Global Employee Location Server

Jeff Zhou
jeffz@cs.wpi.edu
3/19/2011

Modified based on CS4514 B05 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.
System Framework

Four Layer stacks
How the System Works: Layer by Layer

Application Layer

Client Requests:
- Application depended
- At least 5 operations, at least one long operation in each direction that will easily test your sliding window
- Message that specifies the client type (employee or HR)
- Input and query a photo, select all users in a country

Server Responses:
- Application depended

Client \( i \) \hspace{1cm} \rightarrow \hspace{1cm} \text{Messages} \hspace{1cm} \rightarrow \hspace{1cm} \text{Server} \hspace{1cm} \rightarrow \hspace{1cm} \text{Child Process} \ i

\hspace{1cm} \downarrow \hspace{1cm} \text{APP} \hspace{1cm} \downarrow \hspace{1cm} \text{nwl\_send (… msg …)} \hspace{1cm} \uparrow \hspace{1cm} \text{APP} \hspace{1cm} \uparrow \hspace{1cm} \text{nwl\_recv (… msg …)}
How the System Works: Layer by Layer

Network Layer

Client

Message

<table>
<thead>
<tr>
<th>NWL</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n _packets )</td>
</tr>
</tbody>
</table>

End of Message
Can be an special packet, OR a special position in each packet, eg. 1st byte

dll\_send (… pkt …)
dll\_recv (… pkt …)

Server

Message

<table>
<thead>
<tr>
<th>NWL</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n _packets )</td>
</tr>
</tbody>
</table>

Tasks for NWL
Disassemble and assemble packets from Msg.
No ACK in this layer
Two 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

- packet
- \( n_{frames} \)
- 1 Byte End of Packet
- 2 Bytes Error Detection
- 2 Bytes SEQ#

Server

- packet
- \( n_{frames} \)
- phl_send (… frm …)
- phl_recv (… frm …)

- Frames
- Selective Repeat
- ACK or NAK?
- Piggyback?

Maximum packet size 128 bytes

Sliding window size \( \geq 4 \)
Selective Repeat

```
fr 0  fr 1  fr 2  fr 3  fr 4  fr 5  fr 6  fr 7  fr 8  fr 9  fr 10 fr 11 fr 12
A A A A A A A A A A A A
K K K K K K K K K K K K

error
```
How the System Works: Layer by Layer

Physical Layer

TCP Connection

Client

Server

- Force Single Bit Errors
  - Data: 5\textsuperscript{th} frame
  - Ack: 7\textsuperscript{th} frame
  - Server and clients

Frame

PHL

read (… data …)

write (… data …)
How the Functions Work: Layer by Layer

client APP

User input

Pick a command

q cmd?

No

Build Msg

nwl_send(…msg…)

nwl_recv(…ack…)

Yes

server child process

APP

fork()

nwl_recv(……)

q cmd?

No

Build Msg

nwl_send(…msg…)

Yes
How the Functions Work: Layer by Layer

\texttt{nwl\_send (… msg …)} \quad \texttt{nwl\_recv (… msg …)}

- \texttt{nwl\_send (… msg …)}
  - Split msg into pkts
  - Pick a pkt
  - Last pkt?
    - No: \texttt{dll\_send (… pkt …)}
    - Yes: Set EOM

- \texttt{dll\_recv (… pkt …)}
  - Last pkt?
    - No
    - Yes: Reassemble pkts into msg
      - Set EOM
      - \texttt{dll\_send (… pkt …)}
  - 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 … )

1. Split a packet into payloads
2. Create a new frame
3. Start a Timer
4. Send a frame to PHL
5. Wait for receiving a ACK frame
6. Retransmit frames if timeout or error ACK frame!
7. Receive a ACK frame correctly, then continue ...

Sliding window size = 1

phl_send (…)

phl_recv (…)

Layer
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 on textbook()
  - 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](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 */
}
```
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);
    /* 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?