ANNOUNCEMENTS

- Proxy Lab
  - No late/penalty dates
  - Absolute last turn-in: Thursday, April 25, 2013, 23:59

- Malloc Lab
  - Redemption is highly likely to be offered
    - More details in lecture
ROUTE FOR TODAY

- Threads and Mutual Exclusion
  - With examples
  - And maybe demos
- Teamwork
  - And Git
- Reader-Writer Locks
- Proxy (and You!)
- Rations, Extras, and Demos
  - Some (important) stuff not from recitation
**THREADS MODEL**

- stack
- stack
- stack
  - heap
  - data
  - code

- registers
- registers
- registers
Mutual Exclusion

- Mutexes
  - Only one thread in a section at a time
    - Called the “critical section”
    - Essentially, locking
  - Other threads must wait to enter
    - Mutual exclusion

- Semaphores
  - Fixed number of threads run the code at a time
    - Mutexes are a special case of semaphores (1 thread)
    - Examples to follow
      - Props to TA Tommy Klein
**Example: \( N^2 \)**

- Let's write a multithreaded program!
  - Spawns \( N \) threads
    - Each thread stores the current value of a global variable
    - Increments by 1, \( N \) times
    - Writes the result back into the global variable
  - When threads finish running, print the global
  - Result: \( N^2 \)
Example: $N^2$ (Without locks)

```c
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>
#define N 1000

static unsigned int global = 0;

// Have a thread add N to the global variable
void* threadFunc(void* vargp) {
    int i = 0;
    unsigned int locGlob = global;
    for (i = 0; i < N; i++)
        locGlob = locGlob + 1;
    global = locGlob;
    return NULL;
}

int main() {
    pthread_t tids[N];
    pthread_t tid;
    int i = 0;
    for (i = 0; i < N; i++) // Spawn n threads
        pthread_create(tids+i, NULL, threadFunc, NULL);
    for (i = 0; i < N; i++) // Wait for finish
        pthread_join(tids[i], NULL);
    printf("%u\n", global);
    return 0;
}
```
**EXPECTED/DESIRED BEHAVIOR**

- Thread 1: read global=0 into globLoc
- Thread 1: add 1000 to globLoc
- Thread 1: write global=globLoc=1000
- Thread 2: read global=1000...
Actual Behavior

- Thread 1: read global=0 into globLoc
- Thread 2: read global=0 into globLoc
- ....

Results will vary
Let’s see this in action!

Note: When at home, compile with the following

- `gcc filename.c -pthread -o outname`
- pthread support for the pthreads library
- `-o` specifies an output filename
ACHIEVING OUR GOALS

- Ensure each thread read/writes the correct value
  - **Key:** Synchronize access to the critical section
  - Use a mutex to serialize access to the global variable

- Special notes for this example
  - This will cause the code to run sequentially
  - Thread overhead will actually give worse performance compared to a sequential solution
EXAMPLE FIXED: $N^2$ (WITH LOCKS)

```c
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>
define N 1000

static unsigned int global = 0;
sem_t mutex;

//Have a thread add N to the global variable
void* threadFunc(void* vargp)
{
    int i = 0;
    sem_wait(&mutex); //Start critical code
    unsigned int locGlob = global;
    for (i = 0; i < N; i++)
        locGlob = locGlob + 1;
    global = locGlob;
    sem_post(&mutex); //End critical code
    return NULL;
}

int main()
{
    pthread_t tids[N];
    pthread_t tid;
    //Initialize semaphore to allow only 1 thread
    sem_init(&mutex,0,1);
    int i = 0;
    for (i = 0; i < N; i++) //Spawn n threads
        pthread_create(tids+i,NULL,threadFunc,NULL);
    for (i = 0; i < N; i++) //Wait for all finish
        pthread_join(tids[i], NULL);
    printf("%u\n",global);
    return 0;
}
```
Now with locks!
Multi-threaded Cache

- Why bother?
  - Sequential accesses are bottlenecks
    - We have parallel proxies!
- Multiple threads can read from a cache safely
  - Cache search and return blocks
  - No race when there are only reads
- But what about writes!?
  - Overwriting while another thread is reading?
  - Two threads writing to same cache block?
  - These are BAD THINGS™
**READER-WRITER LOCKS**

- **Key idea:** Cache can be read in parallel safely
  - If thread is writing, no other thread can read or write
    - Exclusive write access
  - If thread is reading, no other thread can write
    - Concurrent read access

- **Potential issues**
  - Write starvation
    - Reader threads block out writer threads
    - **Fix:** Prioritize writers
  - Read starvation
  - Aim for a fair policy
READER-WRITER LOCKS

How would you make a reader-writer lock?

