# Lecture 15

CprE 308

February 16, 2015

# Intro

# Recap

- Previously introduced Threads implementation
- Mentioned Deadlocks...

# Today's Topics

- Producer Consumer
- Sleep and Wakeup

# Producer-Consumer

# Producer-Consumer

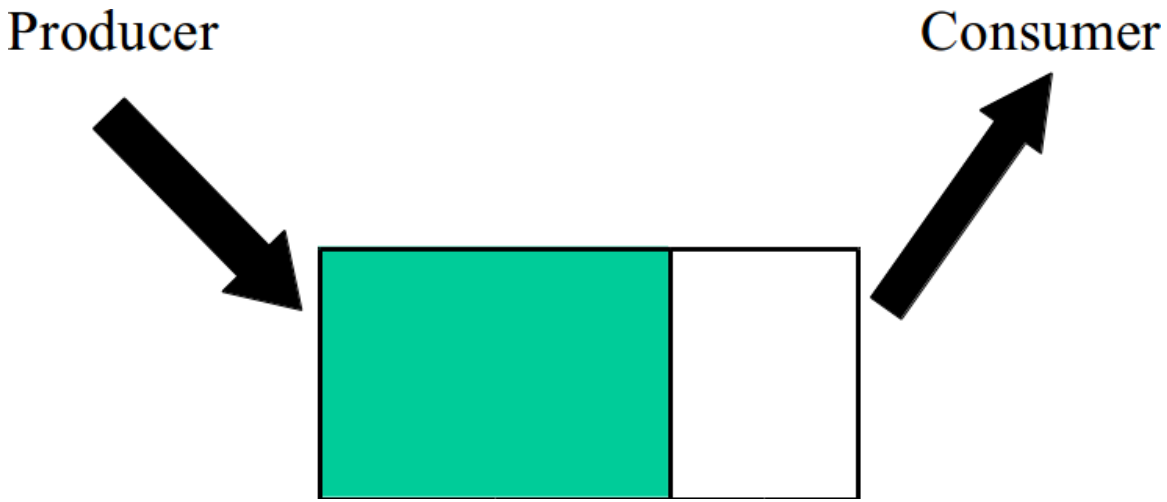


Figure 1:

- Mutual Exclusion
- Buffer Full
- Buffer Empty

# How is this solution?

# Producer

```
while(TRUE) {
   item = produce();
   insert(item,buffer);
   count++;
}
```

# Consumer

```
while(TRUE) {
  item = remove(buffer);
  count--;
  consume(item);
}
```

# How about this?

# Producer

```
while(TRUE) {
  item = produce();
  lock(mutex);
  insert(item,buffer);
  count++;
  unlock(mutex);
}
```

# Consumer

```
while(TRUE) {
  lock(mutex);
  item = remove(buffer);
  count--;
  unlock(mutex);
  consume(item);
}
```

# Producer-Consumer

- Cannot be solved by mutexes alone
- Need a way to block till some condition is satisfied
  - Condition variables (preferred with `pthreads`)
  - Semaphores

# Sleep and Wakeup

# Sleep and Wakeup Variables

# Shared Variables

- count (number of items in buffer)
- buffer
- N (maximum size of buffer)

# Sleep and Wakeup Example

# Producer

```
while(TRUE) {
  item = produce();
  if(count==N)
    sleep();
  insert(item,buffer);
  count++;
  if(count==1)
    wakeup(consumer);
}
```

# Consumer

```
while(TRUE) {
  if(count==0)
    sleep();
  item = remove(buffer);
  count--;
  if(count==N-1)
    wakeup(producer)
  consume(item);
}
```

# Sleep and Wakeup Example with Locks

# Producer

```
while(TRUE) {
  item = produce();
  if(count==N)
    sleep();
  lock(mutex);
  insert(item,buffer);
  count++;
  unlock(mutex);
  if(count==1)
    wakeup(consumer);
}
```

# Consumer

```
while(TRUE) {
  if(count==0)
    sleep();
  lock(mutex);
  item = remove(buffer);
  count--;
  unlock(mutex);
  if(count==N-1)
    wakeup(producer)
  consume(item);
}
```

# Semaphores

# Semaphore: Interface

- S: Integer value
- `Down(S)`:

```
when(S>0)
   S = S - 1;
```

- `Up(S)`:

```
S = S + 1;
```

- Atomic actions
- Down might block
- Up never blocks

# Semaphore: Implementation

`Down(S)`

- `If(S=0)` then
  - Suspend thread, put into a waiting queue
  - Schedule another thread to run
- Else decrement `S` and return

`Up(S)`

- Increment `S`
- If any threads in waiting queue, then
  - release one of them (make it runnable)

Both the above are done **atomically**

# Producer Consumer using Semaphores

# Shared Variables

- count (number of items in buffer)
- buffer
- N (maximum size of buffer)

# Semaphores

- Empty - semaphore initialized to N (number of free slots in buffer)
- Full - semaphore initialized to zero (number of items in buffer)

