CSE4/521 Introduction to Operating System Fall 2008
Project 2: Inter-process communication and synchronization

Objectives:

Learn to:

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

Problem Statement:

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

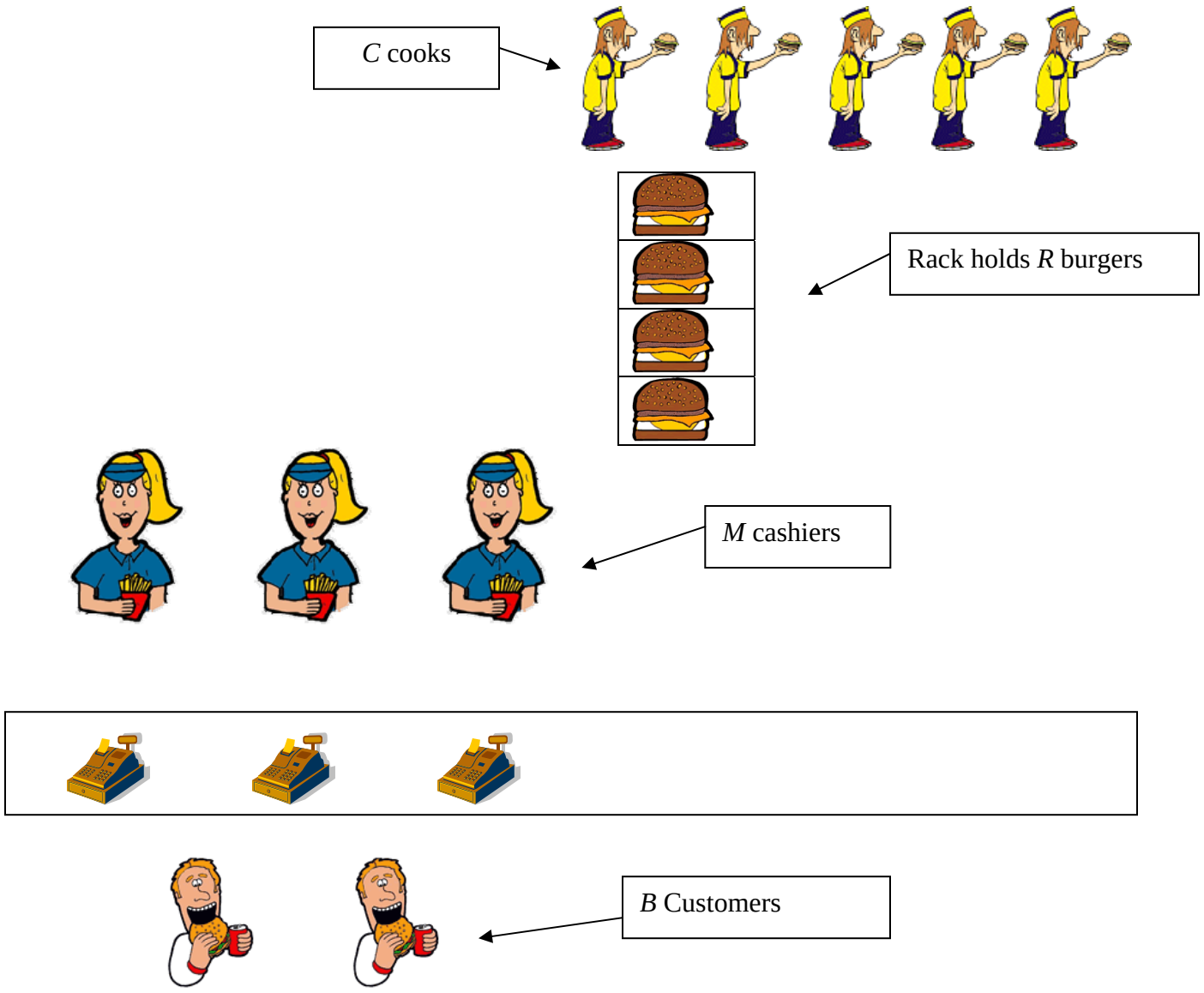
- Moe cannot plant a seed unless at least one empty hole exists, but Moe does not care how far Larry gets ahead of Moe.
- Curly cannot fill a hole unless at least one hole exists in which Moe has planted a seed, but the hole has not yet been filled. Curly does not care how far Moe gets ahead of Curly.
- 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.
(LarryMoeCurly.c →LCM)

2.[1] 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 can not order until the cashier is ready
- 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.

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



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)

3. Thread Scheduling

Write a scheduler for user level threads that works on round robin policy. A controller (factory) creates as many threads as required; it employs a timer that interrupts the scheduler at predetermined intervals to switch the currently running thread to a thread in front of a waiting queue. Parameters such as time interval and execution time for the scheduled thread have to be passed into a controller that coordinates the scheduler and a timer that maintains the intervals. You can use alarms and signals to control the threads. (ThreadSched.c → TS)

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 (10 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

10/28 submit on-line before mid-night.