Anita's Super Awesome Recitation slides

15/18-213: Introduction to Computer Systems

Threading and Proxy, 30 July 2013

Anita Zhang, Section M

Announcements

- Proxy Lab
  - No partners
  - Due next Wednesday, August 7, 2013, 11:59 PM
    - Cash in your late days
- Next week: Exam Review
  - If you want me/Michael to do specific problems or review specific topics, email the list

Route for Today

- Networking Functions “Detailed”
- Threads and Mutual Exclusion
  - With examples
  - And maybe demos
- Git
- Reader-Writer Locks
- Proxy (and You!)
- Rations, Extras, and Demos
  - Some (important) stuff not from recitation

Why These Helper Functions?

- Just raw `socket`, `bind`, `connect`, `listen` are complicated
  - Look in csapp.c… So many arguments and fields
  - To save you from the pain of figuring out how to wrestle these functions, we made these helpers
    - Combines multiple system calls into few friendly functions!

Servers: open_listenfd

- `int open_listenfd(int port)`
  - CSAPP wrapper
    - Contains proper error handling using uppercase functions
  - Performs 3 functions in one!
    - `socket`
    - `bind`
    - `listen`
  - Returns a file descriptor used for accept and to read from and write to a client.

Clients: open_clientfd

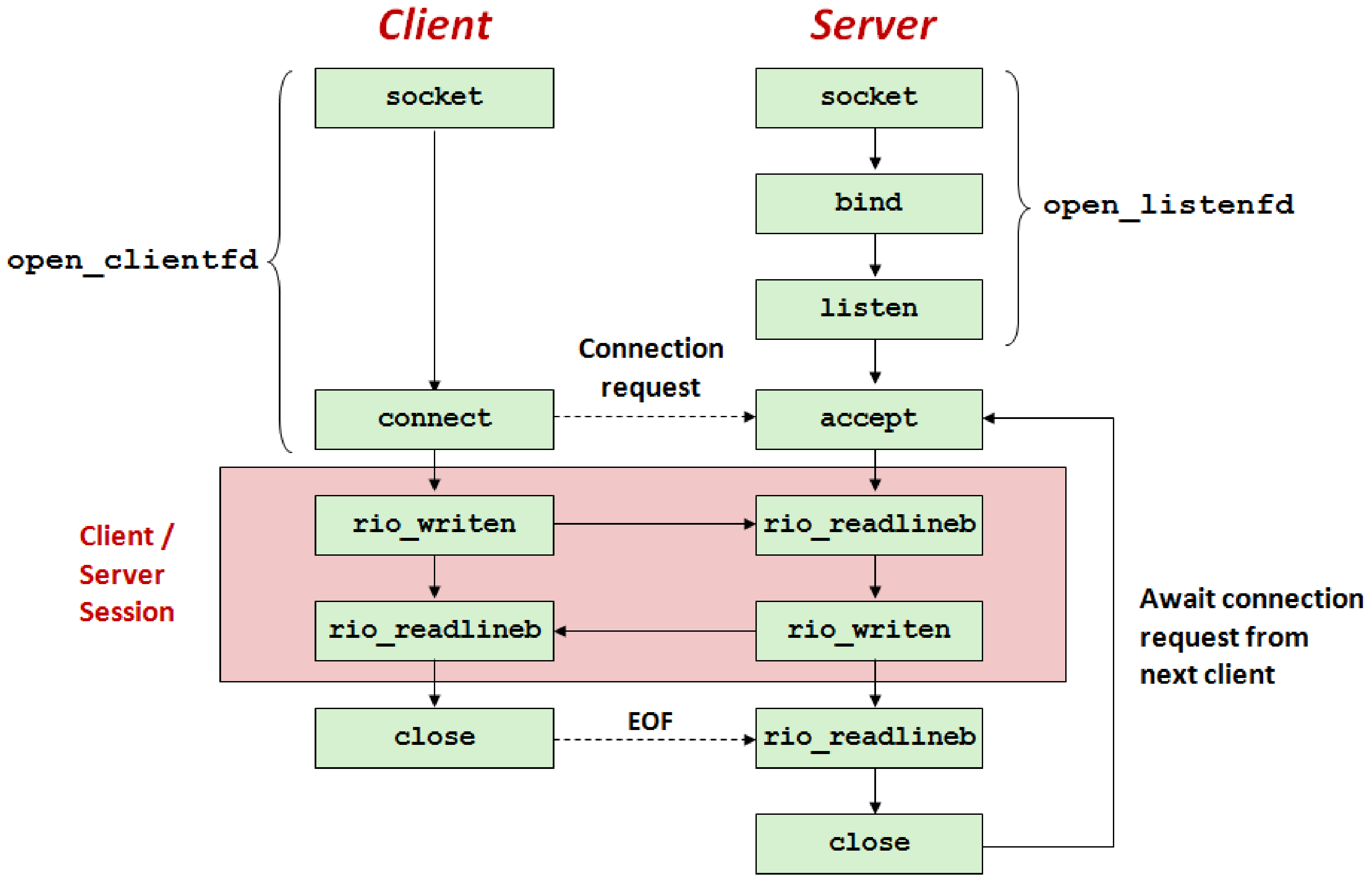
- `int open_clientfd(char *hostname, char *port)`
  - CSAPP wrapper
    - Contains proper error handling using uppercase functions
  - Performs 2 functions in one!
    - `socket`
    - `connect`
  - Note that the port is taken as a `char *`, not an `int`
    - Falls in line with the string parsing you'll be doing
  - Returns a file descriptor used to write headers and read back from a server.

