Recitation 11: ProxyLab

15-213 Fall 2009
November 16, 2009

Including Material from Previous Semesters' Recitations.
Overview

- Lecture Review
  - Unix Socket Interface
  - Echo Server

- ProxyLab
  - Pair Programming
  - Introduction to the Lab

- Threading
Lecture Review
Unix Socket Interface

- socket()
- bind()
- listen()
- accept()
- connect()
- send()
- recv()

Read the man pages to get a better understanding of how they're used and the errors that may occur.
Example Echo Server
ProxyLab
Pair Programming

- ProxyLab is the first lab on which you can work in pairs, but you can do it alone if you want.
- Finding a programming partner can be harder than finding a date, but here are two ways to start:
  - Ask someone before, during, or after a recitation or lecture.
  - Post a personal ad on the message board.

- **Use common etiquette:**
  - If someone thinks you'll be working with them, but you change your mind, let them know.
  - When working with someone, do your fair share of the work and be available to meet with your partner.

- Document your code so that your partner can understand it.
- Understand the whole program, even the parts your partner worked on.
Pair Programming Methods

- **You have two basic options:**
  - Work together programming on the same computer.
    - This makes your program better the first time through and helps you catch bugs, but may not be the most efficient method.
  - Code separately, especially by taking advantage of the modularity of your program. Ways to share code:
    - Email code back and forth.
    - Giving each other access to a part of your AFS space.
    - Use revision control software.
      - We recommend git ([http://git-scm.com/](http://git-scm.com/)).
ProxyLab Introduction

- Proxies are used for various reasons:
  - Firewalls
  - Filters
  - Removing identifying information from requests
  - Caching

- You will write a simple proxy that processes requests concurrently and caches web objects.
  - Requires knowledge of network programming, concurrency, and synchronization.
Proxy Example

1. I can has www.google.com?

2. I can has www.google.com?

3. Yes! Here it is 010101010101...

4. Yes! Here it is 010101010101.
Caching Proxy

- A proxy that remembers web objects it fetches for a user. So if a web object is requested again, the proxy server can immediately serve the web object, instead of fetching it again.

1. I can has www.google.com?
2. Hey! I remember that site!
3. Yes! Here it is 0101010101...
ProxyLab

- Appropriate and graceful error handling.
- Document design decisions
- Code organization
  - Break proxy into multiple functions
- Complete lab in three stages
  - Basic sequential proxy
  - Handling concurrent requests
  - Caching
- Understand what is robust about the rio package
  - Behavior of network sockets
- Each member of a group must understand everything in the group's program.
Testing and Grading your Proxy

- No autograder, but everyone in a pair must submit his or her lab to Autolab!
- When done, each group will meet with a TA and demo its lab.
- The TAs will grade labs by hand.
- Test your proxy by setting up your browser to use your proxy.
  - For Firefox, use FoxyProxy: [http://foxyproxy.mozdev.org](http://foxyproxy.mozdev.org)
Running and Testing your Proxy

[16:08] hpitelka@cobia:~/15213/proxylab$ hostname
cobia.ics.cs.cmu.edu

[16:08] hpitelka@cobia:~/15213/proxylab$ ./port_for_user.pl
hpitelka
hpitelka: 50206

[16:09] hpitelka@cobia:~/15213/proxylab$ ./proxy 50206

![FoxyProxy configuration settings](image_url)
Threading
Multi-Threaded process

Thread 1
- Stack 1

Thread 1 context:
- Data registers
- Condition codes
- SP-1
- PC-1

Thread 2
- Stack 2

Thread 2 context:
- Data registers
- Condition codes
- SP-2
- PC-2

Thread N
- Stack N

Thread N context:
- Data registers
- Condition codes
- SP-N
- PC-N

Shared resources:
- Kernel context:
  - VM structures
  - Descriptor table

Private Address Space
- shared libraries
- run-time heap
- writable data
- read-only data
- code
- 0
Posix Threads (Pthreads) Interface

- **Standard interface for ~60 functions**
  - Creating and reaping threads.
    - pthread_create
    - pthread_join
  - Determining your thread ID
    - pthread_self
  - Terminating threads
    - pthread_cancel
    - pthread_exit
  - Synchronizing access to shared variables
    - pthread_rwlock_init
    - pthread_mutex_\[un\]lock
    - pthread_cond_init
    - pthread_rwlock_\[wr\]dlock
Multi-threaded Hello World

/* hello.c - Pthreads "hello, world" program */

#include "csapp.h"

void *thread(void *vargp);

int main() {
    pthread_t tid;
    int i;
    for(i = 0; i < 42; ++i) {
        pthread_create(&tid, NULL, thread, NULL);
        pthread_join(tid, NULL);
    }
    exit(0);
}

/* thread routine */
void *thread(void *vargp) {
    printf("Hello, world!\n");
    return NULL;
}
Exiting a process and thread

- `pthread_exit()` only terminates the current thread, NOT the process

- `exit()` terminates ALL the threads in the process, i.e., the process itself
Joinable & Detached Threads

- **Joinable** thread can be reaped and killed by other threads
  - must be reaped (with pthread_join) to free memory resources.

- **Detached** thread cannot be reaped or killed by other threads
  - resources are automatically reaped on termination.

- Default state is joinable
  - use pthread_detach(pthread_self()) to make detached.
Review

- Lecture Review
  - Unix Socket Interface
  - Echo Server

- ProxyLab
  - Pair Programming
  - Introduction to the Lab

- Threading

- Next: Synchronization and Thread Safety
Questions?