PA6: Distributed Image Renderer

Jed Liu

Department of Computer Science
Cornell University

CS 316 Section
27-30 November 2007
Ray Tracing 101

Point of view

Image

Light source

Viewing ray

Illumination

Visible point

Object in scene
Point of view

Image

Viewing ray

Visible point

Light source

Illumination

Object in scene

Embarrassingly parallel!
We provide ray-tracing servers.
You write a *multithreaded* client to use them to render an image.

In the process, you will learn about:
- Programming threads in Linux
- Thread-safe data structures
- Basic network programming
- Basic task scheduling
System Architecture

- We give you the servers and image assembler.
- You implement everything else.
We give you the servers and image assembler.
You implement everything else.
Don’t panic. We also give you lots of help.
Outline

- Pthreads library
- Networking
  - Sockets API
  - Marshalling & unmarshalling
- Tools to make life easier
- Suggestions for getting started
- Resources
Many C functions will be covered
Won’t cover all aspects – just the important parts
  ▶ All optional function parameters are glossed over
Up to you to figure out error handling
  ▶ Read man pages for this
POSIX Threads (aka Pthreads)

(POSIX: “Portable Operating System Interface”)

Creating a thread

```c
#include <pthread.h>

pthread_t thread_id;
void* f(void* args);
void* args = ...;
...

pthread_create(&thread_id, NULL, f, args);
```

- `pthread_exit()` – terminate current thread
- `pthread_join()` – wait until another thread terminates
example0.c
Thread Synchronization: Mutexes

 Creating mutexes

```c
pthread_mutex_t mutex;
pthread_mutex_init(&mutex, NULL);
```

 Using mutexes

```c
pthread_mutex_lock(&mutex);
... // Critical section
pthread_mutex_unlock(&mutex);
```

 Destroying mutexes

```c
pthread_mutex_destroy(&mutex);
```
example1.c
Creating condition variables
```c
pthread_cond_t cond_var;
pthread_cond_init(&cond_var, NULL);
```

Waiting for a condition
```c
pthread_cond_wait(&cond_var, &mutex);
```

Signalling a condition
```c
// Unblocks single thread.
pthread_cond_signal(&cond_var);
// Unblocks multiple threads.
pthread_cond_broadcast(&cond_var);
```

Destroying condition variables
```c
pthread_cond_destroy(&cond_var);
```
example2.c
## Networking 101: Protocol Stack

<table>
<thead>
<tr>
<th>Layer</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Layer</td>
<td>HTTP</td>
</tr>
<tr>
<td>Transport Layer</td>
<td>TCP</td>
</tr>
<tr>
<td>Network Layer</td>
<td>IP</td>
</tr>
<tr>
<td>Link Layer</td>
<td>Ethernet</td>
</tr>
<tr>
<td>Physical Layer</td>
<td>CAT6</td>
</tr>
</tbody>
</table>
Networking 101

What happens in your browser when you go to http://cuinfo.cornell.edu/?
Networking 101

What happens in your browser when you go to http://cuinfo.cornell.edu/?

1. Use DNS to resolve cuinfo.cornell.edu to 132.236.218.15
2. Establish TCP connection to 132.236.218.15 on port 80
3. Use HTTP protocol to fetch web page and related files

- Demo
Networking in PA6

- We will establish for you TCP connections to given hostname:port combinations.
- Your clients should implement a protocol that the server understands.
  - Will be specified in the write-up.

- Network programming in C done using sockets API
  - For TCP, exposes a stream interface
Socket I/O

```plaintext
int sock = ...;
char* buf = ...;
int amt_to_read = ...;
int amt_to_write = ...;

// Reading
int amt_read =
    read(sock, buf, amt_to_read);

// Writing
int amt_written =
    write(sock, buf, amt_to_write);
```
Problem: if nothing to be read, `read()` blocks until remote end sends more data

Usually okay if just have a single connection. What about multiple connections?

- Could be blocked reading from A when there is data ready to be processed from B.
Problem: if nothing to be read, `read()` blocks until remote end sends more data

Usually okay if just have a single connection. What about multiple connections?

- Could be blocked reading from A when there is data ready to be processed from B.

`select()`: Wait until data available from at least one out of a set of sockets
Using `pselect()` (I)

```c
int* socks_to_watch = ...;
int num_socks = ...;
int i, num_ready;

int nfds = 0;
fd_set fds;
    FD_ZERO(&fds); // Initialize fds
for (i = 0; i < num_socks; i++) {
    int sock = socks_to_watch[i];
    FD_SET(sock, &fds); // Add socket to set
    if (sock >= nfds) nfds = sock + 1;
}

num_ready =
pselect(nfds, &fds, NULL, NULL, NULL, NULL);
```
num_ready =
    pselect(nfds, &fds, NULL, NULL, NULL, NULL);

if (num_ready > 0) {
    // There is data to be read.
    for (i = 0; i < num_socks; i++) {
        int sock = socks_to_watch[i];
        if (FD_ISSET(sock, &fds)) {
            // Data available from sock
        }
    }
}
Now have single thread multiplexing reads across several sockets.

But what about writes?

- Need to unblock `pselect()` when there’s data available to be sent.

Use signals to interrupt `pselect()`

- Need to mask signals until `pselect()` called.
- Doing this right can be complicated.
- We set things up for you; up to you to handle `pselect()`’s EINTR error condition correctly.
Network data format doesn’t always match data layout in memory
  - e.g. `struct { char a; int b; }
  - Endianness may not match

Need to convert between memory & network representation
  - Marshalling: memory → network
  - Unmarshalling: network → memory
Always need to perform endianness conversion in program

TCP/IP standardizes on big endian

How to write portable code?

- Use htonl() and ntohl() to convert ints
- Also have htons() and ntohs() for shorts
- Write code once, works everywhere
Tools to Make Life Easier: netcat

- Dumps network data to screen
- Can act as server or client
- Also has tunnelling mode
  - Sits between server & client
- Installed as `nc` on CSUG machines
  - Type "nc -h" for help on using it
Tools to Make Life Easier: valgrind

- Analyzes memory usage
- Detects buffer overflows, memory leaks, double-frees, segfaults
Suggestions for Getting Started

- Start early!
- Tackle threading before networking
  - Implement thread-safe queues first
- Talk to netcat before talking to the servers
  - Make sure you’re sending well-formed messages before trying to read responses
- Start early!
- Leave work manager for last
  - Should be easy after everything else is working
  - In meantime, have manager send out single request to each server
- Start early and **ask us if you get stuck**
Resources

- man pages
  - man 2 <syscall-function>
  - man 3 <library-function>
  - man 3p <posix-library-function>
- Wikipedia has articles on many C functions, often with example code
- Googling for C function names often turns up informative pages with example code
- Office hours, cs316-1@cs.cornell.edu