Proxylab
and stuff

15-213: Introduction to Computer Systems
Recitation 13: November 19, 2012

Donald Huang (donaldh)
Section M
Topics

- Summary of malloclab
- News
- Sockets
- Threads
- Proxylab
Malloclab review

- Questions, comments, lessons learned?
- Starting simple and making improvements is a good way to write code
  - Nice because you can have something simple working relatively easily
  - You can test each new optimization alone
- When you write your code, write it so that it can be easily maintained/changed
  - “how can I write my explicit list so that I can change it to seglists later?”
Topics

- Summary of malloclab
- News
- Sockets
- Threads
- Proxylab
News

- Last day to turn in malloc is today (if using two late days and taking a late penalty)
- Proxylab was out last Friday
Topics

- Summary of malloclab
- News
- Sockets
- Threads
- Proxylab
Sockets

- To get a struct hostent for a domain name:
  - struct hostent * gethostbyname(const char *name);
    - not threadsafe, threadsafe version is gethostbyname_r

- What is a socket?
  - To an application, a socket is a file descriptor that lets the application read/write from/to the network
  - (all Unix I/O devices, including networks, are modeled as files)

- Clients and servers communicate with each other by reading from and writing to socket descriptors

- The main difference between regular file I/O and socket I/O is how the application “opens” the socket descriptors
Sockets API

- `int socket(int domain, int type, int protocol);`
- `int bind(int socket, const struct sockaddr *address, socklen_t address_len);`
- `int listen(int socket, int backlog);`
- `int accept(int socket, struct sockaddr *address, socklen_t *address_len);`
- `int connect(int socket, struct sockaddr *address, socklen_t address_len);`
- `int close(int fd);`
- `ssize_t read(int fd, void *buf, size_t nbyte);`
- `ssize_t write(int fd, void *buf, size_t nbyte);`
Sockets API

- int socket(int domain, int type, int protocol);
  - used by both clients and servers
  - int sock_fd = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
  - Create a file descriptor for network communication
  - One socket can be used for two-way communication
### Sockets API

- **int bind(int socket, const struct sockaddr *address, socklen_t address_len);**
  - used by servers
  - `struct sockaddr_in sockaddr;`
    - `memset(&sockaddr, 0, sizeof(sockaddr));`
    - `sockaddr.sin_family = AF_INET;`
    - `sockaddr.sin_addr.s_addr = INADDR_ANY;`
    - `sockaddr.sin_port = htons(listenPort)`
    - `err = bind(sock_fd, (struct sockaddr *) sockaddr, sizeof(sockaddr));`
  - `sock_fd`: file descriptor of socket
  - `my_addr`: address to bind to, and information about it, like the port
  - `addrlen`: size of addr struct
  - Associate a socket with an IP address and port number
Sockets API

- `int listen(int socket, int backlog);`
  - used by servers
  - err = listen(sock_fd, MAX_WAITING_CONNECTIONS);
  - socket: socket to listen on
  - backlog: maximum number of waiting connections

- `int accept(int socket, struct sockaddr *address, socklen_t *address_len);`
  - used by servers
  - struct sockaddr_in client_addr;
  - socklen_t my_addr_len = sizeof(client_addr);
  - client_fd = accept(listener_fd, &client_addr, &my_addr_len);
  - socket: socket to listen on
  - address: pointer to sockaddr struct to hold client information after accept returns
  - return: file descriptor
Sockets API

- `int connect(int socket, struct sockaddr *address, socklen_t address_len);`
  - used by clients
  - attempt to connect to the specified IP address and port described in `address`

- `int close(int fd);`
  - used by both clients and servers
  - (also used for file I/O)
  - `fd`: socket fd to close
Sockets API

- `ssize_t read(int fd, void *buf, size_t nbyte);`
  - used by both clients and servers
  - (also used for file I/O)
  - `fd`: (socket) fd to read from
  - `buf`: buffer to read into
  - `nbytes`: buf length

- `ssize_t write(int fd, void *buf, size_t nbyte);`
  - used by both clients and servers
  - (also used for file I/O)
  - `fd`: (socket) fd to write to
  - `buf`: buffer to write
  - `nbytes`: buf length
Overview of the Sockets Interface

**Client**
- `socket`
- `connect`

**Server**
- `socket`
- `bind`
- `listen`
- `accept`

**open_clientfd**
- `Connection request` from `connect` to `accept`

**Client / Server Session**
- `rio_writen` to `rio_readlineb`
- `rio_readlineb` to `close`

**Await connection request from next client**
- `rio_readlineb` to `rio_writen`
- `close`

- `open_listendifd`
Topics

- Summary of malloclab
- News
- Sockets
- Threads
- Proxylab
Threads

■ Similarities to processes
  ▪ each thread has its own logical control flow (its own registers, so its own eip and stuff)
  ▪ multiple threads can be in the middle of running at the same time, possibly on different cores
  ▪ the kernel decides when to context switch to and from a thread (or a thread can voluntarily give up its share of cpu time by calling sleep, pause, sigsuspend, or something similar)

■ Differences with processes
  ▪ threads share code and data; processes generally don’t
  ▪ threads are less expensive to make than processes (processes are about 2x more expensive to create and reap)
Threads

- Each thread has its own stack/registers, including stack pointer and program counter (imagine what would happen otherwise)
- Processes start out as having one thread, and they also have code, data, page directory/table, file descriptors, and some other things associated with them
Threads: pthreads interface

- Creating/reaping threads
  - pthread_create
  - pthread_join

- To get your thread ID
  - pthread_self

- Terminating threads
  - pthread_cancel
  - pthread_exit

- Synchronizing access to shared variables
  - pthread_mutex_init
  - pthread_mutex_[un]lock
  - pthread_rwlock_init
  - pthread_rwlock_[wr]dlock
A thread terminates *implicitly* when its top-level thread routine returns.

A thread terminates *explicitly* by calling `pthread_exit(NULL)`.

`pthread_exit(NULL)` only terminates the current thread, **NOT** the process.

`exit(0)` terminates **ALL** the threads in the process (meaning the whole process terminates).

`pthread_cancel(tid)` terminates the thread with id equal to `tid`.
Threads

- Joinable threads can be reaped and killed by other threads
  - must be reaped with pthread_join to free memory and resources
- Detached threads 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
Multithreaded Hello World

```c
/* 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;
}
```
Topics

- Summary of malloclab
- News
- Sockets
- Threads
- Proxylab
Proxylab: what a proxy is??

- intermediary between client and server
Proxylab

- write a web proxy
  - multi-threaded
  - caching
- should work for most sites (not https connections though)
  - cnn.com
  - google.com
  - youtube.com
  - (not facebook.com)
- Forwards requests from the client to the server
  - acts as a server to the client, but as a client to the server the client is asking you to read from
Proxylab: recommended progress

- implement a sequential proxy
  - this proxy will be very slow in loading webpages
- upgrade to multithreaded proxy
  - should be decently fast
- add caching
  - this involves some multithreading issues we’ll talk about solving next week

- You are not given any tests, so make sure you test your proxy well on your own!
Questions?
(sockets, proxylab, what a proxy is??)

(comes to office hours if you need help)