CS4516: Medical Examiner Client/Server

Evan Frenn (ejfrenn@cs.wpi.edu)
Overview

**Objective:** Implement both a client and concurrent server with an overlay network emulating four layers of the TCP/IP stack.

- Application Layer: Medical Examiner functionality (Messages)
- Network Layer: Converts messages to packets; Handles sequencing (Packets)
- Data Link Layer: Go Back N sliding window; Error Detection (Frames)
- Physical Layer: Handles sockets and error creation
System Structure

Client
- Application Layer
- Network Layer
- Data Link Layer
- Physical Layer

Server
- Application Layer
- Network Layer
- Data Link Layer
- Physical Layer

TCP Connection
Application Layer

- Minimum of six client request types. **Must include:** Input of a photo
- Must include message to distinguish user type (authoritative vs. query only)
- It is optional to implement application layer separately or as part of the network layer
Network Layer

Packet Structure
- 2 Byte Seq #
- Message Payload

Note: The NL may pass packets to the DLL until the sliding window is full
Data Link Layer

Frame Structure
- Payload
- 2 Bytes Sequence #
- 2 Byte Error Detection
- 1 Byte End-of-Frame
- 1 Byte Frame Type

Sliding Window >= 4

ARQ
- Go-Back N
- ACK or NACK?
- Piggyback the ACK?

Payload Size <= 150 Bytes

pl_send(frame)
pl_rcv(frame)
Go-Back N ARQ

Timeout Occurs for frame 3!!
4 outstanding frames so go back 4

Out-of-sequence frames

ACKing next frame expected

Leon-Garcia & Widjaja: Communication Networks
Physical Layer

Forced Single Bit Error
- Every 6\textsuperscript{th} data frame
- Every 8\textsuperscript{th} ACK frame

recv(sockfd, data, len, ...)  
send(sockfd, data, ...)
Concurrent Server

• Server must be able to process multiple clients in parallel

• A few ways to achieve concurrency:
  - Use fork() – separate processes, no shared memory
  - Use threads (pthreads)
Concurrent Server (using fork)

```c
pid_t pid, id;
int listenfd, connfd;
/* 1. create a socket socket() */
if ((listenfd = socket(AF_INET, SOCK_STREAM, 0)) < 0 ){
    perror("Error creating socket");
    exit(1); }
/* 2. fill in sockaddr_in{ } with server's well-known port */
...
/* 3. bind socket to a sockaddr_in structure bind() */
bind (listenfd, ...);
/* 4. specify the backlog of incoming connection requests listen() */
listen (listenfd, 5);
while(1){
    connfd = accept(listenfd, ... );
    if(( pid = fork()) == 0){
        close(listenfd); /* child closes listening socket */
        doit(connfd); /* process the request. this is where the work is done. */
        close(connfd); /* done with this client */
        exit(0);
    }
    close(listenfd); /* parent closes the socket */
    exit(0);
}
```
Data Link Layer: dl_send(packet,...)

1. Split packet into frame payloads
2. Create frame (set params then EC)
3. Start a timer
4. pl_send(frame)
5. Go-Back N
6. Timeout Handler
Frame Structure

1. Compute Seq Number, Frame Type and End-Of-Frame (EOF) bytes

2. Error-Detection (ED) bytes (XOR on Seq + FT + EOF + Data)

EOF: End of Frame
FT: Frame Type
ED: Error Detection
Seq: Sequence Num
Data Link Layer: Frame Reception

1. Compute Error Detection value
2. ED Ok? (Yes/No)
   - Yes: ACK/FRAME
     - ACK: Update window/timer
       - No: Drop Packet
     - Frame: SEQ Right? (Yes/No)
       - Yes: pl_send(ACK frame)
9. Reassemble Packet (store for dl_recv())
   - No: Return
Timers

• Timers will be used by the data link layer to detect frame loss.
  ➢ DLL sets a timer when transmitting a frame
  ➢ When the timer expires, DDL handles retransmit of send window up to lost frame
  ➢ With Go Back N, it is possible to use a single timer and keep track of time intervals between transmission of frames.

• Two options for timers:
  ➢ Select
  ➢ Signals and Timers
Example Using select()

- Can be used to both send/recv at same time and/or as a timer for the data link layer

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

```c
int main(void) {
    fd_set rfds;
    struct timeval tv;
    int retval;
    /* Watch stdin (fd 0) to see when it has input. */
    FD_ZERO(&rfds);
    FD_SET(0, &rfds);
    /* Wait up to five seconds. */
    tv.tv_sec = 5;
    tv.tv_usec = 0;
    retval = select(1, &rfds, NULL, NULL, &tv);
    if (retval == -1)
        perror("select()");
    else if (retval)
        printf("Data is available now.\n");
    else // retval == 0 here
        printf("No data within five seconds.\n");
    exit(EXIT_SUCCESS);
}
```
Signal Example

```c
#include <signal.h>
#include <time.h>

timer_t timer_id;

void timeout(int signal_number){
    printf("\n SIGNUM: %d\n", signal_number);
    exit(0);
}

void start_timer(){
    struct itimerspec time_val;
    signal (SIGALRM, timeout);
    timer_create(
        CLOCK_REALTIME,
        NULL, &timer_id);

    /* set timeout to 1 second */
    time_val.it_value.tv_sec = 1;
    time_val.it_value.tv_nsec = 0;
    time_val.it_interval.tv_sec = 0;
    time_val.it_interval.tv_nsec = 0;
    timer_settime(timer_id, 0, &time_val, NULL);
}

main(){
    start_timer();
    while(1);
}
```
Send/Recv “concurrently”

• One option mention earlier is to use select()
• Another option is to use fcntl() to have the socket not block during recv()
• int fcntl(int s, int cmd, long arg);
  Example:

    #include <sys/unistd.h>
    #include <sys/fcntl.h>

    fcntl(socketFD, F_SETFL, O_NONBLOCK);
Questions?