CS 241 Section Week #7 (10/22/09)

Topics This Section

- Midterm Statistics
- MP5 Forward
- Classical Synchronization Problems
- Problems

Midterm Statistics

Avg: 18.6
Median: 19
Max: 25
Min: 10
Standard Deviation: 3.3

MP5 Overview

MP5 Overview

This is your first long MP. You have two weeks to complete it.

You are to create a "deadlock resilient semaphore" library. You should implement six functions.

You need to implement:

- Deadlock prevention
  - enforce a global ordering on all locks
  - Locks should be acquired in descending order
- Deadlock avoidance
  - No cycle exist in a wait-for graph
  - You need to implement the cycle detection algorithm

MP5 Overview

- Deadlock detection
  - Periodically incur the cycle detection algorithm
  - once a deadlock is detected, you need only send a SIGINT signal. The library does NOT need to worry about how SIGINT is handled.

Since we only allow one instance of each resource, you do not need to implement the Banker's algorithm for deadlock prevention. You may use a resource allocation graph instead.

The given test cases are far from complete. You should derive your own test cases.

Classical Synchronization Problems

Example 1: Producer-Consumer Problem

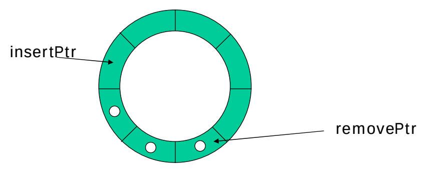
Producers insert items

Consumers remove items

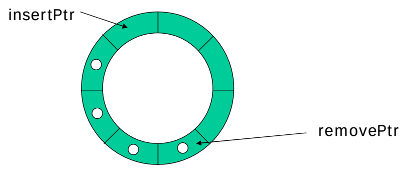
Shared bounded buffer

e.g. a circular buffer with an insert and a removal pointer.

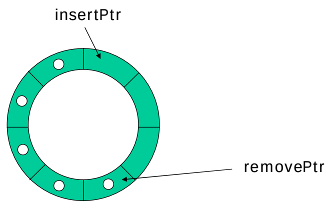
Producer-Consumer



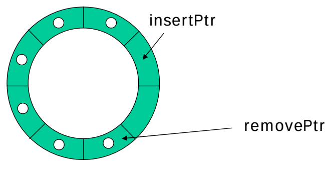
Producer-Consumer



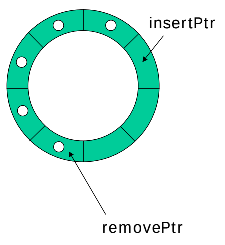
Producer-Consumer



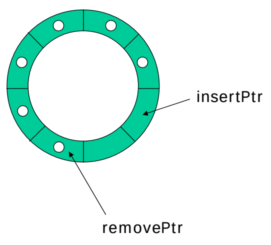
Producer-Consumer



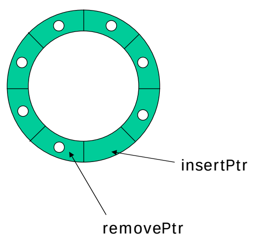
Producer-Consumer



Producer-Consumer

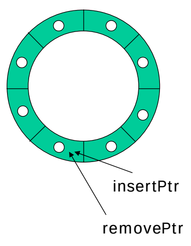


Producer-Consumer

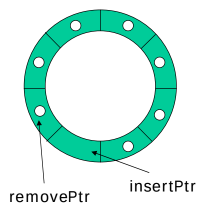


Producer-Consumer

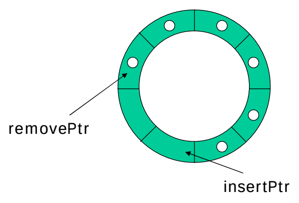
BUFFER FULL: Producer must be blocked!



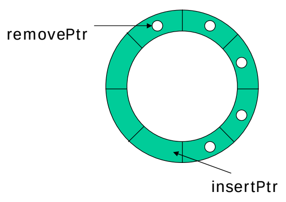
Producer-Consumer



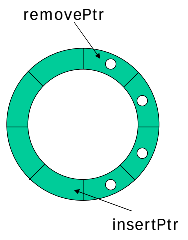
Producer-Consumer



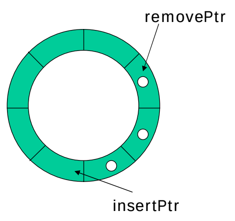
Producer-Consumer



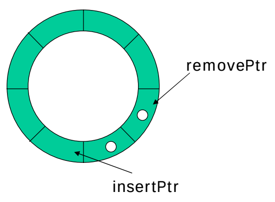
Producer-Consumer



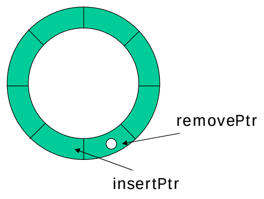
Producer-Consumer



Producer-Consumer

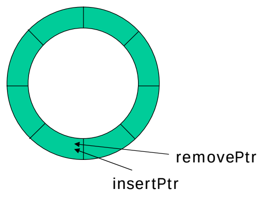


Producer-Consumer



Producer-Consumer

BUFFER EMPTY: Consumer must be blocked!



Challenge

Need to prevent:

Buffer Overflow
   Producer writing when there is no storage
Buffer Underflow
   Consumer reading nonexistent data
Race condition
   Two processes editing the list at the same time

Synchronization variables

Create these variables to prevent these problems:

`items` semaphore
   Counts how many items are in the buffer
   Cannot drop below 0
`slots` semaphore
   Counts how may slots are available in the buffer
   Cannot drop below 0
`list` mutex
   Makes buffer access mutually exclusive

Producer-Consumer Example

`ds7-problem1.c` shows an example implementation for one producer and one consumer, but without synchronization code.

- Running it shows
  - Buffer underflows
    - Nonsense data is consumed
  - Buffer overflows
    - Unconsumed data is overwritten

- *Think*: When should the consumer block? When should the producer block?

Example 2: Dining Philosophers

Dining Philosopher Challenge

{ Think | Eat }

N Philosophers circular table with *N* chopsticks

To eat the Philosopher must first pickup two chopsticks

*i*th Philosopher needs *i*th & *i*+1st chopstick

Only put down chopstick when Philosopher has finished eating

Devise a solution which satisfies mutual exclusion but avoids starvation and deadlock

The simple implementation

```
while(true) {
    think()
    lock(chopstick[i])
    lock(chopstick[(i+1) % N])
    eat()
    unlock(chopstick[(i+1) % N])
    unlock(chopstick[i])
}
```

Does this work?

Deadlocked!

When every philosopher has picked up his left chopstick, and no philosopher has yet picked up his right chopstick, no philosopher can continue.

Each philosopher waits for his right neighbor to put a chopstick down, which he will never do.

This is a *deadlock*.

Formal Requirements for Deadlock

Mutual exclusion
- Exclusive use of chopsticks

Hold and wait condition
- Hold 1 chopstick, wait for next

No preemption condition
- Cannot force another to undo their hold

Circular wait condition
- Each waits for next neighbor to put down chopstick

The simple implementations satisfies all of these.

Problems for Week 7 (contd)

2) `ds7-problem2.c` contains dining philosophers code.

Alter the program to prevent deadlock. There are multiple ways to do this.

Think: What are the conditions of deadlock? Can any of them be removed?