Recitation 11

April 17, 2017
Topic

• Synchronization
  – Condition Variables
  – Semaphore
Why do we need Condition Variables?
-- Producer/Consumer Problem (1)

- Producer produces items in buffer
  - Producer i) locks buffer, ii) produces if space available, iii) unlocks buffer
  - Producer has to wait while buffer is full
- Consumer consumes items in buffer
  - Consumer i) locks buffer, ii) consumes if item available, iii) unlocks buffer
  - Consumer has to wait while buffer is empty
Why do we need Condition Variables?  
-- Producer/Consumer Problem (2)

• rec11-files/busywait.c
  – gcc -pthread -o busywait busywait.c
  – Run busywait

• What is the problem of busywait.c?
  – Consumes unnecessary CPU cycles
  – Depending on scheduling, if consumer thread never gets a chance to execute, \textit{count} == BUFSIZE will always be true, producer thread may busy wait forever
Why do we need Condition Variables?  
-- Producer/Consumer Problem (3)

• How to fix the problem of busywait.c?
• Need richer synchronization
  – Producer: if buffer full, “I’m going to sleep, don’t wake me up until condition buffer_not_full is met”
    • After producing an item, signal buffer_not_empty condition
  – Consumer: if buffer empty, “I’m going to sleep, don’t wake me up until condition buffer_not_empty is met”
    • After consuming an item, signal buffer_not_full condition
• Read code: condvar.c
  – Pay attention to the usage of pthread_cond_t, pthread_cond_wait, and pthread_cond_signal
Condition Variables API

Posix condition variables

```c
#include <pthread.h>

int pthread_cond_signal(pthread_cond_t* cond);
int pthread_broadcast(pthread_cond_t* cond);
int pthread_cond_wait(pthread_cond_t* cond,
                      pthread_mutex_t* mutex);

pthread_cond_t cond =
       PTHREAD_COND_INITIALIZER;
```
DIY

• Can you modify condvar.c for:
  – Multiple threads of consumers
  – Multiple threads of producers
  – Broadcast vs. Signal on the Condition Variables?
Semaphores: Review

• Understand Semaphores
  – The Value of Semaphores
    • number of available resources
  – P operation
    • (atomically) if value>0 block, otherwise decrement value
  – V operation
    • (atomically) increment value, release if anyone blocked
  – The Queue of Semaphores
    • list of threads waiting for the value to be >0
Semaphores API

Posix Semaphores

#include <semaphore.h>
int sem_wait(sem_t* sem);  // like P or down
int sem_post(sem_t* sem);  // like V or up

// pshared=0 => only threads of process can access
int sem_init(sem_t* sem, int pshared, 
         unsigned value);

sem_t sem;  // this is akin to create
Semaphores: Rewrite the Producer/Consumer Problem

BB: Posix semaphore (cont’d)

```c
void buffer_insert(item_t item){
    sem_wait(&producers_slots); //this is like a P()
    pthread_mutex_lock(&ring_access);
    buffer[in] = item;
    //count++; // NOTE no external state needed
    pthread_mutex_unlock(&ring_access);
    sem_post(&consumer_slots); //this is like a V()
}

<buffer_remove> on your own
```
Questions?