Discussion 3:

A2 Prep, DNS/Caching Questions

By Joey Buiteweg

(some slide idea credits to Yiwen Zhang)

Learning Objectives

By the end of this discussion we will:

- Know how to use the select() syscall
- Be able to answer questions about DNS and Video streaming

But first!
Common issues I’ve seen

Communication Functions: recv()

Receive a message.

Similar to hearing something during the phone conversation.

```
ssize_t recv(int sockfd, const void * buf, size_t len,
             int flags);
// Example
ssize_t bytes_recvd = recv(sockfd, buffer, MSG_SIZE, 0);
// here bytes_recvd is not always == to MSG_SIZE

// But we also can do
ssize_t bytes_recvd = recv(sockfd, buffer, MSG_SIZE
                           MSG_WAITALL);
// here bytes_recvd is == MSG_SIZE, but we block until then
```

Communication Functions: recv()

Receive a message.

Similar to hearing something during the phone conversation.

```
ssize_t recv(int sockfd, const void * buf, size_t len,
             int flags);
// Example
ssize_t bytes_recvd = recv(sockfd, buffer, MSG_SIZE, 0);
// here bytes_recvd is not always == to MSG_SIZE

// But we also can do
ssize_t bytes_recvd = recv(sockfd, buffer, MSG_SIZE
                           MSG_WAITALL);
// here bytes_recvd is == MSG_SIZE, but we block until then
```

Some problematic server code

```
int client_sd; char buffer[1000];
ssize_t total = 0;
char FIN = 'F';
while (true){
    ssize_t recvd_bytes = recv(client_sd, buffer, 1000, 0);
    total += recvd_bytes;
    // Expecting a designation that client is done sending.
    if (buffer[0] == FIN){
        break;
    }
}
// What could go wrong here?
```

Is this any better?

```
int client_sd; char buffer[1000];
ssize_t total = 0;
char FIN = 'F';
while (true){
    ssize_t recvd_bytes = recv(client_sd, buffer, 1000, 0);
    total += recvd_bytes;
    // Expecting a designation that client is done sending.
    if (buffer[999] == FIN){
        break;
    }
}
// What could go wrong here?
```

Some problematic client code

```
char buffer[1000];
memset(buffer, 0, 1000);
while (!timeout){ // Assume timeout is managed for us
    ssize_t sent_bytes = send(server_sd, buffer, 1000, 0);
    total += recvd_bytes;
}
send_fin_message()
const char * ACK = “ACK”;
ssize_t bytes_recvd = recv(server_sd, buffer, 1000, 0);
if (strcmp(buffer, ACK) != 0){ // Expect buffer to have ACK
    perror(“Problem receiving ACK”);
    exit(EXIT_FAILURE);
}
// What could go wrong here?
```

Much better

```
char buffer[1000];
memset(buffer, 0, 1000);
while (!timeout){ // Assume timeout is managed for us
    ssize_t sent_bytes = send(server_sd, buffer, 1000, 0);
    total += recvd_bytes;
}
send_fin_message()
const char * ACK = “ACK”;
ssize_t bytes_recvd = recv(server_sd, buffer, 1000,
                                MSG_WAITALL);
// Expect buffer to have all of ACK message
if (strcmp(buffer, ACK) != 0){
    exit(EXIT_FAILURE)
}
```

std::string / cstrings (e.g char str[size])

Calls to recv and send DO NOT interpret the buffer you give as a cstring.

All calls to the socket syscalls see are an array of bytes.

It just so happens that we use the type “char” to mean the same thing as byte.

```
// Ideally we’d write
byte buffer[size];
// Or
uint8_t buffer[size];
// Instead of
char buffer[size];
```

It is your responsibility to ensure your buffer can be considered a valid cstring

std::array

Great type for holding buffer information.

Nice wrapper around c-style arrays

https://en.cppreference.com/w/cpp/container/array

A2 Prep

Administrivia

A1 is due very soon, 9/23/2020

- Although you all get a late day so really it’s 9/24/2020

Read the spec carefully for submission instructions, check piazza posts too

Administrivia

Assignment 2 spec coming out soon

You will implement a video streaming proxy server and a DNS server

Hard; the hardest assignment of the 4 (in my and most people's opinion).

