Project 2: Inter-Process Communication and Synchronization

Qinya Li
Department of Computer Science and Engineering
Shanghai Jiao Tong University
Spring 2023

Objectives

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

Enviroment

- VitrualBox/VMWare (optional)
- Ubuntu Linux (recommended)

What to Submit

- A“tar” file of your DIRECTORY, containing:
  - Your makefile
  - Any “.cc”, “.c”, and “.h” files
  - Any “readme” or “.pdf” files asked for in the project
  - A text file containing the runs of your programs for each of the project parts “typescript”
    - Do not submit ALL runs you have done, just the output required to demonstrate a successful (or unsuccessful) run
    - If you cannot get your program to work, submit a run of whatever you can get to work as you can get partial credit
    - Copy & paste, or use “>” to redirect output to a file
- DO NOT SUBMIT your object or executable files, remove them before you pack your directory

How to Submit

- Remove your ".o" files and executables

  rm *.o

  rm basic_server

  rm basic_client

- Pack your entire directory (including the directory)

  tar –cvf Prj2+StudentID.tar project2

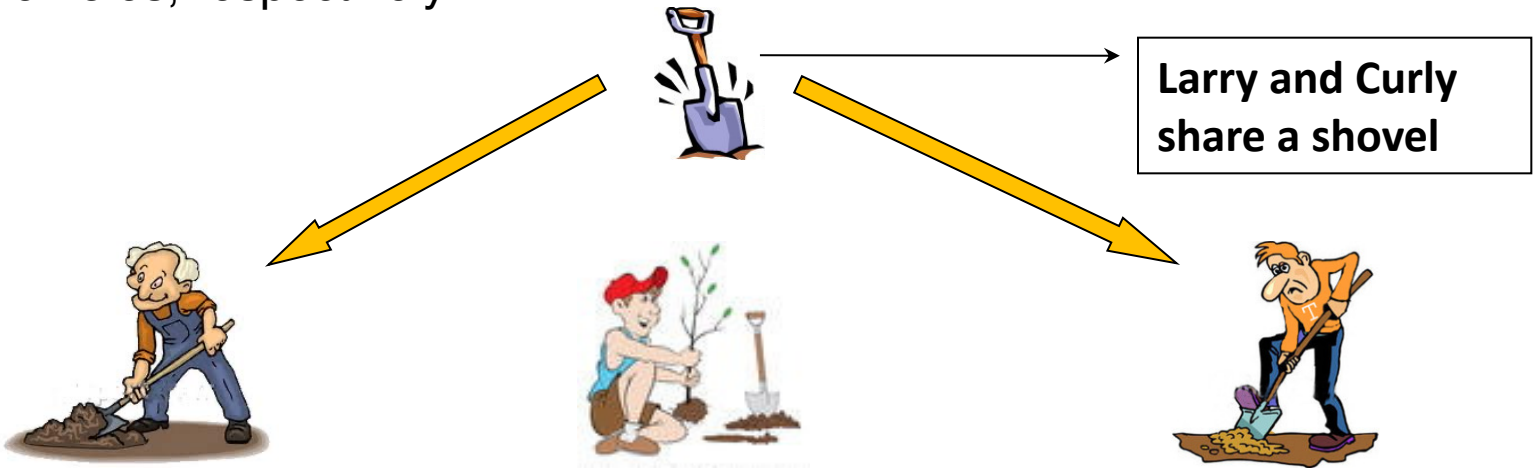
- Send your Prj2+StudentID.tar file to Canvas

Resources

- Unix programming
  - http://users.actcom.co.il/~choo/lupg/tutorials/multi-process/multi-process.html#%0Ashmem
- Posix Thread Programming
  - https://computing.llnl.gov/tutorials/pthreads/
- POSIX Semaphores
  - http://www.csc.villanova.edu/~mdamian/threads/posixsem.html

Problem 2.1 – Stooge Farmers Problem

- Larry digs the holes. Moe places a seed in each hole. Curly then fills the hole up.
- Moe cannot plant a seed unless at least one empty hole exists.
- Curly cannot fill a hole unless at least one hole exists in which Moe has planted a seed.
- 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.



