Lab 12: Classic Problems in Concurrent Programming

Goals:

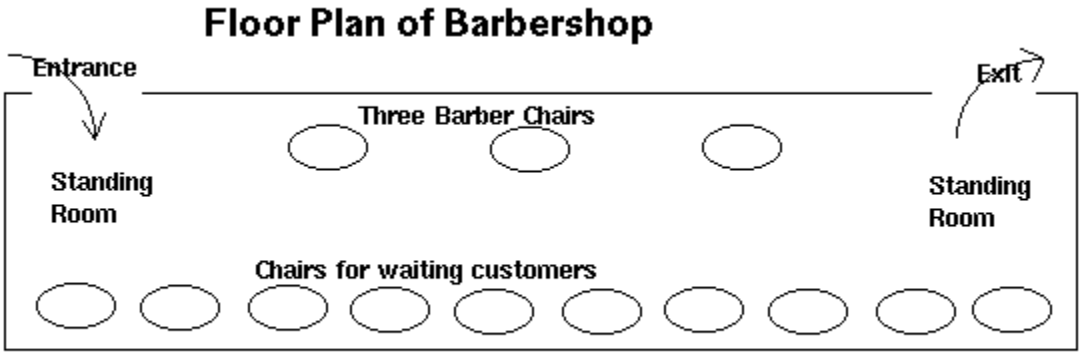
- Provide practice with process coordination using semaphores (and possibly other IPC mechanisms).
- Learn about classic problems in concurrent programming.

Collaboration: You will complete this lab in teams of **2-3** of your choice. You may consult me or your classmates with proper attribution.

Overview: In this lab, you are to write a C program to solve **ONE** of the following two problems.

Option 1: The Barbershop Problem

Many books, including our textbook, state variations of the Sleeping Barber Problem, which was first proposed by Edsger W. Dijkstra in 1968. (E. W. Dijkstra, "Co-operating Sequential Processes", in F. Genuys (ed.), *Programming Languages*, Academic Press, 1968, pp. 43-112.) This exercise considers the following version:



Three barbers work independently in a barber shop:

- The barbershop has 3 barber chairs, each of which is assigned to one barber.
- Each barber follows the same work plan:
    - The barber sleeps (or daydreams) when no customer customer is waiting (and is not in the

barber's own chair).

- When the barber is asleep, the barber waits to be awaken by a new customer. (A sign in the shop indicates which barber has been asleep longest, so the customer will know which barber to wake up if multiple barbers are asleep.)
- Once awake, the barber cuts the hair of a customer in the barber's chair.
- When the haircut is done, the customer pays the barber and then is free to leave.
- After receiving payment, the barber calls the next waiting customer (if any). If such a customer exists, that customer sits in the barber's chair and the barber starts the next haircut. If no customer is waiting, the barber goes back to sleep.

- Each customer follows the following sequence of events.
  - When the customer first enters the barbershop, the customer leaves immediately if more than 20 people are waiting (10 standing and 10 sitting). On the other hand, if the barbershop is not too full, the customer enters and waits.
  - If at least one barber is sleeping, the customer looks at a sign, wakes up the barber who has been sleeping the longest, and sits in that barber's chair (after the barber has stood up).
  - If all barbers are busy, the customer sits in a waiting-room chair, if one is available. Otherwise, the customer remains standing until a waiting-room chair becomes available.
  - Customers keep track of their order, so the person sitting the longest is always the next customer to get a haircut.
  - Similarly, standing customers remember their order, so the person standing the longest takes the next available waiting-room seat.

For this exercise, you are to write a C program to simulate activity for this barbershop:

a. Simulate each barber and each customer as a separate process.

b. Altogether, 30 customers should try to enter.

c. Use a random number generator so that a new customer arrives every 1, 2, 3, or 4 seconds. (This might be accomplished by a statement such as `sleep(1+(rand()%4));`.

d. Similarly, each haircut lasts between 3 and 6 seconds.

e. Each barber should report when he/she starts each haircut and when he/she finishes each haircut.

f. Each customer should report when he/she enters the barbershop. The customer also should report if he/she decides to leave immediately.

g. Similarly, if the customer must stand or sit in the waiting room, the customer should report when each activity begins.

h. Finally, the customer should report when the haircut begins and when the customer finally exits the shop.

i. Semaphores and shared memory should be used for synchronization.

Option 2: Baboons Crossing a Canyon

This exercise is a slightly modified version of exercises 2.35 and 2.36 in Tannenbaum and Woodhull, *Operating Systems: Design and Implementation, Second Edition.*

A student majoring in anthropology and minoring in computer science has embarked on a research project to see if African baboons can be taught about deadlocks. She locates a deep canyon and fastens a rope across it, so the baboons can cross hand-over-hand.

Passage along the rope follows these rules:

- Several baboons can cross at the same time, provided that they are all going in the same direction.
- If eastward moving and westward moving baboons ever get onto the rope at the same time, a deadlock will result (the baboons will get stuck in the middle) because it is impossible for one baboon to climb over another one while suspended over the canyon.
- If a baboon wants to cross the canyon, he must check to see that no other baboon is currently crossing in the opposite direction.
- Your solution should avoid starvation. When a baboon that wants to cross to the east arrives at the rope and finds baboons crossing to the west, the baboon waits until the rope in empty, but no more westward moving baboons are allowed to start until at least one baboon has crossed the other way.

For this exercise, you are to write a C program to simulate activity for this canyon crossing problem:

a. Simulate each baboon as a separate process.

b. Altogether, 30 baboons will cross the canyon, with a random number generator specifying whether they are eastward moving or westward moving (with equal probability).

c. Use a random number generator, so the time between baboon arrivals is between 1 and 8 seconds.

d. Each baboon takes 1 second to get on the rope. (That is, the minimum inter-baboon spacing is 1 second.)

e. All baboons travel at the same speed. Each traversal takes exactly 4 seconds, after the baboon is on the rope.

f. Use semaphores for synchronization. You may also use shared memory. (Additional communication via sockets is allowed, but do not use sockets unless such communication is clearly needed.)

What to Turn In

Write a program to solve **ONE** of the exercises given above.

Part A: Due Monday, 4 December, 2006:

- A complete description of your algorithm, in English or Pseudocode, including any shared data and synchronization.

Part B: Due Friday, 8 December, 2006 at 5 p.m.:

- An updated description of your algorithm.
- A complete program listing, with appropriate comments for each procedure and for any logical blocks of code.
- Two or three test runs.
- A brief written commentary or annotation describing the sequence of events in the test runs. (e.g., Are any processes blocked or sleeping at any point? Are activities occurring in the expected sequence?)
- Use the format for submitting assignments for organizing these pieces.

Janet Davis (davisjan@cs.grinnell.edu)

Created November 26, 2006
Last revised December 5, 2006
With thanks to Henry Walker