Concurrent Server Using Selective Repeat Data Link Layer

Mingzhe Li
lmz@cs.wpi.edu
12/05/2005
Description

• Objective:
  To implement a simple concurrent server and clients having four emulated network protocol stacks.
  - Application layer: Customized Applications
  - Network layer: Message $\rightarrow$ Packet (send&recv)
  - Datalink layer: Packet $\leftrightarrow$ Frame and Selective Repeat sliding window protocol
  - Physical layer: TCP connection.

• To get full credit, you need a sending and receiving windows size $\geq 4$.

• Your programs should compile and work on any host of ccc.WPI.EDU.
You can either use multiprocesses (fork()) or multithreading (pthread).
You need to implement concurrent access to the database.
System Framework

Client

- APP Layer
- NW Layer
- DLL
- PHL

Server

- APP Layer
- NW Layer
- DLL
- PHL

TCP Connection

Four Layer stacks
Selective Repeat

A

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

B

A
A
C
C
K
K
1
2

N
A
A
C
K
K
K
K
K
K
1
2
2
2
7
8
9
0
1
2

error

A
A
A
A
A
A
A
A
A

C
C
C
C
C
C
C
C
C

K
K
K
K
K
K
K
K
K

1
1
1
1

WPI
How the System Works: Layer by Layer

Application Layer

Client \( i \) \( \rightarrow \) Server

Messages

\( \text{APP} \)

\( \text{APP} \)

Client Requests:

Application depended

\( \text{nwl\_send (… msg …)} \)

\( \text{nwl\_recv (… msg …)} \)

Server Responses:

Application depended

\( \text{nwl\_send (… msg …)} \)

\( \text{nwl\_recv (… msg …)} \)

At least 5 operations, there is at least one long operation in each direction that will easily test your sliding window
How the System Works: Layer by Layer

Network Layer

Client

Message

NWL

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

Server

Message

NWL

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

dll_send (... pkt ...)
dll_recv (... pkt ...)

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

Note: The max_size of a packet is 256 bytes, 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

ACL or NAK?

Piggyback?

packet

DLL

packet

DLL

\(n_{\text{frames}}\)

packet

\(n_{\text{frames}}\)

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

phl_send (… frm …)

phl_recv (… frm …)

Selective Repeat

Note: The max_size of a frame payload is 100 bytes

Sliding window size \(\geq 4\)
How the System Works: Layer by Layer

Client

- frame
- PHL
- read (… data …)
- write (… data …)
- TCP Connection

Server

- frame
- PHL
- Identify client when start
- Force Single Bit Errors
  - Data: 8th frame
  - Ack: 6th frame
  - Server and clients
How the Functions Work: Layer by Layer

client APP

User input

Pick a command

q cmd?

No

Build Msg

nwlc_send(…msg…)

nwlc_recv(…ack…)

Yes

server child process

APP

fork()

nwlc_recv(……)

q cmd?

No

Build Msg

nwlc_send(…msg…)

Yes
How the Functions Work: Layer by Layer

\[
\text{nwl\_send (… msg …)} \quad \text{nwl\_recv (… msg …)}
\]

Split msg into pkts

Pick a pkt

Last pkt?

Yes

Change EOM

 dll\_send(… pkt …)

No

 dll\_recv (… pkt …)

Last pkt?

No

Reassemble pkts into msg

Yes

Return msg to APP

Note: you need have a mechanism to decide the last packet in a message (EOM).
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 \textit{if timeout or error ACK frame!}

Receive a ACK frame correctly, then continue ...

\textbf{Sliding window size = 1}

\textbf{phl_send (...)}

\textbf{phl_recv (...)}
How the Functions Work: Layer by Layer

dll_recv (… pkt … )

1. Receive a frame from PHL
2. Compute ED byte and check error
3. Drop if error detected
4. Drop if duplicate, else send ACK
5. Reassemble the packet
6. 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.
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? Use wait()!
• 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

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

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 */
}
Questions?