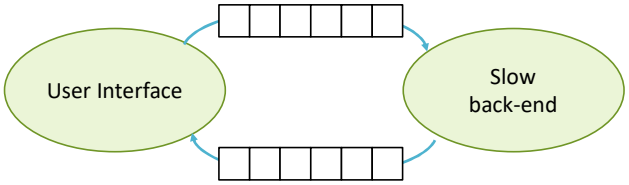
CS221
C and Systems Programming

REVIEW: Why use multiple threads?

Want to use more than one CPU core at once, to complete big jobs faster

Want part of the program to be more responsive i.e. have lower latency

Maximize Responsiveness

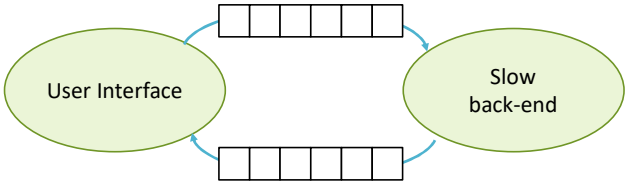


UI may produce a "burst" of jobs, faster than back-end can read them.

Back-end may produce a "burst" of results faster than UI can handle them.

We use Queues to store "jobs" for back-end and "results" for UI

Maximize Responsiveness



Synchronization:

- push "jobs" to back-end via a queue
- pull results from back-end via a second queue
- **Only one thread can access queue at a time…**
- …or the queue gets **corrupted**

Mutex

Mutex = "mutual exclusion" lock
Only 1 thread can get mutex at a time. Others wait till it is available.

Mutex

```
#include <semaphore.h>

sem_t mySemaphore;
sem_init(&mySemaphore, 0, 1); // start count, max allowed
…
sem_wait(&mySemaphore);
// do stuff
sem_post(&mySemaphore);
```


Support from OS and CPU, so even if two threads call sem_wait() at the same time:
1) the semaphore does not get corrupted,
2) exactly 1 thread gets the semaphore

It's like "passing a baton" back and forth. Only whoever has the baton can access some shared resource.

Mutex – what's a shared resource?

```
typedef struct {
    unsigned long long data[LEN];
    int head;
    int tail;
    int fullness; // how many pushed but not popped
} FifoULL;
FifoULL fifo;
```

Thread #1:	Thread #2:
`push(&fifo);`	`pop(&fifo);`

Threads' instructions run simultaneously and unpredictably. Thread#1 may change head before updating data. Then Thread#2 reads garbage from data. Thread #1 needs to complete all its actions on the shared queue (resource) before anyone else tries to use it.

Mutex – what's a shared resource?

```
push(FifoULL* ff, unsigned long long value) {
    sem_wait(&sem);
    // manage the Fifo
    sem_post(&sem);
}

pop(FifoULL* ff) {
    sem_wait(&sem);
    // manage the Fifo
    sem_post(&sem);
}
```


Could we write sem_wait() and sem_post() ourselves? NO. Then those methods would become risky shared-resource code. Need special support from CPU assembly language and operating system.

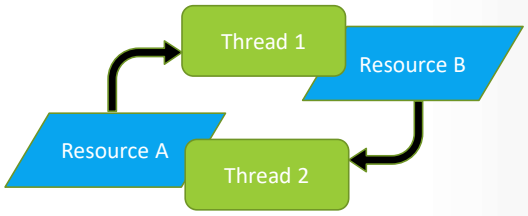
Debugging…

Multithreaded programs difficult to debug

- While we look at one thread, the other keeps running

Multithread programs with mutexes extra-difficult to debug

- Problems of synchronization, mutexes, and **deadlock**



Suppose Responsive is waiting to get top mutex and Backend is waiting to get bottom mutex…

REVIEW: testsem.c

Avoiding Deadlock

Can we use a one-way process flow? Parallel or Pipelined?

If not, can we minimize the number of shared resources and mutexes?

Hold a mutex for as little time as possible

- Just run a few lines of code while holding any mutex
- Never wait for one mutex while holding another!

Super-careful design and TESTING

You will need a detailed view of what each thread is doing, when it accesses shared resources, and possible conflicts between threads.

Code-along

10

Add third thread to testsem.c
Use ctime() to print time-of-day for different printouts?

Memory Allocator

Central allocator holds a "pool" of memory blocks. Other threads request and release them.

Why?

- Faster than malloc() and free()
- If often need the same large block size, reduce risk of memory fragmentation

Review mempool.c

Mutex fairness? If multiple threads are all waiting for a mutex, do they each get it sometimes?