# Sockets

Apr. 13rd

# Client Flow

```
void simpleClient(string h,string p){
   Socket sock;
   sock.open(h,p);
   //read and write with sock.fd
   sock.close();
}
```

# Server Flow

```
void simpleServer(string p){
  vector<Socket> sockets(1);
  sockets[0].listen(p);
  while(1){
    waitOnEvents(sockets,-1);
    if(sockets[0].hasData()){
      sockets.push_back(sockets[0].accept());
    }
    for(int i=sockets.size()-1; i >= 1; —i){
      if(sockets[i].isBroken()){
        //client closed
        sockets.erase(sockets.begin() + i);
      }else if(sockets[i].hasData()){
        //need to read some data maybe write response
      }
    }
  }
}
```

# Server Flow

```
void simpleServer(string p){
  vector<Socket> sockets(1);
  sockets[0].listen(p);
  while(1){
    waitOnEvents(sockets,-1);
    if(sockets[0].hasData()){
      sockets.push_back(sockets[0].accept());
    }
    for(int i=sockets.size()-1; i >= 1; --i){
      if(sockets[i].isBroken()){
        //client closed
        sockets.erase(sockets.begin() + i);
      }else if(sockets[i].hasData()){
        //need to read some data maybe write response
      }
    }
  }
}
```

What is this?

What is this?

What is this?

What is this?

# Fool in the rain

```
bool waitOnEvents(const vector<Socket>& sockets, int timeout){
  vector<struct pollfd> helper;
  for(Socket s: sockets){
    struct pollfd pfd;
    pfd.fd = s.fd;
    pfd.events = POLLIN | POLLRDHUP | POLLHUP;
    helper.push_back(pfd);
  }
  return poll(helper.data(),helper.size(),timeout) > 0;
}
```

# Fool in the rain

-1 means wait until event
0 means check and return
otherwise wait max *timeout* seconds

```
bool waitOnEvents(const vector<Socket>& sockets, int timeout){
  vector<struct pollfd> helper;
  for(Socket s: sockets){
    struct pollfd pfd;
    pfd.fd = s.fd;
    pfd.events = POLLIN | POLLRDHUP | POLLHUP;
    helper.push_back(pfd);
  }
  return poll(helper.data(),helper.size(),timeout) > 0;
}
```

Kinds of events
to check

**Poll:**
check for events on
those file descriptors

# Hangup

```
bool Socket::isBroken(){
    struct pollfd pfd;
    pfd.fd = fd;
    pfd.events = POLLRDHUP | POLLHUP;
    poll(&pfd,1,0);
    return (pfd.revents & POLLRDHUP) || (pfd.revents & POLLHUP);
}
```

# Hello

```
bool Socket::hasData(){
        struct pollfd pfd;
        pfd.fd = fd;
        pfd.events = POLLIN;
        poll(&pfd,1,0);
        return pfd.revents & POLLIN;
}
```

# Example Time

https://tools.ietf.org/html/rfc958

https://en.wikipedia.org/wiki/Network_Time_Protocol

https://tf.nist.gov/tf-cgi/servers.cgi

https://tools.ietf.org/html/rfc4330

# Example Time

```
#include "Socket.h"

int main(){
    unsigned int seconds;
    Socket sock;
    sock.open("time.nist.gov", "123");

    sock.close();
    cout << seconds << " since 1900" << endl;
    return 0;
}
```

# Request

Network formats are often binary based not ascii
Request is 48 bytes long
Only first byte needs to be set

**Most Significant Bit**

| LI | | VN | | | MODE | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 |

**Least Significant Bit**

0x1b

Individual bytes are always in big endian format (most significant bit first)
x86 has least significant byte first

# Example Time

```
#include "Socket.h"

int main(){

   unsigned int seconds;
   unsigned char msg[48];
   memset(&msg,0,48);
   msg[0] = 0x1b;
   Socket sock;
   sock.open("time.nist.gov", "123");

   sock.close();
   cout << seconds << " since 1900" << endl;
   return 0;
}
```

**Will not respond multiple times within four seconds**

# Example Time

```
#include "Socket.h"

int main(){

   unsigned int seconds;
   unsigned char msg[48];
   memset(&msg,0,48);
   msg[0] = 0x1b;
   Socket sock;
   sock.open("time.nist.gov", "123");
   write(sock.fd,msg,48);

   sock.close();
   cout << seconds << " since 1900" << endl;
   return 0;
}
```

# Example Time

```
#include "Socket.h"

int main(){
   unsigned int seconds;
   unsigned char msg[48];
   memset(&msg,0,48);
   msg[0] = 0x1b;
   Socket sock;
   sock.open("time.nist.gov", "123");
   write(sock.fd,msg,48);
   read(sock.fd,msg,48);
   sock.close();
   seconds = ((unsigned int*)msg)[10];
   cout << seconds<< " since 1900" << endl;
   return 0;
}
```

# Example Time

```
#include "Socket.h"

int main(){
   unsigned int seconds;
   unsigned char msg[48];
   memset(&msg,0,48);
   msg[0] = 0x1b;
   Socket sock;
   sock.open("time.nist.gov", "123");
   write(sock.fd,msg,48);
   read(sock.fd,msg,48);
   sock.close();
   seconds = ((unsigned int*)msg)[10];
   cout << seconds<< " since 1900" << endl;
   return 0;
}
```

**Needs UDP not TCP!**

**Modify our Socket struct open**

# Example Time

```
#include "Socket.h"

int main(){
   unsigned int seconds;
   unsigned char msg[48];
   memset(&msg,0,48);
   msg[0] = 0x1b;
   Socket sock;
   sock.open("time.nist.gov", "123", false);
   write(sock.fd,msg,48);
   read(sock.fd,msg,48);
   sock.close();
   seconds = ((unsigned int*)msg)[10];
   cout << seconds<< " since 1900" << endl;
   return 0;
}
```

**Needs UDP not TCP!**

**Modify our Socket struct open**

# Example Time

```
#include "Socket.h"

int main(){
  unsigned int seconds;
  unsigned char msg[48];
  memset(&msg,0,48);
  msg[0] = 0x1b;
  Socket sock;
  sock.open("time.nist.gov", "123", false);
  write(sock.fd,msg,48);
  read(sock.fd,msg,48);
  sock.close();
  seconds = ((unsigned int*)msg)[10];
  cout << seconds<< " since 1900" << endl;
  return 0;
}
```

**Time isn't right**
**— big changes after a few seconds**

**Response is in Network Order**
**(Most significant byte first)**

# Example Time

```
#include "Socket.h"

int main(){
   unsigned int seconds;
   unsigned char msg[48];
   memset(&msg,0,48);
   msg[0] = 0x1b;
   Socket sock;
   sock.open("time.nist.gov", "123", false);
   write(sock.fd,msg,48);
   read(sock.fd,msg,48);
   sock.close();
   swap(msg[43],msg[40]);
   swap(msg[42],msg[41]);
   seconds = ((unsigned int*)msg)[10];
   cout << seconds<< " since 1900" << endl;
   return 0;
}
```