CMPUT 379

Lab 06

Assignment (2.a)

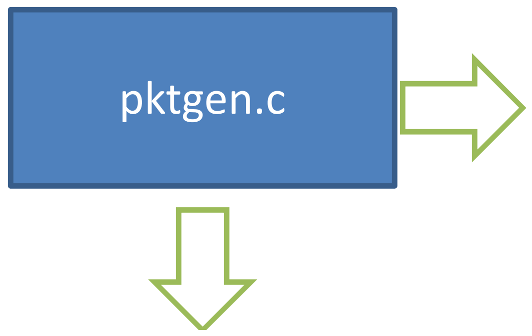
- **Packet generator program**
- **Router program**

router.c
pktgen.c

Packet generator program

<u>Command:</u>

<port number to connect to router> <packets file path>



IP Packet

```
1, 192.168.192.4, 192.224.0.7, 4, "Hello"
```

Packets file

```
NetA to NetB: 0
NetA to NetC: 0
NetB to NetA: 0
NetB to NetC: 0
NetC to NetA: 0
NetC to NetB: 0
Invalid Destination: 0
```

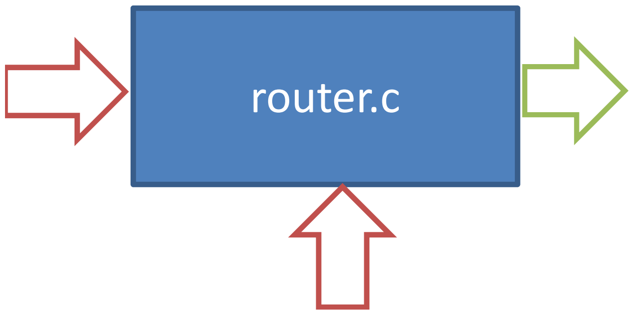
Router program

Command:

<port number to listen to> <routing table file path> <statistics file path>

IP Packet

```
1, 192.168.192.4, 192.224.0.7, 4, “Hello”
```



Statistic file

```
expired packets: 0
unroutable packets: 0
delivered direct: 0
router B: 0
router C: 0
```

Routing table

```
192.168.192.0   18    RouterB
192.168.128.0   17    0
192.224.0.0     16    RouterC
```

Router program

1. Decrement the TTL field of the packet
2. Figure out which of the entries in the routing table (if any) match the packet
    - extract the first <net-prefix-length> number of bits from the destination address.
3. If the value of the nexthop field for the entry is 0:
    - Update the "delivered direct" counter
4. If the value of the nexthop field for the entry is RouterB or RouterC
    - update the counter corresponding to the packets forwarded to that router
5. Update statistic file

Assignment (2.b)

Server Command:

<UDP port> <documents directory> <log file>

Client Command:

