CSE4/521 Introduction to Operating System Fall 2011
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:

1. (30 points) Bounded-buffer Producer/Consumer problem: Implement a solution to the bounded-buffer producer/consumer problem. Implement the solution using synchronization primitives (pthread_mutex_t for mutual exclusion and sem_t for minding the buffer size) provided by the POSIX threads. Your implementation will consist of a control program that (i) initializes the buffer and the synchronization variables and (ii) creates and terminates the threads for the producer and the consumer. The producer generates printable characters and places them into the buffer and the consumer pulls the characters out of the buffer one at a time and prints them out. Let the control program simulate all possible conditions: buffer empty (consumer waiting), buffer full (producer waiting), buffer not empty (both producer and consumer working). Assume that production and consumption per character take the same amount of time.

2. (35 points) Mother Hubbard (children, mother, father)
There are 12 children. Each child needs to be woken up, fed breakfast, sent to school, given dinner, given a bath, read a book, and tucked in bed. The mother applies each task in order, to each child in order, before being able to move on to the next task. Then after all the children have all the tasks done to them, the mother can take a nap break. After a child takes their bath, the father can read the child a book and tuck the child in bed. When all children are in bed, then the father can go to sleep. When the father goes to sleep, he wakes up the mother and the cycle repeats. (Note that the Mother can give a child a bath while the father read another child a book, but a child who has not had a bath yet, cannot get read a book.)
The following occurrences must be printed out at the right times:

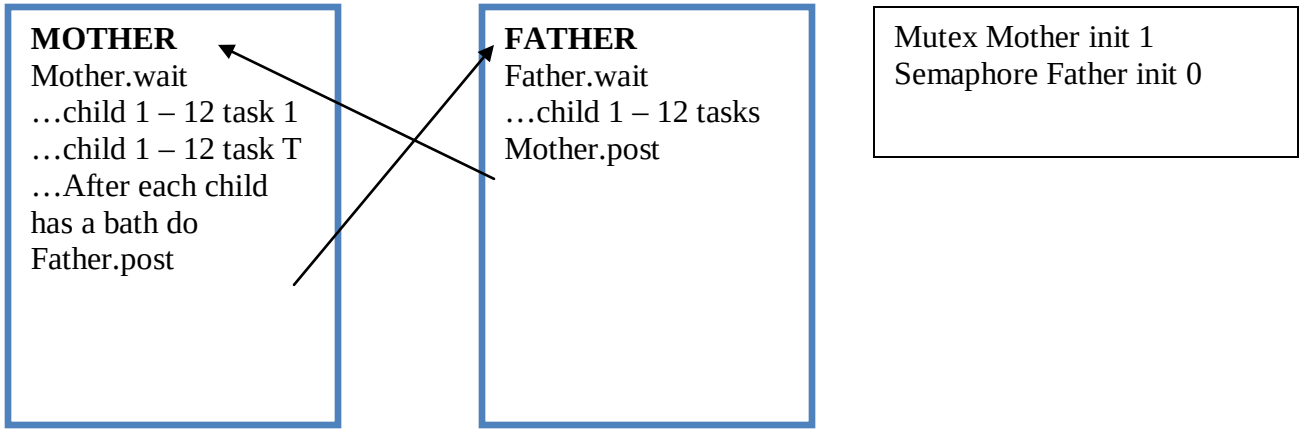
- When either the Mother or Father wakes up or goes to sleep.
    - i.e. “Father is going to sleep and waking up Mother to take care of the children.”
- When a child is fed breakfast, sent to school, given dinner, given a bath by the Mother
    - i.e. “Child #3 is being fed breakfast.”
- When a child is read a book and tucked in bed by the Father
    - i.e. “Child #11 is being read a book.”
- When the beginning/end of a cycle/day occurs and what # cycle/day it is.
    - i.e. “This is day #3 of a day in the life of Mother Hubbard.”

2.1 Implementation issues:

- Have this cycle go on for N times, as specified by argv[1] from the command line.
    - Program launched like so: ./mh 100
    - This will launch “Mother Hubbard” program for 100 cycles
    - Name your C program: “mh.c” with executable name “mh”
- Launch two threads: one for the Mother function, the other for the Father function
- When the Mother takes a nap break, she must be “blocked” (no busy-wait loops)

- When the Father goes to sleep, he must be "blocked" (no busy-wait loops)
- When the Mother performs a task on a child, call the usleep(100); command to slow down a bit.
- At the start of the program, the Mother should be awake and the Father is asleep.

Solution:



3. (35 points) Airline Passengers

There are P passengers, B baggage handlers, S security screeners, and F flight attendants. Each passenger must give their baggage to a baggage handler, then go have their "security" screened by a security screener, and finally be seated by a flight attendant. When all P passengers have this done, the plane takes off. All P passengers arrive at the same time to the terminal and must wait on a line to be processed, but are processed in no particular order (yes, seems unfair but that's the way it is at this airline, so stuff it! :P We know you had a choice, so thank you for choosing us!)

The following occurrences must be printed out at the right times:

- When a passenger is waiting to be processed.
  - i.e. "Passenger #61 arrived at the terminal."
  - i.e. "Passenger #12 is waiting at baggage processing for a handler."
  - i.e. "Passenger #33 is waiting to be screened by a screener."
  - i.e. "Passenger #23 is waiting to board the plane by an attendant."
  - i.e. "Passenger #5 has been seated and relaxes."

3.1 Implementation issues:

- Have this cycle go on for only one plane, using actor counts from the command line.
  - Program launched like so: ./airline 100 3 5 2
  - This will launch "Airline" program for 1 plane departure, using 100 passengers, 3 baggage handlers, 5 security screeners, 2 flight attendants.
  - Name your C program: "airline.c" with executable name "airline".
- Create B threads, one for each of the baggage handlers.
- Create S threads, one for each of the security screeners.

- Create F threads, one for each of the flight attendants.
- THEN create P threads, one for each of the passengers. (Do this last)
- When one of the actors is waiting, they should be "blocked" (meaning no "busy-wait" or "spin-loops")

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 internal documentation to explain your design. (-10 points for missing internal documentation).
- 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 April 2nd, 20111 Saturday; submit on-line before mid-night.