# Producer Consumer using Semaphores (Example)

# Producer

```
while(TRUE) {
  item = produce();
  down(Empty);
  lock(mutex);
  insert(item,buffer);
  count++;
  unlock(mutex);
  up(Full);
}
```

# Consumer

```
while(TRUE) {
  down(Full);
  lock(mutex);
  item = remove(buffer);
  count--;
  unlock(mutex);
  up(Empty);
  consume(item);
}
```

# (Blocking) Mutex - Special case of Semaphore

- Initialize Semaphore `S=1`
- Lock Mutex = `Down(S)`
- Unlock Mutex = `Up(S)`
- One Difference:
  - With `pthread_mutexes`, only the thread which currently holds the lock can unlock it
  - Semaphores have no such restriction

# Example (Game)

- Computer Game with multiple players
- Not more than 2 players in a room
- Semaphore `S`, initialize `S=2`
- Player executes
  - `Down(S)` before entering
  - `Up(S)` while leaving

# Producer Consumer using Semaphores with Mutexes

# Producer

```
while(TRUE) {
  item = produce();
  down(Empty);
  down(mutex);
  insert(item,buffer);
  up(mutex);
  up(Full);
}
```

# Consumer

```
while(TRUE) {
  down(Full);
  down(mutex);
  item = remove(buffer);
  up(mutex);
  up(Empty);
  consume(item);
}
```

# Example (Web Server)

- Web Server can handle only 10 threads at a time
  - Multiple points where threads are being created
  - How to ensure no more than 10 active threads?

# Example (Web Server)

- Web Server can handle only 10 threads at a time
  - Multiple points where threads are being created
  - How to ensure no more than 10 active threads?
- Semaphore with initial value = 10
  - `Down()` before thread creation
  - `Up()` once thread finishes

# POSIX Semaphores

- `man sem_overview`
- `int sem_init(sem_t *sem, int pshared, unsigned int value);`
- `int sem_wait(sem_t *sem); /* decrement */`
- `int sem_trywait(sem_t *sem);`
- `int sem_post(sem_t *sem); /* increment */`
- `int sem_getvalue(sem_t *sem, int *sval);`
- `int sem_destroy(sem_t *sem);`

# Question

- What if we changed the order of `lock()` and `down()` in producer/consumer example?

# Switching `lock()` and `down()`

# Producer

```
while(TRUE) {
  item = produce();
  down(Empty);
  lock(mutex);
  insert(item,buffer);
  count++;
  unlock(mutex);
  up(Full);
}
```

# Consumer

```
while(TRUE) {
  lock(mutex);
  down(Full);
  item = remove(buffer);
  count--;
  unlock(mutex);
  up(Empty);
  consume(item);
}
```

# Semaphore Example: Implementing `wait()` system call

- Parent process does a `wait()` system call on child
  - wait till child finishes before exiting
- What if parent executed `wait()` after child exited?
  - wait should return immediately

# Solution: Semaphore

- Semaphore `zombie`: initialize to 0
- Parent: `down(zombie)` inside `wait()`
- Child: `up(zombie)` upon exiting

# Condition Variables

# Condition Variables

- Allows a thread to wait till a condition is satisfied
- Testing if the condition must be done within a mutex
- With every condition variable, a mutex is associated

# Condition variables Code

- `pthread_cond_t condition_variable`
- `pthread_mutex_t mutex;`

# Waiting Thread

```
pthread_mutex_t(&mutex);
while(!cond. satisfied) {
  pthread_cond_wait(
    &condition_variable,
    &mutex);
}
// critical section
pthread_mutex_unlock(
  &mutex);
```

# Signaling Thread

```
pthread_mutex_lock(&mutex);
/* change variable value */
if(cond. satisfied) {
  pthread_cond_signal(
    &condition_variable);
}
pthread_mutex_unlock(
  &mutex);
```

# Condition variable and mutex

- A mutex is passed into wait: `pthread_cond_wait(cond_var,mutex)`
- Mutex is released before the thread sleeps
- Mutex is locked again before `pthread_cond_wait()` returns
- Safe to use `pthread_cond_wait()` in a while loop and check condition again before proceeding

# Example Usage

- Write a program using two threads
  - Thread 1 prints “hello”
  - Thread 2 prints “world”
  - Thread 2 should wait till thread 1 finishes before printing
- Use a condition variable

# Solved using condition variables

# Global

```
int thread1_done = 0;
pthread_cond_t cv;
pthread_mutex_t mutex;
```

# Thread 1

```
printf("hello");
pthread_mutex_lock(&mutex);
thread1_done = 1;
pthread_cond_signal(cv);
pthread_mutex_unlock(
  &mutex);
```

# Thread 2

```
pthread_mutex_lock(&mutex);
while(thread1_done == 0) {
  pthread_cond_wait(
    &cv, &mutex);
}
printf(" world\n");
pthread_mutex_unlock(
  &mutex);
```