GETADDRINFO

- Remember thread-safety and `gethostbyname`?

-
  ```
  int getaddrinfo(const char *node,
                  const char *service,
                  const struct addrinfo *hints,
                  struct addrinfo **res);
  ```
  - Reentrant! Thread-safe!
  - Fills out the provided struct addrinfo ** with addresses you can use when calling connect
  - Field definitions:
    - node – hostname
    - service – port
    - hints – Used to select from the *res* arg. NULL for this lab.
    - res – Linked list of socket address structures

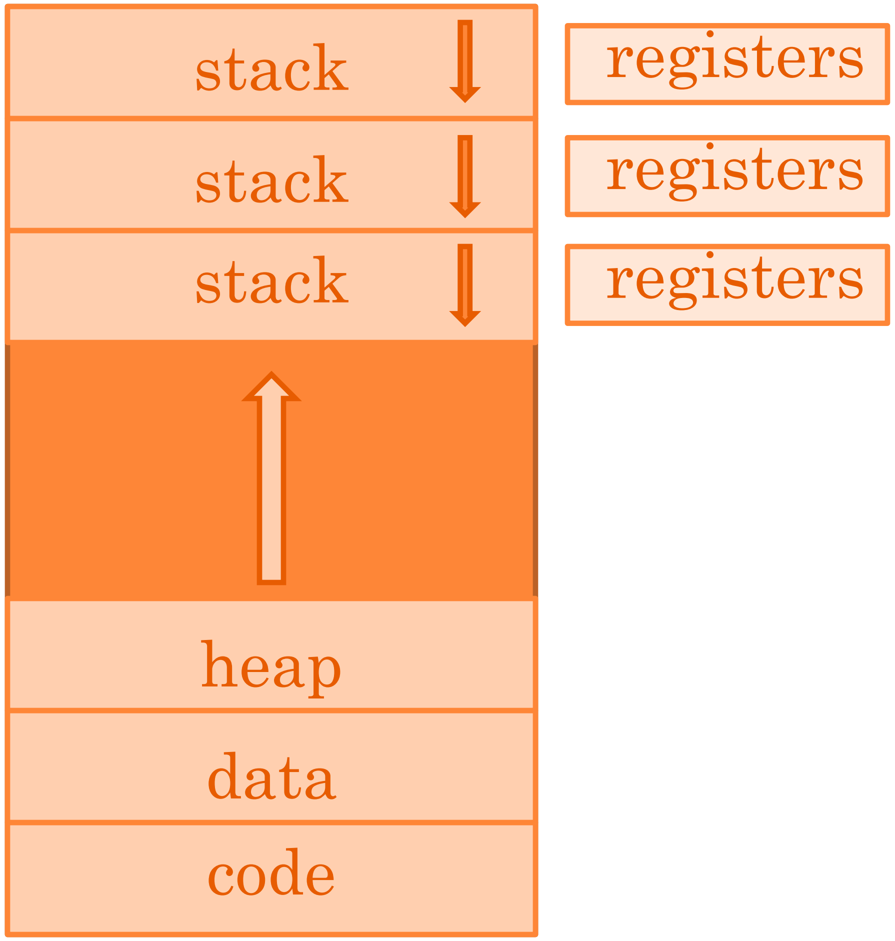
READ/WRITE FOR CLIENT/SERVER



GETADDRINFO – THE RES ARGUMENT

- The res field
  - User only needs to pass in a `struct addrinfo *`
  - `getaddrinfo` creates a dynamically allocated linked list so you only need to point to that list
  - Only fields from `addrinfo` we care about for `connect`
    - `ai_addr`
    - `ai_addrlen`
  - Iterate over all the elements until `connect` succeeds
    - StackOverflow can tell you why…
- Replacing `gethostbyname` with `getaddrinfo` only affects connect, not socket

Review: Threads Model



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 initialized to 1
    - Examples to follow

Example: N^2

- Example brought to you by Tommy Klein
- 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)

```
#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

Insert Demo Here

- 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;
}
```

```c
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;
}
```

Insert Other Demo Here

- 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 do I make a reader-writer lock?"
  - Luckily, you don't have to!
  - pthread_rwlock_* functions handle that for you
    - pthread_rwlock_t lock;
      - Defining a lock (example)
    - pthread_rwlock_init(&lock, NULL);
      - Initializes the lock with attributes. NULL is default.
    - pthread_rwlock_rdlock(&lock);
      - Reader locks a region
    - pthread_rwlock_wrlock(&lock);
      - Writer locks a region
    - pthread_rwlock_unlock(&lock);
      - Unlocks a lock

Doing Proxy

- Work division
  - Two main components of this lab
    - "Proxy"
      - Making connections and sending data back and forth
      - Threading is a small component that build on this
    - "Caching"
      - Collecting web objects and forwarding properly
- Use Git for version control
  - Somewhat less important without partners
  - Useful for keeping track of changes that might break your proxy

Git

- What is this "Git" thing?
  - Version Control System (VCS)
    - Keeps history
    - Revision control
  - Widely used
    - And is on all campus machines
- Repository options
  - GitHub
  - BitBucket
  - AFS

Git Basic Commands

- git clone
- git pull
- git add <file or option>
- 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)

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
  - Anything HTTPS or SSL related will not work
- 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.. Maybe now
  - Not as time-consuming as malloc
    - String parser in C is tedious and half the battle
  - 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
    - Update this when you start new files

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

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