Problem 2.1 – General Requirement

- Source file: LarryCurlyMoe.c
- Executable file: LCM
  - g++ LarryCurlyMoe.c –o LCM –lpthread
- Run: LCM Maxnum
  - Maxnum: Max number of unfilled holes
- Sample run: “./LCM 3”

Problem 2.1 – Sample Output

```
> ./LCM 3
Maximum number of unfilled holes: 3
Begin run.
Larry digs another hole #1.
Moe plants a seed in a hole #1.
Curly fills a planted hole #1.
Larry digs another hole #2.
Larry digs another hole #3.
Moe plants a seed in a hole #2.
Curly fills a planted hole #2.
Moe plants a seed in a hole #3.
Curly fills a planted hole #3.
……
Larry digs another hole #11.
Moe plants a seed in a hole #9.
Curly fills a planted hole #9.
Larry digs another hole #12.
Moe plants a seed in a hole #10.
Curly fills a planted hole #10.
End run.
```

Problem 2.1 – Functions You Will Need

- Header: #include <semaphore.h>
- sem_t: structure of semaphore

```
sem_t Shovel;
```

- sem_init: initialize semaphore
  - sem_init(sem_t *sem, int pshared, unsigned int value);

```
sem_init(&Shovel,0,1);
// set up a binary semaphore to act as a mutex for
   the critical section "use of shovel", initialize it to 1
```

- sem_destroy: destroy a semaphore

```
sem_destroy(&Shovel);
```

Problem 2.1 – Functions You Will Need

- sem_wait: wait for/block on a semaphore
- sem_post: signal/post on a semaphore

```
// Have Larry wait until he can use the shovel
// [in other words, wait to use the critical resource]
sem_wait(&Shovel);
// Larry now has the shovel, so dig the hole
// [thereby using the critical resource, in this case
// it is the global variable “HolesDug”]
printf("Larry digs another hole #%i.\n",++HolesDug);
// Larry is done with the shovel, so signal on the semaphore
// [thereby freeing the critical resource CR and unblocking
// a thread/process that is waiting on the CR]
sem_post(&Shovel);
```

Problem 2.1 – Understanding Semaphore

- **sem_wait**: decrease the value of the semaphore, and if the value is <0, it will BLOCK the thread
- **sem_post**: increase the value of the semaphore, and if the value becomes >= 0, the OS will unblock a blocked thread (a thread who called **sem_wait** and was blocked)

Problem 2.1 – Basic Program Flow

1. Create 3 functions called “Larry”, “Curly”, “Moe”
2. Create a few semaphores
3. Initialize the semaphores
4. Create 3 threads, run the three functions from step 1 on each of the threads.
5. Run for AT LEAST 100 steps (so have at least 100 holes filled by Curly)

Problem 2.1 – What you want to see...

- You need to see some interleaving of the functions running on the threads.
- At first, one thread will execute and dig some holes, then another will seed the holes, then the other will fill the holes
- Then, given enough time, there should be an interleaving showing a dig, a seed, and a fill
  - sleep(rand() % NUM);

Problem 2.2 – Faneuil Hall Problem

- Immigrants, Judges, and Spectators are each modeled as a thread.
- Actions and constraints.

Immigrants

- Actions
  - enter
  - checkIn
  - sitDown
  - swear
  - getCertificate
  - leave
- Constraints
  - Can’t enter and leave when the judge is in the building.
  - Can’t be confirmed unless they have sat down.

Judges

- Actions
  - enter
  - confirm
  - leave
- Constraints
  - Can’t confirm until all the immigrants who have entered, have already checked in.

Spectators

- Actions
  - enter
  - spectate
  - leave
- Constraints
  - Can’t enter when the judge is in the building.

Tips

- Add random delays. (Simulate time passing.)
  - sleep(rand() % NUM);
- Output as much information as you can.

Problem 2.2 – Sample Output

```
> ./faneuil
Immigrant #0 enter
Immigrant #0 checkIn
Immigrant #0 sitDown
Judge #0 enter
Judge #0 confirm the immigrant #0
Judge #0 leave
Immigrant #0 getCertificate
Immigrant #0 leave
Spectator #0 enter
Immigrant #1 enter
Immigrant #1 checkIn
Immigrant #1 sitDown
Spectator #0 spectate
Spectator #0 leave
Judge #1 enter
Judge #1 confirm the immigrant #1
Immigrant #1 getCertificate
……
```