- Luckily, you don't have to!
  - pthread_rwlock_* handles that for you
    - pthread_rwlock_t lock;
    - pthread_rwlock_init(&lock,NUL);
    - pthread_rwlock_rdlock(&lock);
    - pthread_rwlock_wrlock(&lock);
    - pthread_rwlock_unlock(&lock);
TEAMWORK

- Allowed **one** partner
  - Highly encouraged
    - Less work for us, less work for you
  - Grading is the same, solo or paired
  - Sign-up on Autolab
  - The staff can help partner match
    - Email the staff list: 15-213-staff@cs.cmu.edu

- Collaboration
  - Work division
    - “Proxy” and “caching” are independent tasks
  - Use Git for version control
    - Branch feature helps with work division
**GIT**

- **What is this “Git” thing?**
  - Version Control System (VCS)
    - Keeps history
    - Revision control
  - Widely used
    - And is on all campus machines

- **Repository options**
  - Make sure only you and your partner can access it!
  - GitHub
  - BitBucket
  - AFS
**Git Commands**

- `git clone`
- `git pull`
- `git add .`
- `git commit`
- `git push`

There is (was?) a StuCo on Git

- [98-174: Modern Version Control with Git](#)
**Git on Sharks/Unix**

- May run into errors pushing/pulling from AFS
  - Needs you to set up SSH keys
- [Generating SSH keys (from GitHub)](https://github.com/username/ssh-key)
WHAT YOUR PROXY SHOULD DO

- Access most sites
  - reddit, Vimeo, CNN, YouTube...
  - POST operations (sending data) will not work
    - Login boxes, comment boxes...
    - Only required to support GET requests
      - Feel free to do POST too

- Cache requests
  - Size limit
  - LRU eviction policy
  - Must allow for concurrent reads
  - Read the write-up
Proxy and You

- Previously: You write code for your use
- Now: You write code for a user

Your proxy must be robust
- Cannot crash for any reason
- Expect garbage inputs
  - Malformed web addresses, any requests...
- Never trust the user
  - Assume monkeys and cats

See: O.S. Boot Camp Slides (slide 3)
Cats on Keyboards
PROXY AND YOU

- Memory management
  - (Always) free what you malloc
  - Web servers/proxies expected to run “forever”
    - Memory leaks add up
  - Can’t (don’t want to) always have to restart
    - Includes: crashes, exceptions, memory leaks...
    - Prof. Koopman has some commentary on this
Proxy Lab

- Test extensively!
  - No autograded feedback
  - Browse sites with Firefox and your proxy
  - Try to break your proxy
  - Ask staff if you’re unsure about functionality
    - What should/shouldn't be working on your proxy

- Start early
  - Not as time-consuming as malloc
    - ... But teamwork is half the battle
    - ... As is parsing
  - Keep testing
    - Always find new ways to break your proxy
Proxy Rations

Materials we provide

- ./port_for_user.pl andrewID
  - Returns a port number for your use
- Tiny Web server
  - Basic example of a web server
- CS:APP source and header files
  - Use and modify as you see fit
- proxy.c
  - Put relevant code here
  - You are not limited to one file
- Makefile
  - Has relevant flags for pthread compilation
**Thread Safety, Reentrancy, & CSAPP**

- Do not blindly use the provided wrappers
- Wrappers may not have the desired behavior
  - Most are designed to exit on failure
    - Is that really what you want?
- Beware of thread-unsafe/ non-reentrant functions
  - `gethostbyname()`, `gethostbyaddr()`....
  - Pretty much anything that returns a pointer
    - “But why!?"
    - “Well.. Depends on where that pointer is coming from”
- The pthread/lpthread flags may help you
  - Useful stackoverflow post
Tiny/Proxy Demo

- Insert Godzilla here
- And using your proxy with Firefox
CREDITS ONLY

- 98-174: Modern Version Control with Git
- Generating SSH Keys (GitHub)
- 15-410 (OS) Boot Camp Slides
- Cat on a Keyboard
- stackoverflow on malloc