<server IP> <server UDP port> <file to download>

file_server.c

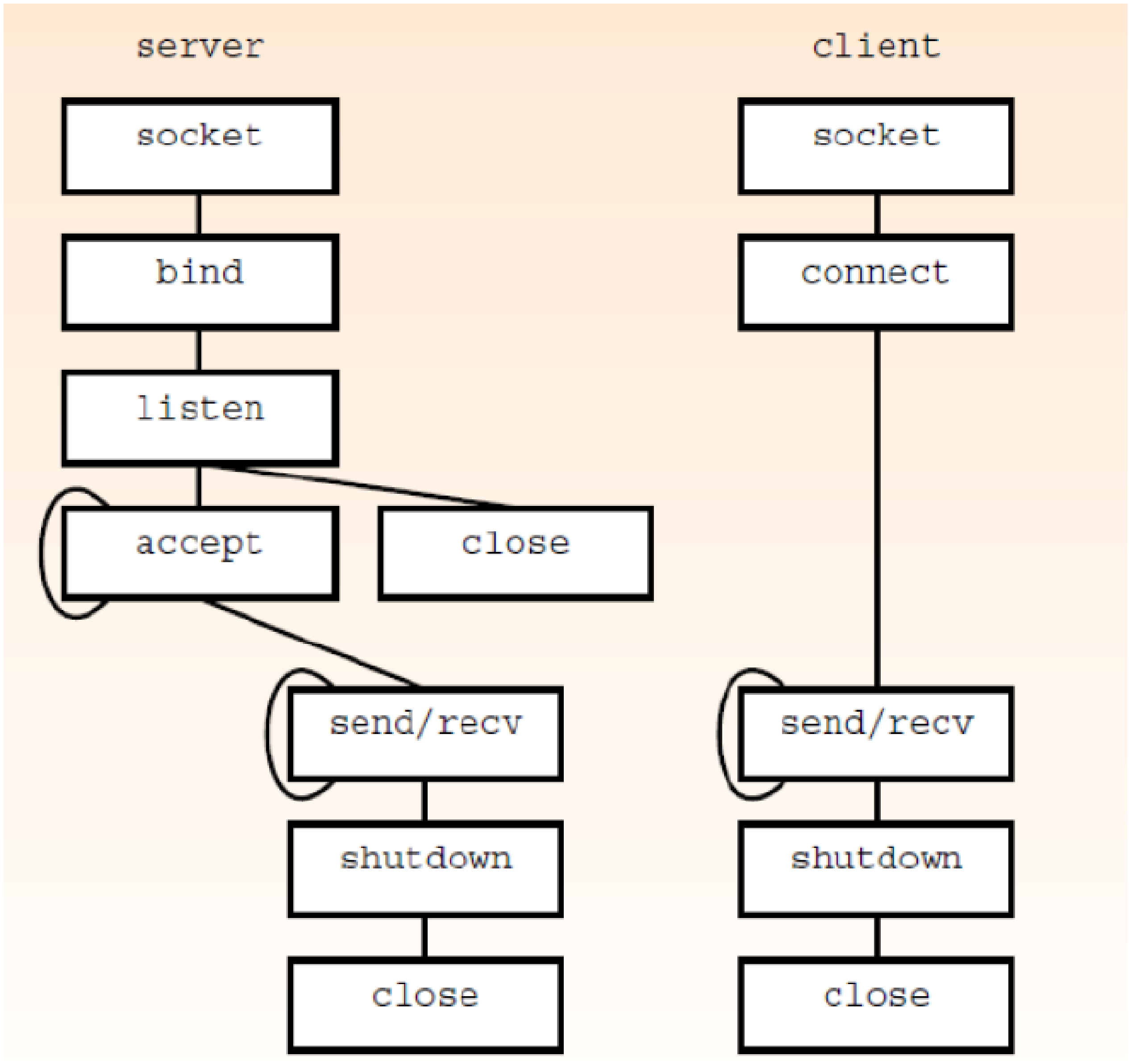
file_client.c

Server program

- The server must *daemonize* on startup.
- If the supplied logfile does not exist, the server should create it.
- **file_server** must be implemented using processes and fork().
  - The server accepts a request from a client.
  - Splits the intended file into chunks of 1KB and sends each chunk as a separate message.
  - If the file has a size that is a multiple 1KB, the server should send an additional message containing only "$" after sending all of the chunks.

Client program

- Send request to server.
- Accepts chunks of messages from the server.
  - If a chunk contains less than 1KB of data or only the message “$”, the desired file transmission is completed.
  - If 5 seconds have been passed since the last chunk received, the client assumes that the transmission had been aborted, and the client prints the error message on the screen.


server
client
socket
bind
listen
accept
close
send/recv
shutdown
close
socket
connect
send/recv
shutdown
close

The Number Server

- Unix Socket

  http://goo.gl/C0PqHz

- INET Socket

  http://bit.ly/17awYWz

Server (1)

```
int    sock, snew, fromlength, number, outnum;
struct sockaddr_in  master, from;
sock = socket (AF_INET, SOCK_STREAM, 0);

master.sin_family = AF_INET;
master.sin_addr.s_addr = INADDR_ANY;
master.sin_port = htons (MY_PORT);

bind (sock, (struct sockaddr*) &master, sizeof (master))

number = 0;
listen (sock, 5);
```

Server (2)

```
while (1) {
        fromlength = sizeof (from);
        snew = accept (sock, (struct sockaddr*) & from,
                       & fromlength);
        outnum = htonl (number);
        write (snew, &outnum, sizeof (outnum));
        close (snew);
        number++;
}
```

Sample code with Fork

- http://tinyurl.com/nwr9tbj

Fork

- The function fork() is called once (in the parent process) but it returns twice.

- waitpid() is used by the parent process to query the change state of a particular child.

Fork

```
while (1) {
        fromlength = sizeof (from);
        snew = accept (sock, (struct sockaddr*) & from,
                       & fromlength);
```

Fork

```
        outnum = htonl (number);
        write (snew, &outnum, sizeof (outnum));
```

Exit

```
        close (snew);
        number++;
}
```

Fork

```
while (1) {
        fromlength = sizeof (from);
        snew = accept (sock, (struct sockaddr*) & from,
                       & fromlength);

Fork    pid = fork();
        if(pid == 0) {
                outnum = htonl (number);
                write (snew, &outnum, sizeof (outnum));
Exit            exit(0);
        }
        close (snew);
        number++;
}
```

waitpid

```
static void kidhandler(int signum) {
        waitpid(WAIT_ANY, NULL, WNOHANG);
}

int main(int argc,  char *argv[])
        {
        struct sigaction sa;
        sa.sa_handler = kidhandler;
        sigemptyset(&sa.sa_mask);
        sa.sa_flags = SA_RESTART;
        if (sigaction(SIGCHLD, &sa, NULL) == -1)
                err(1, "sigaction failed");


}
```