Disaster Victim Location
Database Server

Hao Wan
hale@wpi.edu
01/22/2013

Modified based on CS4516 D11 slides
Description

- **Objective:**

  To implement a simple concurrent server that has four emulated network protocol stacks.
  - Application layer: Messages
  - Network layer: Messages $\leftrightarrow$ Packets
  - Datalink layer: Packets $\leftrightarrow$ Frames and Selective Repeat Sliding Window protocol
  - Physical layer: TCP connection
You can either use multiprocesses (fork()) or multithreading (pthread).

You need to implement concurrent access to the database (lock).
System Framework

Client

APP Layer

NW Layer

DLL

PHL

Server

APP Layer

NW Layer

DLL

PHL

TCP Connection

Four Layer Stack
How the System Works: Layer by Layer

Application Layer

Client Requests:
- Application depended
- Messages

Server Responses:
- Application depended

Client Requests:
- nwl_send (… msg …)
- nwl_recv (… msg …)

- At least 6 operations: both uploading and downloading a photo of a person, and dealing with the two types of users
- Message that specifies the client type (FEMA-authorized or query client)
- Ability to query all victims in a specific location.
How the System Works: Layer by Layer

Network Layer

Client

Message

NWL

\( n_{\text{packets}} \)

End of Message

Can be an special packet, OR a special position in each packet, eg. 1\text{st} byte

dll_send (… pkt …)

dll_recv (… pkt …)

Server

Message

NWL

\( n_{\text{packets}} \)

Tasks for NWL

Disassemble and assemble packets from Msg.

No ACK in this layer

\( \leq 200 \) Bytes of message payload

2 Bytes sequence no. for packets

Note: The network layer will send packets until blocked by the Data Link Layer. But HOW?
How the System Works: Layer by Layer
DataLink Layer

Client

Server

- phl_send (… frm …)
- phl_recv (… frm …)

DLL

1 Byte End of Packet
2 Bytes Error Detection
2 Bytes SEQ#

Maximum packet size 1-120 bytes
Sliding window size ≥4

n_frames

Frames

ACK or NAK?
Piggyback?

Frames

Selective Repeat

packet

n_frames

packet
Selective Repeat

Errors could happen in both sides!
How the System Works: Layer by Layer

Physical Layer

- **Client**
  - frame
  - PHL
  - read (… data …)
  - write (… data …)
- **Server**
  - frame
  - PHL

- **Force Single Bit Errors**
  - Data: 7th frame
  - Ack: 11th frame
  - Server and clients

TCP Connection (Socket)
How the Functions Work: Layer by Layer

client APP

User input

Pick a command

q cmd?

Yes

Build Msg

nwl_send(…msg…)

nwl_recv(…ack…)

No

server child process APP

fork()

nwl_recv(……)

q cmd?

Yes

Build Msg

nwl_send(…msg…)

What to do if “Yes”?
How the Functions Work: Layer by Layer

\texttt{nwl\_send (\ldots \textit{msg} \ldots)} \quad \texttt{nwl\_recv (\ldots \textit{msg} \ldots)}

- Split msg into pkts
- Pick a pkt
- Last pkt?
- Set EOM
- dll\_send(\ldots pkt\ldots)

- dll\_recv (\ldots pkt \ldots)
- Last pkt?
- Reassemble pkts into msg
- Return msg to APP

Note: you need have a mechanism to decide the last packet in a message (EOM). The diagram here offers only a reference.
How the Functions Work: Layer by Layer

dll_send (… pkt … )

Split a packet into payloads

Create a new frame

Start a Timer

Send a frame to PHL

Wait for receiving a ACK frame

Retransmit frames if timeout or error ACK frame!

Receive a ACK frame correctly, then continue ...

Sliding window size = 1

phl_send (…)

phl_recv (…)

How the Functions Work: Layer by Layer

dll_recv (… pkt … )

- Receive a frame from PHL
- Compute ED byte and check error
- Drop if error detected
- Drop if duplicate, else send ACK
- Reassemble the packet
- If EOP, forward the packet to NWL

phl_recv (…)

phl_send (…)

Question: When is the correct time to send NAK or ACK?
Not after ED drop, but on receiving next frame or dup frame.
Debugging output

- Output that helps debugging the program
- Can be easily turned on/off by a macro
- The following statistics must be calculated and reported:
  - The total number of data frames transmitted successfully
  - The total number of data frames received successfully
  - The total number of data frames received with errors
  - The total number of ACK’s transmitted successfully
  - The total number of ACK’s received successfully
  - The total number of ACK’s received with errors
  - The total number of duplicate frames received.
Project Tips-1

- Sliding Window Protocol: Selective repeat (N>=4)
  - Try to implement windows size 1 first
  - Then implement N (multiple timers)
- Follow the example in the book (protocol 6)
- How to terminate client process:
  - When the client gets the response to the quit message
  - A “clean” way to terminate the server child process/thread? Use wait()/pthread_join()!
• Simulate multiple timer in software
  – Approach I
    • Using link list or array
    • pp.223 of Tanenbaum handout
    • Need signal()
  – Approach II
    • Using link list or array
    • Update the *struct timeval* for next select() call
Project Tip3

• How could the NWL Keep sending packets until blocked by the Data Link Layer?

Our suggestion is that you could use pipe to implement it: NWL keeps writing packets to the pipe until the pipe is full.

• A simple code of pipe could be found at http://web.umr.edu/~ercal/284/PipeExamples/Examples.html

• Pipe is more like a socket between local processes.
Concurrent TCP Server Example (fork)

```c
pid_t pid;
int listenfd, connfd;

/* 1. create a socket socket() */
if ((listenfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    err_quit("build server socket error\n", -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, LISTENQ);
while (1) {
    connfd = accept(listenfd, ...);  /* probably blocks */
    if(( pid = fork()) == 0) {
        close(listenfd);  /* child closes listening socket */
        doit(connfd);    /* process the request */
        close(connfd);   /* done with this client */
        exit(0);
    }
    close(connfd);    /* parent closes connected socket */
}
```
void main(void)
{
    fd_set rfds;
    struct timeval tv;
    int retval;

    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);
    /* Don't rely on the value of tv now! */

    if (retval == -1)
        perror("select()");
    else if (retval)
        printf("Data is available now.\n");
    /* FD_ISSET(0, &rfds) will be true. */
    else // retval == 0 here
        printf("No data within five seconds.\n");
    exit(EXIT_SUCCESS);
}
Questions?