CSE4/521 Introduction to Operating System Fall 2010
Project 2: Exercises in Concurrency and Inter-process Communication Models

Objectives:

Learn to:

- Solve inter-process communication problems during concurrent execution of processes.
- Use Posix Pthread library for concurrency.

Problem Statement:
1a. Multi-processor Synchronization: Larry, Moe, and Curly are planting tulip bulbs. Larry digs the holes. Moe then places a bulb in each hole. Curly then fills the hole up. There are several synchronization constraints:

- Moe cannot plant a bulb unless at least one empty hole exists, but Moe does not care how far Larry gets ahead of Moe.
- Curly cannot fill a hole unless at least one hole exists in which Moe has planted a bulb, but the hole has not yet been filled. Curly does not care how far Moe gets ahead of Curly.
- Curly does care that Larry does not get more than MAX holes ahead of Curly. Thus, if there are MAX unfilled holes, Larry has to wait.
- There is only one shovel with which both Larry and Curly need to dig and fill the holes, respectively.

Design, implement and test a solution for this IPC problem, which represent Larry, Curly, and Moe. Use semaphores as the synchronization mechanism.
(LarryMoeCurly1.c →LCM1)

1b. In this problem, we add one more interaction between Moe and Curly. Moe will have to use a special "blub planter" to plant a bulb and Curly will have to use the same "bulb planter" to add the fertilizer. And after that Curly uses the shovel as above (1a) to fill the hole. Extend the problem above to include this scenario. Design and implement the solution for this problem. (LarryMoeCurly2.c →LCM2)

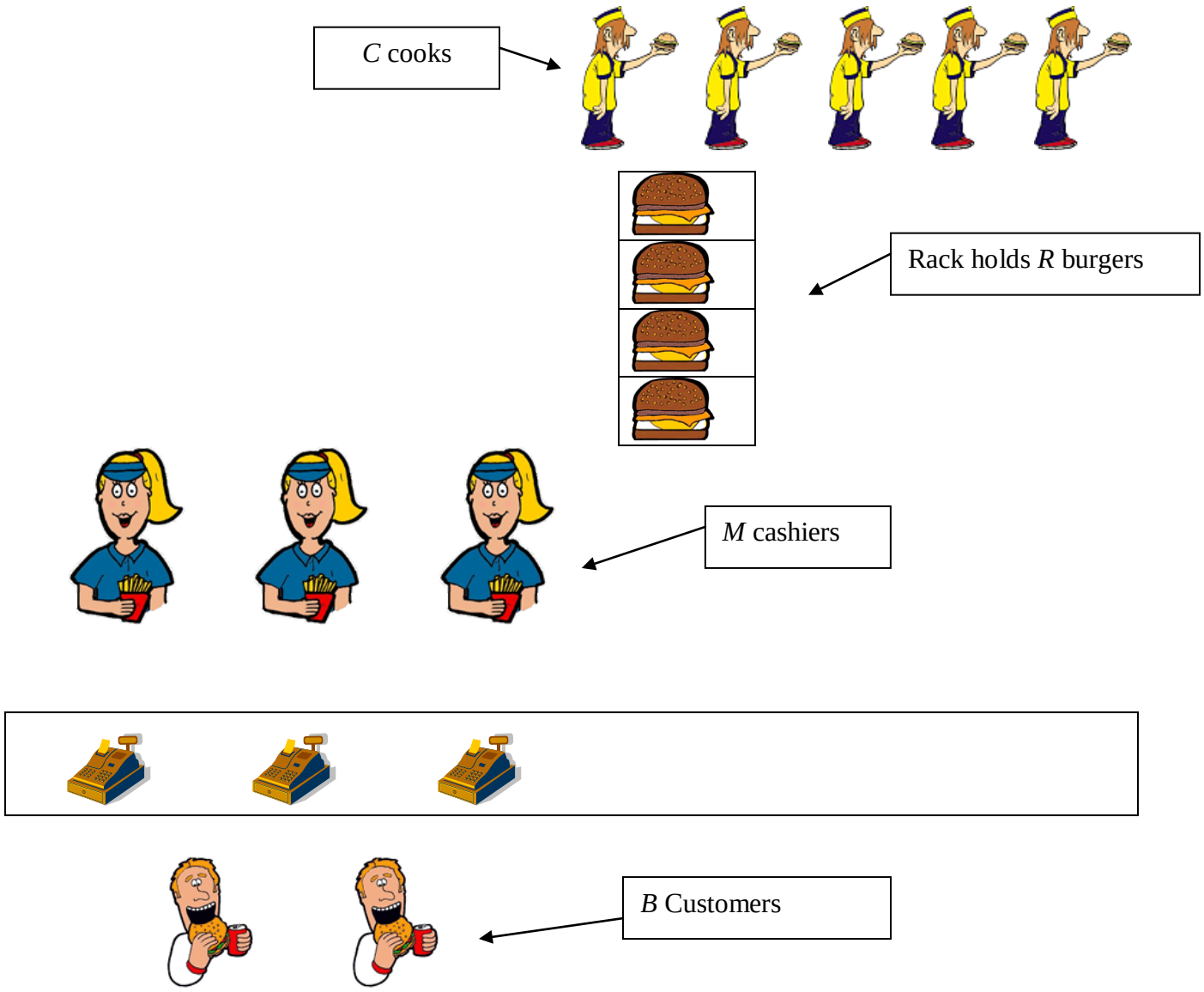
2. Exercise in concurrency by J. Trono: Santa-Elf-Reindeer: Implement a solution to Santa-elf-reindeer problem we discussed in class. A solution is provided in the paper by J. Trono. (SantaElfReindeer.c →SER)

3.[1] **IPC models Combination:** Design, implement and test a solution for the **IPC problem** specified below. Suppose we have the following scenario:
Operation of this scenario is as follows:

- Cooks, Cashiers, and Customers are each modeled as a thread
- Cashiers sleep until a customer is present
- A Customer approaching a cashier can start the order process
- A Customer cannot order until the cashier is ready

[1] This problem designed by C. Egert who was an UB Instructor; currently at RIT

- Once the order is placed, a cashier has to get a burger from the rack
- If a burger is not available, a cashier must wait until one is made
- The cook will always make burgers and place them on the rack
- The cook will wait if the rack is full
- There are NO synchronization constraints for a cashier presenting food to the customer.



a) Identify the classical IPC problem(s) needed to solve this problem.
b) List the names below and identify which items map to each classical IPC problem (use the variables provided in the diagram).
c) If this problem is to be modeled using semaphores, how many semaphores are needed? Identify how each semaphore is to be used and what the initial value should be set to. Use the variables in the diagram as necessary.

d) Implement a (concurrent multi-threaded) solution to solve the problem and test it thoroughly. Show output runs that illustrate the various possibilities of the set up. (BurgerBuddies.c → BBC)

Material to be submitted:

- Submit the source code for the programs. Use meaningful names for the file so that the contents of the file is obvious from the name. You may zip all the source files into a single file. Also provide a Pr2README file that explains the contents of the zip file. Pr2README file should have an observation section for each of the three problems. Use tables where ever suitable (5 points for documentation)
- Use internal documentation to explain your design.
- Test runs: It is very important that you show that your program works for all possible inputs. Submit a single script that shows for each program the working for correct input as well as graceful exit on error input.
- Include your makefile within your zip/tar file.

Due date

10/31/2009 Sunday; submit on-line before mid-night. 10/30, 10/31 we will not answer any basic debugging question. By then you should have completed all the work (design and coding) and be working on last minutes touches and testing.