Synchronization and Deadlocks

Jonathan Misurda
jmisurda@cs.pitt.edu

Race Condition

<table>
<thead>
<tr>
<th>Shared Data:</th>
</tr>
</thead>
<tbody>
<tr>
<td>tail</td>
</tr>
<tr>
<td>A[i]</td>
</tr>
</tbody>
</table>

Enqueue():
- $A[tail] = 20$
- $tail++$
- $A[tail] = 9$
- $tail++$

Thread 0
Thread 1

Critical Regions

Thread 0
Thread 1

Synchronization

- Scheduling can be random and preemption can happen at any time
- Need some way to make critical regions “atomic”
- Need help from the Operating System

Mutex

- MUTual EXclusion
- A mutex is a lock that only one thread can acquire
- All other threads attempting to enter the critical region will be blocked

Critical Sections

<table>
<thead>
<tr>
<th>Shared Data:</th>
</tr>
</thead>
<tbody>
<tr>
<td>tail</td>
</tr>
<tr>
<td>mutex</td>
</tr>
<tr>
<td>A[i]</td>
</tr>
</tbody>
</table>

Enqueue():
- lock(&mutex);
- $A[tail] = 20$
- $tail++$
- unlock(&mutex);

Thread 0

Blocked!

Locked!
pthread_mutex_t

#include <stdio.h>
#include <pthread.h>

int tail = 0;
int A[20];

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

void enqueue(int value)
{
    pthread_mutex_lock(&mutex);
    A[tail] = value;
    tail++;
    pthread_mutex_unlock(&mutex);
}

Producer/Consumer Problem

Shared variables
#define N 10;
int buffer[N];
int in = 0, out = 0, counter = 0;

Producer
while (1)
{
    if (counter == N)
        sleep();
    buffer[in] = ...;
    in = (in + 1) % N;
    counter++;
    if (counter==N-1)
        wakeup(consumer);
}

Consumer
while (1)
{
    if (counter == 0)
        sleep();
    ... = buffer[out];
    out = (out+1) % N;
    counter--;
    if (count == N-1)
        wakeup(producer);
}

Deadlocks

• “A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause.”

• Caused when:
  1. Mutual exclusion
  2. Hold and wait
  3. No preemption of resource
  4. Circular wait

Condition Variables

• A condition under which a thread executes or is blocked

• pthread_cond_t

• pthread_cond_wait (condition, mutex)

• pthread_cond_signal (condition)

Producer/Consumer

#define N 10
int buffer[N];
int counter = 0, in = 0, out = 0, total = 0;

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t prod_cond = PTHREAD_COND_INITIALIZER;
pthread_cond_t cons_cond = PTHREAD_COND_INITIALIZER;

void *producer(void *junk)
{
    while(1)
    {
        pthread_mutex_lock(&mutex);
        if (counter == N)
            pthread_cond_wait(&prod_cond,
                               &mutex);
        buffer[in] = total++;
        printf("Produced: %d\n", buffer[in]);
        in = (in + 1) % N;
        counter++;
        if (counter == 1)
            pthread_cond_signal(&cons_cond);
        pthread_mutex_unlock(&mutex);
    }
}

void *consumer(void *junk)
{
    while(1)
    {
        pthread_mutex_lock(&mutex);
        if (counter == 0)
            pthread_cond_wait(&cons_cond,
                               &mutex);
        printf("Consumed: %d\n", buffer[out]);
        out = (out+1) % N;
        counter--;
        if (counter == N-1)
            pthread_cond_signal(&prod_cond);
        pthread_mutex_unlock(&mutex);
    }
}

Semaphores

• A lock that remembers “missed” wakeups

• Mutexes are a special case of Semaphores that only count to 1
Producer/Consumer

```c
#include <semaphore.h>
#define N 10
int buffer[N];
int counter = 0, in = 0, out = 0, total = 0;
sem_t semmutex, semempty; // sem_init(&semmutex, 0, 1); in main()
sem_t semfull; // sem_init(&semfull, 0, 0); in main()

void *producer(void *junk) {
    while(1) {
        sem_wait(&semempty);
        pthread_mutex_lock(&mutex);
        buffer[in] = total++;
        printf("Produced: %d\n", buffer[in]);
        in = (in + 1) % N;
        counter++;
        pthread_mutex_unlock(&mutex);
        sem_post(&semfull);
    }
}

void *consumer(void *junk) {
    while(1) {
        sem_wait(&semmutex);
        sem_wait(&semfull);
        printf("Consumed: %d\n", buffer[out]);
        out = (out + 1) % N;
        counter--;
        sem_post(&semmutex);
    }
}
```

Deadlock!

```c
#include <semaphore.h>
#define N 10
int buffer[N];
int counter = 0, in = 0, out = 0, total = 0;
sem_t semmutex; // sem_init(&semmutex, 0, 1); in main()
sem_t semempty; // sem_init(&semempty, 0, 0); in main()
sem_t semfull; // sem_init(&semfull, 0, 0); in main()

void *producer(void *junk) {
    while(1) {
        sem_wait(&semmutex);
        sem_wait(&semempty);
        buffer[in] = total++;
        printf("Produced: %d\n", buffer[in]);
        in = (in + 1) % N;
        counter++;
        sem_post(&semfull);
        sem_post(&semmutex);
    }
}

void *consumer(void *junk) {
    while(1) {
        sem_wait(&semmutex);
        sem_wait(&semfull);
        printf("Consumed: %d\n", buffer[out]);
        out = (out + 1) % N;
        counter--;
        sem_post(&semmutex);
        sem_post(&semmutex);
    }
}