15213 Lecture 26: Synchronization Advanced

Learning Objectives

- Recognize the uses of mutexes and semaphores in ensuring consistent execution in parallel programs.
- Identify different models of parallelism such as producer-consumer and readers-writers.
- Compare and contrast different concurrency issues when parallel programming.

1 Getting Started

The directions for today's activity are on this sheet, but refer to accompanying programs that you'll need to download. To get set up, run these commands on a shark machine:

1. `$ wget http://www.cs.cmu.edu/~213/activities/lec26.tar`
2. `$ tar xf lec26.tar`
3. `$ cd lec26`

2 Algorithm Design

2.1 Tracking Multiple Requests

A new mall, the Carnegie Mall, is opening up near Carnegie Mellon, and it has been the talk of the town for years. There's only one problem: the mall owners anticipate extremely high traffic on opening day, leading to long wait times and overcrowding which could harm customer satisfaction. As a parallel-guru yourself after taking 213, you've been tasked to track the influx of traffic into the mall from all doors to ensure that the occupancy does not reach above a certain threshold. Let us design a data structure to keep track of the occupants of this mall, as well as the customers entering and exiting.

1. Use mutexes/semaphores to help design your data-structure fields, and describe what the initial values of these fields will be. Let 213 be the maximum number of people allowed in the mall at once. Here we will assume a thread is created each time a person leaves or enters a door, and multiple people could enter/leave different doors at once.

2. What operations would you use in order to track when people enter/exit. When would it be safe to let an additional person in?

3. What happens when your mall reaches full occupancy?

2.2 Multi-Lock Data-Structs

The original mall schematics called for all doors to be 20ft wide by 7ft tall to accompany multiple people entering and exiting at the same time through each door. Yet a miscommunication with the construction company led to the doors being 7ft wide and 20 ft tall (whoops!), meaning only one person can exit or enter at a time. If one person tries to exit while another person tries to enter, then only one will be allowed through while the other will have to wait. There are 15 doors total in the mall, and entering/exiting a door takes 5 seconds.

4. Identify the resources for which people entering/exiting the building are competing. What addition do you need to make to your original data structure?

2.3 Parallel Bugs

With minutes before the mall's opening, you scratch down a quick idea and implement it into the system. On opening day, you notice that despite people entering and leaving the building, the count of people in the mall never changes. Below is your algorithm, where `get_door` will attempt to lock the door and `get_spot_in_mall` will attempt to get a spot in the mall if any are available. The function `sleep` is used to denote that passing through a door takes 5 seconds.

```
Enter:
  get_door(doorID);
  get_spot_in_mall();
  sleep(5000);
  return_door(doorID);
Exit:
  get_door(doorID);
  sleep(5000);
  return_spot_in_mall();
  return_door(doorID);
```

5. Identify a potential error in the code and the type of concurrency issue present. Can you think of a specific scenario that would cause this error?

6. How would you fix the above code?

This system that was just designed for the mall is a good example of the producer-consumer problem, where people entering the mall are consuming eligible spots for occupying, while people leaving the mall are producing new available spots. Thus, it is important to give our producers priority to produce so that consumers may have something to consume.

3 Readers-Writers Problem

Examine the file `writeError.c` using your editor of choice. The program spawns multiple pthreads, where thread i updates the contents of the buffer at the i'th index with number i. Once you're comfortable with the program, build (`$ make writeError`) and run it (`$ ./writeError`).

7. The program prints each element of the buffer, where we expect the output to print numbers 0 to 9. Run the program multiple times. Is the output accurate? Is it consistent?

8. Please fix the program to correctly output numbers 0 through 9 each time it is run. Describe the adjustments you made.

This next task focuses on the 'first' readers-writers problem, where no reader will be forced to wait unless a writer has already obtained an exclusive lock. Examine the files `readWrite.c` and `multiReadWrite.c` using your editor of choice. The program executes 30 instances of reads and writes interleaved, where "Read" and "Write" are printed the order they occur. Once you're comfortable with the program, build (`$ make readWrite`) (`$ make multiReadWrite`) and run each with (`$ ./readWrite`) (`$ ./multiReadWrite`).

9. Compare the outputs of the two programs. Can you describe what is happening in `multiReadWrite.c` compared to `readWrite.c`?

10. Examine the contents of `multiReadWrite.c`. How does this help explain the behavior to the previous question?

11. Can you think of one downside to this approach in `multiReadWrite.c`? (HINT: consider what would happen if we increased the execution time of a read operation.)

4 *(Advanced)* Concurrency Issues

Professors at Hogwarts noticed that students spend a lot of their time fooling around and aren't focusing on preparing for the OWLs, which are round the corner. Keeping this in mind, Professor Dumbledore launched a new scheme wherein only those students who complete their Potions assignments would get to visit Hogsmeade. However, there's a catch. In order to prevent collaboration and cheating, professors have announced that only a single student is allowed to access the Potions section of the library at a time. Moreover, any student working on the assignment must use only

the magic ink and the anti-cheating parchment in the library. There are 50 students working on the assignment.

To make the task easier, the following procedure was laid out for students:

- Claim the ink and the parchment.
- Use the ink to take note of what questions you're looking to seek answers for.
- After you're done with question writing, release the ink.
- Hold the parchment and wait until you have access to the Potions books in the library.
- After you're done reading the books, leave them. Go back to claim the ink and write the solutions on the parchment.
- Release the ink and the parchment.

One of the students wrote the following code to simulate the given procedure:

```
sem_t ink;
sem_t parchment;
sem_t books;

int main(int argc, char** argv)
{
int P = 1, I = 1, B = 1, S = 50;
sem_init(&ink, 0, I);
sem_init(&parchment, 0, P);
sem_init(&books, 0, B);

int* students = malloc(S * sizeof(int));
launch_threads();
reap_threads();
}

void launch_threads()
{
  for(int i = 0; i < S; i++)
  {
    students[s] =
pthread_create(students+i,
                  NULL, student_fn, NULL);
  }
}

void reap_threads()
{
  for(int i = 0; i < S; i++)
  {
    pthread_join(students[i]);
```

```
  }
}

void* student_fn(void* varp)
{
  P(&ink);
  P(&parchment);
  sleep(genrand() % 60); // Time for you to write questions

  V(&ink);
  P(&books);
  sleep(genrand() % 900); // Time for you to study the books

  V(&books);
  P(&ink);
  sleep(genrand() % 60); // Time for you to write solutions

  V(&parchment);
  V(&ink);

  return 0;
}
```

However, the student noticed that sometimes the code gets stuck in a deadlock, and they're unsure why.

12. Which of the following do you think caused the deadlock?
    - Parchment and ink
    - Parchment and book
    - Books and ink

13. Can we fix this without changing the procedure? If so, how?