Writing Your Own Exceptions

Take advantage of C++'s exception system and write your own exceptions for your socket classes. Here's an example:

```
class SocketException : public std::exception
{
public:
  SocketException(std::string message="General Socket Exception!")
        : m_message(message) {}
  ~SocketException() throw() {}

  const char* what() const throw() { return m_message.c_str(); }

private:
  std::string m_message;
};
```

Writing Your Own Exceptions

You can throw that exception in your socket classes:

```
m_socket_fd = socket(m_family, SOCK_STREAM, 0);
if (m_socket_fd == -1)
  {
    throw SocketException("Socket creation failure!");
  }
```

Writing Your Own Exceptions

Which, in turn, can be caught somewhere else:

```
ServerSocket *serverSocket = 0;
try
  {
    serverSocket = new ServerSocket(nArgs.port);
  }
catch (std::exception &e)
  {
    std::cout << e.what() << std::endl;
    exit(EXIT_FAILURE);
  }
```

Another tip - catching signals

If you want to catch Ctrl-C in your code to help clean up your socket correctly, here's how.

- Use the signal function to register a callback function when specific signals are caught (see *man signal* for more info)

An example:

```
void signalHandler(int sig);

int main(int argc, char* argv[])
{
  // Setup a signal handler to catch the ^C (SIGINT) when we exit
  signal(SIGINT, signalHandler);

  while (1);
}

void signalHandler(int sig)
{
  std::cout << "Signal (sig=" << sig << ") Caught!" << std::endl;
  exit(EXIT_SUCCESS);
}
```

Another tip - catching signals

An example with socket's in mind:

```
void signalHandler(int sig);

int main(int argc, char* argv[])
{
  // Setup a signal handler to catch the ^C (SIGINT) when we exit
  signal(SIGINT, signalHandler);

  // setup sockets

  while (1)
  {
    ...
  };
}

void signalHandler(int sig)
{
  std::cout << "Signal (sig=" << sig << ") Caught!" << std::endl;

  // close sockets
  close(welcome_socket_fd);
  close(client_socket_fd);
  ...

  exit(EXIT_SUCCESS);
}
```

Become a real file transfer protocol!

Your first major programming assignment is to transfer files between two machines using the HTTP protocol. Ask yourself the following question:

- Can my code send and receive arbitrarily sized files?
- Can I transfer images (or binary data) with my code?

What are your answers?

Sending arbitrary sized data

First, what needs to be considered?

Sending arbitrary sized data

First, what needs to be considered?

- recv blocks!
- recv's buffer is limited

How do you get around this?

Recv buffer is limited!

Let's first focus on the recv call and the recv buffer you provide.

- What size(s) do you have for the recv buffer?

Recv buffer is limited!

Let's first focus on the recv call and the recv buffer you provide.

- What size(s) do you have for the recv buffer?
- Will that size work for a HTTP Response with a Ubuntu ISO image in it???

Recv buffer is limited!

Let's first focus on the recv call and the recv buffer you provide.

- What size(s) do you have for the recv buffer?
- Will that size work for a HTTP Response with a Ubuntu ISO image in it???

- Recall that recv tells you the number of bytes returned.
- So, keep calling recv until you've read everything.

Recv buffer is limited, cont'd!

What about the following idea?

```
int totalBytesReceived = 0;
do
  {
    int numBytesReceived = ::recv( m_socket_fd, buf, MSGSZ, 0 );
    totalBytesReceived += numBytesReceived;

    //
    // Concatenate message onto other received messages...
    // Pseudo-code!!!!
    msgBuffer += buf;

  } while (!done);
```

Will this work?

Recv buffer is limited, cont'd!

What about the following idea?

```
int totalBytesReceived = 0;
do
  {
    int numBytesReceived = ::recv( m_socket_fd, buf, MSGSZ, 0 );
    totalBytesReceived += numBytesReceived;

    //
    // Concatenate message onto other received messages...
    // Pseudo-code!!!!
    msgBuffer += buf;

  } while (!done);
```

Will this work? Maybe...

- Remember... recv blocks!
- What happens if there's no more data to be read???
- How do you know you're done?

Need to ask the OS if there is more data!

Problem

Because recv blocks, we need a way to ask the Operating System if there is more data that is internally buffered with the OS on this socket!

Luckily, we can use the select function to

- Query the status of file descriptors
- CHeck read, write, or other state

Important Socket Related Functions - select

- select - Allows a program to query the state of file descriptors.

```
int select(int nfds, fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds, struct timeval *timeout);
```

Important Socket Related Functions - select

- select - Allows a program to query the state of file descriptors.

```
int select(int nfds, fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds, struct timeval *timeout);
```

- Example Usage:

```
#include <sys/select.h>
...
  // check the socket to see if it has more input to read
fd_set rfds;
  FD_ZERO(&rfds);
  FD_SET(m_socket_fd, &rfds);

  // timer to wait is set to 0 to not wait
struct timeval tv;
  tv.tv_sec = 0;
  tv.tv_usec = 0;

  ...

  int selectRetVal = select(m_socket_fd+1, &rfds, NULL, NULL, &tv);
```

Bringing it together

```
int totalBytesReceived = 0;
do
  {
    int numBytesReceived = ::recv( m_socket_fd, buf, MSGSZ, 0 );
    totalBytesReceived += numBytesReceived;

    //
    // Concatenate message onto other received messages...
    // Pseudo-code!!!!
    msgBuffer += buf;

    // use select to see if there's more data...
    selectRetVal = select(m_socket_fd+1, &rfds, NULL, NULL, &tv);
  } while (???);
```

Bringing it together

```
int totalBytesReceived = 0;
do
  {
    int numBytesReceived = ::recv( m_socket_fd, buf, MSGSZ, 0 );
    totalBytesReceived += numBytesReceived;

    //
    // Concatenate message onto other received messages...
    // Pseudo-code!!!!
    msgBuffer += buf;

    // use select to see if there's more data...
    selectRetVal = select(m_socket_fd+1, &rfds, NULL, NULL, &tv);
  } while (numBytesReceived == MSGSZ && selectRetVal);
```

DNS - Domain Name System

Domain name system will be fully discussed on Wednesday.