~3.5 weeks to do it

A1 Reflection

Question:

- Can our A1 server code handle connections from multiple clients?
- How might we implement this?
  - Threads
  - select() !!

select(): I/O Multiplexing (the old fashioned way)

select()

Allow a program to monitor multiple file descriptors, waiting until one or more of the file descriptors becomes "ready" for some class of I/O operation.

```
#include <sys/select.h>
int select(int nfds, fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds, struct timeval *timeout);
// For A2, we only care about readfds, also no timeout
// so typically we invoke with
// select(FD_SETSIZE, &readfds, NULL, NULL, NULL);

void FD_SET(int fd, fd_set *set); // Add fd to the set
void FD_CLR(int fd, fd_set *set); // Remove fd from the set
// Return true if fd is in set, might not be after select()
int FD_ISSET(int fd, fd_set *set);
void FD_ZERO(fd_set *set); // Clear all entries from set
```

select() Pseudocode (part 1)

```
// setup master/server socket to call accept() from
master_socket = get_binded_socket();
client_arr = get_client_arr();
fd_set readfds;
while (true) {
    FD_ZERO(&readfs); // Clear leftover fds of previous loop
    // This allows us to check if we can accept() new client
    FD_SET(master_socket, &readfds);
    add_all_clients_to_set(readfds, client_arr);
    select(FD_SETSIZE, &readfds, NULL, NULL, NULL);
    // select() has modified readfds to contain only fds
    // who have actions available (e.g, accept() or recv())
    // pt.2 slide code goes here
}
```

select() Pseudocode (part 2)

```
if (FD_ISSET(master_socket, &readfds)){
    // add client by calling accept()
}

for (auto & clientfd: client_arr) {
    if (FD_ISSET(clientfd, &readfds)){
        // either client has disconnected via close()
        // or client has data available for us to recv()
    }
}
// loop back to top from previous slide
```

select(): Demo

Code available on course github page

poll(): I/O Multiplexing (the more modern way)

Information about poll() to review on your own

https://beej.us/guide/bgnet/html/#poll

YOU DO NOT HAVE TO USE poll() to finish A2

Just linking here for your own reference.

The code linked there is also on the course github page.

Lecture Based Questions

Q1 Web Caching

Why can web caching reduce the delay in receiving a requested object?

- Contents are cached closer to the clients.

Will it reduce the delay for all objects requested by a user?

- No, not every object will be found in the cache

What other advantages does caching provide? hint: think bandwidth

- Reduces congestion, frees up resources for eveyone

Q2 DNS

Suppose EECS has a one DNS server for all computers in the department.

(Probably not the case, just assume for this exercise)

How could you determine if an external web site was likely to be accessed from another computer in EECS a couple of seconds ago?

Perform two consecutive dig (or nslookup, or host, whatever you prefer) queries and compare the query time.

Q3 DNS Recursive

Suppose you are trying to access the page web.eecs.umich.edu/course/eecs489.

You are connected to your home WiFi with its own local DNS (from your ISP), and are **not** connected to MWireless/umich's network.

Give the order of name servers queried over time and their replies.

Assume:

- No prior caching, **Recursive** name resolution
- umich.edu and eecs.umich.edu are in zeparate zone
- eecs.umich.edu is authoritative for all hostnames ending in .eecs.umich.edu

Queries: local DNS -> root -> edu -> umich -> eecs

Replies: (eecs -> umich), (umich -> edu), (edu -> root), (root -> local DNS)

Real world answer to Q3

Just run

```
$ dig @8.8.8.8 +trace web.eecs.umich.edu ANY
```

Tells dig to contact Google’s public DNS resolver, and send us ANY resource records it has along the way

Q4 DNS Iterative

Suppose you are trying to access the page web.eecs.umich.edu/course/eecs489.

You are connected to your home WiFi with its own local DNS (from your ISP), and are **not** connected to MWireless/umich's network.

Give the order of name servers queried over time and their replies.

Assume:

- No prior caching, **Iterative** name resolution
- umich.edu and eecs.umich.edu are in zeparate zone
- eecs.umich.edu is authoritative for all hostnames ending in .eecs.umich.edu

Queries: local DNS -> root -> edu -> umich -> eecs

Replies: (root -> local), (edu -> local), (umich -> local), (eecs -> local)

Q5 DASH (Dynamic Adaptive Streaming over HTTP)

Consider system for which there are N video versions (at different rates and qualities) and N audio versions.
Suppose we want to allow the player to choose at any time any of the N video versions and any of the N audio versions.
Consider the following situations:

If we create files so that the audio is mixed in with the video, and the server sends only one media stream at a given time, how many files will the server need to store (each a different URL)? **N * N**

If the server instead sends the audio and video streams separately and has the client synchronize the streams, how many files will the server need to store?
N + N

Q6 HTTP Streaming

Consider a simple HTTP streaming model.
B is the size of the client's application buffer,
Q is the number of bits that must be buffered before the client application begins playout.
r is the video consumption rate.
Assume the server sends bits at a constant rate x whenever the client buffer is not full.
Also assume that $x < r$.

Describe the behavior of the video output: **Alternate between playout and freeze**

Suppose the buffer starts out empty.
How long will it be before the video can begin playout?
Q / x

Assume the current buffer size is **z** (> Q). How long will the playout last?
z / (r - x)