Using Semaphores

CS 241

March 12, 2014

University of Illinois

Slides adapted in part from material accompanying Bryant "Computer Systems: A Programmer's Perspective", 2/E
Announcements

Midterm

• Grades by Friday
• No discussion of solutions yet
  ▪ We will discuss them next week

Brighten lecturing now

• Office hours: Mondays right after lecture

Today

• Using semaphores: the producer-consumer problem
• Using semaphores: the readers-writers problem
Using Semaphores
Review: Semaphores

Problem: coordinating simultaneous access to shared data

```c
int cnt = 0;  // Shared data

void * worker( void *ptr )
{
    int i;
    for (i = 0; i < ITERATIONS_PER_THREAD; i++)
        cnt++;
}
```

Critical section
(just one line in this simple example)

Solution: mutually exclusive access to critical region
- Only one thread/process accesses shared data at a time
Semaphores for mutual exclusion

Basic idea

• Associate a unique semaphore *mutex*, initially 1, with each shared variable (or related set of shared variables)
• Surround corresponding critical sections with \textit{wait}(\textit{mutex}) and \textit{post}(\textit{mutex}) operations.

Terminology

• **Binary semaphore**: semaphore whose value is always 0 or 1
• **Mutex**: binary semaphore used for mutual exclusion
  ▪ \textit{wait} operation: “locking” the mutex
  ▪ \textit{post} operation: “unlocking” or “releasing” the mutex
  ▪ “Holding” a mutex: locked and not yet unlocked
• **Counting semaphore**: used to count a set of available resources
Before: Basic use of semaphores

```c
void * worker( void *ptr )
{
    int i;
    for (i = 0; i < ITERATIONS_PER_THREAD; i++) {
        sem_wait(&cnt_mutex);
        cnt++;
        sem_post(&cnt_mutex);
    }
}
```
Today: Advanced use of semaphores
Using semaphores:
The Producer-Consumer Problem
Producer-consumer problem

Chefs cook items and put them on a conveyer belt

Waiters pick items off the belt
Producer-consumer problem

Now imagine many chefs!

...and many waiters!
Producer-consumer problem

A potential mess!
Producer-consumer problem

Chef (Producer)

Waiter (Consumer)

inserts items

removes items

Shared resource: bounded buffer

Efficient implementation: circular fixed-size buffer
Shared buffer

Chef (Producer)

Waiter (Consumer)
Shared buffer

Chef (Producer)

What does the chef do with a new pizza?

insertPtr

Waiter (Consumer)

Where does the waiter take a pizza from?

removePtr
Shared buffer

Chef (Producer)

Waiter (Consumer)

insertPtr

removePtr

Insert pizza
Shared buffer

Chef (Producer)

Waiter (Consumer)

Insert pizza

insertPtr

removePtr
Shared buffer

Chef (Producer)

Waiter (Consumer)

Insert pizza
Shared buffer

Chef (Producer)

Waiter (Consumer)

Chef (Producer)

Waiter (Consumer)

InsertPtr

RemovePtr

Remove pizza
Shared buffer

Chef (Producer)

Waiter (Consumer)

Insert pizza

insertPtr

removePtr
Shared buffer

Chef (Producer)

Waiter (Consumer)

Insert pizza

insertPtr

removePtr
Shared buffer

Chef (Producer)

Waiter (Consumer)

BUFFER FULL:
Producer must wait!

Insert pizza

insertPtr

removePtr
Shared buffer

Chef (Producer)

Remove pizza

Waiter (Consumer)
Shared buffer

Chef (Producer)

Waiter (Consumer)

removePtr

insertPtr

Remove pizza
Shared buffer

Chef (Producer)

Waiter (Consumer)

Remove pizza

removePtr

insertPtr
Shared buffer

Chef (Producer)

Remove pizza

Waiter (Consumer)
Shared buffer

Chef (Producer)

Waiter (Consumer)

Remove pizza
Shared buffer

Chef (Producer)

Waiter (Consumer)

Remove pizza
Shared buffer

Chef (Producer)

Waiter (Consumer)

Remove pizza
Shared buffer

Chef (Producer)

Waiter (Consumer)

Buffer empty: Consumer must be blocked!

