writing a loop that performs some calculations, etc.
- b.) Write a message-passing solution for distributed memory machines. Create one process to be the time-server. Create a second process that sends “ticks” to the time-server. Create additional processes that want to sleep for varying amounts of time. Use sockets for the message-passing.

Use the same language for both programs. You may use Java or C and pthreads. (If you are not familiar with sockets in C, we will cover the use of sockets in C on Thursday, the 17th.)

Provide a README file that states how to compile your solutions to a.) and b.). Also, state how to run both solutions. Provide some examples of commands to run your programs.

Turnin:

Use the **turnin** program to turn in the code file(s) for your solution.

The assignment name for use in the turnin command is **422assign3**.

See the man page for the **turnin** program for details on what **turnin** can do and how you can confirm that your files were turned in.