



Classic Synchronization Problems (Reader-Writer Problem) Midterm Review Topics

Lecture 19

Klara Nahrstedt

CS241 Administrative

- Read Chapter 5.6 in Stallings
- Homework 1 due 3/2 4pm in Anda Ohlsson's Office
- **MIDTERM – MONDAY, March 5, 11am (will talk about Midterm at the end of lecture)**

Outline

- Readers/ Writers Problem

First Reader-Writer Problem

- readers: read data
- writers: write data
- Rule:
 - Multiple readers can read the data simultaneously
 - Only one writer can write the data at any time
 - A reader and a writer cannot in critical section together.
- Locking table: whether any two can be in the critical section simultaneously

	Reader	Writer
Reader	OK	No
Writer	NO	No

First Readers-Writers Problem (Readers have priority)

Let processes reading do so concurrently

Let processes writing do so one at a time

Introduce semaphores

```
Semaphore mutex = 1;
```

```
Semaphore wrt = 1;
```

```
int readc = 0; /* reader counter */
```

Writer

```
while (TRUE) {
```

```
Think_up_data(); /*noncritical  
section*/
```

```
lock(&wrt);
```

```
do writing
```

```
unlock(&wrt);
```

```
}
```

Reader

```
while (TRUE) {
```

```
lock(&mutex);
```

```
readcount = readcount+1;
```

```
if readcount == 1 then  
lock(&wrt);
```

```
unlock(&mutex);
```

```
do reading
```

```
lock(&mutex)
```

```
readcount=readcount-1;
```

```
if readcount == 0 then  
unlock(&wrt);
```

```
unlock(&mutex);
```

```
Use read data }
```

First Reader-Writer Solution

Does it work? What if?

Problem with this solution

Mutex m, wrt;

int readcount, // shared and initialized to 0

// Writer

lock(&wrt),

.....

writing performed

.....

lock(&wrt),

// Reader

lock(&m);

readcount = readcount + 1;

if (readcount == 1) lock(&wrt);

unlock(&m);

....

reading performed

lock(&m);

readcount = readcount - 1;

if (readcount == 0) unlock(&wrt);

unlock(&m);

Midterm Review

- Review
 - Lecture Notes
 - All quizzes until today
 - SMP Quizzes and Regular Quizzes
 - Review homework 1 solutions
 - Review Stallings and R&R book material
- **Midterm - Monday 11-11:50am, 1404 SC**
- **Review Session – Sunday 2-3:30pm in 1404 SC**
 - You need to bring questions, TAs will respond
- Midterm
 - closed book, closed notes,
 - NO calculator or other calculating electronic devices
 - No cell-phones

Topics: Hardware/OS Overview

- Chapter 1.1-1.7 (Stallings)
- Chapter 2.1-2.2 (Stallings)
- Chapter 1 (R&R)
- Keywords Need to know
 - Processors – registers
 - Interrupts and Interrupt Handling
 - Polling and Programmed I/O
 - Basic Memory principles
 - Kernel mode, user mode
 - Multiprogramming, uni-programming
 - Time sharing
 - Buffer Overflow and security

Topics: Processes

- Chapter 3.1-3.4 (Stallings)
- Chapter 2 and 3 (R&R)
- Keywords need to know
 - What is process?
 - What is the difference between process and program?
 - What is the program image layout?
 - Understand argument arrays
 - What does it mean to have a thread-safe function?
 - What is the difference between static and dynamic variables?
 - What are the major process states?
 - What is the difference between dispatcher and scheduler?
 - What is PCB?
 - What happens when process switches from running to ready state?
 - What is process chain, process fan?

Topics: Threads

- Chapter 12.1-12.5 (R&R)
- Chapter 4.1, 4.5 (Stallings)
- Keywords need to know
 - What is the difference between processes and threads?
 - What is the difference between user-level threads and kernel-level threads?
 - Detaching and joining threads
 - What happens if you if you call `exit(1)` in a thread?
 - What is a graceful way to exit a thread without causing process termination?

Topics: Concurrency (Mutual Exclusion)

- Chapter 14.1-14.3 (R&R)
- Chapter 5.1-5.3 (Stallings – and don't forget Appendix A about the Software Solutions)
- Keywords need to know
 - What are the four conditions to provide appropriate synchronization and enter critical region?
 - What is the difference between counting semaphore and mutex?
 - What do `sem_wait` and `sem_post` do?
 - How can counting semaphores be implemented using binary semaphores?
 - How can `test_and_set` be used for synchronization?
 - How can you make a function atomic?
 - Consider increment (`i++`) and decrement function (`i--`). How do you ensure that race condition does not occur on the shared variable 'i' when two processes use them at the same time?

Topics: Thread Synchronization

- Chapter 13.1-13.2 (R&R)
- Keywords to know
 - What are mutex locks?
 - How do you initialize mutex locks?
 - When would you use mutex instead of counting semaphore?
 - When would you use counting semaphore instead of mutex?
 - Are mutex functions interrupted by signals?

Topics: Scheduling

- Chapter 9.1-9.2 Scheduling (Stallings)
- Keywords need to know
 - Scheduling policies
 - FCFC, SJF, Round Robin, Priority Scheduling
 - Preemptive vs. Non-Preemptive Scheduling
 - Queues in Process management – what is ready queue? How are process states related to process management queues?
 - What is average waiting time?
 - What is the difference between process waiting time and turn-around time?

Topics: Signals

- Chapter 8.1-8.5 (R&R)
- Keywords need to know
 - Signal basic concepts – generating signals, blocked, pending signals, delivered signals, ignored signals, ...
 - What is signal mask and what are the operations to modify signal mask?
 - What is signal handler?
 - What is the role of sigaction?
 - How do you wait for signals?

Topics: Timers

- Chapter 9.1-9.3 (R&R)
- Keywords need to know
 - Understand what the various time functions are for
 - Gettimeofday
 - Understand the different clock resolutions
 - Sleep function
 - What are time intervals? What are they great for?

Topics: Classical Sync Problems

- Chapter 5.3 and 6.6 (Stallings)
- Keywords need to know
 - What is the producer-consumer problem?
 - What are the various semaphores in the producer/consumer solution for?
 - What is the dining philosopher problem?
 - What is the danger of a simple solution for dining philosopher problem?

Summary

Good Luck with the Exam!!!

Please, come to class little earlier (10:55am) so that we can start at 11:00 exactly.