Remove pizza

insertPtr

removePtr

STOP
Designing a solution

Chef (Producer)
- Wait for empty slot
- Insert item
- Signal item arrival

Waiter (Consumer)
- Wait for item arrival
- Remove item
- Signal empty slot available

What synchronization do we need?
Designing a solution

Chef (Producer)
- Wait for empty slot
- Insert item
- Signal item arrival

Wait for item arrival

Mutex (shared buffer)

Wait for empty slot available

Remove item

Waiter (Consumer)
- Wait for item arrival
- Signal empty slot available

What synchronization do we need?
Designing a solution

Chef (Producer)

Wait for empty slot
Insert item
Signal item arrival

Waiter (Consumer)

Wait for item arrival
Remove item
Signal empty slot available

Semaphore (# empty slots)

What synchronization do we need?
Designing a solution

Chef (Producer)
- Wait for empty slot
- Insert item
- Signal item arrival

Semaphore (filled slots)

Waiter (Consumer)
- Wait for item arrival
- Remove item
- Signal empty slot available

What synchronization do we need?
Producer-Consumer Code

Critical Section: move insert pointer

\[
\text{buffer[ insertPtr] = data;}
\]

\[
\text{insertPtr = (insertPtr + 1) % N;}
\]

Critical Section: move remove pointer

\[
\text{result = buffer[removePtr];}
\]

\[
\text{removePtr = (removePtr +1) % N;}
\]
Producer-Consumer Code

Counting semaphore – check and decrement the number of free slots

```c
sem_wait(&slots);
mutex_lock(&mutex);
buffer[insertPtr] = data;
insertPtr = (insertPtr + 1) % N;
mutex_unlock(&mutex);
sem_post(&items);
```

Block if there are no free slots

Counting semaphore – check and decrement the number of available items

```c
sem_wait(&items);
mutex_lock(&mutex);
result = buffer[removePtr];
removePtr = (removePtr + 1) % N;
mutex_unlock(&mutex);
sem_post(&slots);
```

Done – increment the number of available items

Done – increment the number of free slots
Consumer Pseudocode: getItem()

```c
sem_wait(&items);
pthread_mutex_lock(&mutex);
result = buffer[removePtr];
removePtr = (removePtr +1) % N;
pthread_mutex_unlock(&mutex);
sem_signal(&slots);
```

Error checking/EINTR handling not shown
Producer Pseudocode: putItem(data)

```c
sem_wait(&slots);
pthread_mutex_lock(&mutex);
buffer[insertPtr] = data;
insertPtr = (insertPtr + 1) % N;
pthread_mutex_unlock(&mutex);
sem_signal(&items);
```

Error checking/EINTR handling not shown
Readers-Writers Problem
Readers-Writers Problem

Generalization of the mutual exclusion problem

Problem statement:
- *Reader* threads only read the object
- *Writer* threads modify the object
- Writers must have exclusive access to the object
- Unlimited number of readers can access the object

Occurs frequently in real systems, e.g.,
- Online airline reservation system
- Multithreaded caching Web proxy
A solution

Does it work?
https://www.surveymonkey.com/s/82RYXFT

Shared:

```c
int readcnt;    /* Initially = 0 */
sem_t mutex, w; /* Both initially = 1 */
```

Writers:

```c
sem_wait(&w);
/* Critical section */
/* Writing here */
sem_post(&w);
```

Readers:

```c
sem_wait(&mutex);
readcnt++;
if (readcnt == 1) /* First reader in */
    sem_wait(&w); /* Lock out writers */
sem_post(&mutex);

/* Main critical section */
/* Reading would happen here */

sem_wait(&mutex);
readcnt--;
if (readcnt == 0) /* Last out */
    sem_post(&w); /* Let in writers */
sem_post(&mutex);
```

(full code online)
Variants of Readers-Writers

Favor readers
- No reader waits unless a writer is already in critical section
- A reader that arrives after a waiting writer gets priority over writer

Favor writers
- Once a writer is ready to write, it performs its write as soon as possible
- A reader that arrives after a writer must wait, even if the writer is also waiting

*Starvation* (thread waits indefinitely) possible in both cases
- Q: How could we fix this?
Summary

Synchronization: more than just locking a critical section

Semaphores useful for counting available resources
  • sem_wait(): wait for resource only if none available
  • sem_post(): signal availability of another resource

Multiple semaphores / mutexes can work together to solve complex problems
Solution favoring readers

Readers:

```c
void reader(void)
{
    while (1) {
        sem_wait(&mutex);
        readcnt++;
        if (readcnt == 1) /* First reader in */
            sem_wait(&w); /* Lock out writers */
        sem_post(&mutex);

        /* Main critical section */
        /* Reading would happen here */

        sem_wait(&mutex);
        readcnt--;
        if (readcnt == 0) /* Last out */
            sem_post(&w); /* Let in writers */
        sem_post(&mutex);
    }
}
```