CSE150 – Project 3
Networks and Distributed Systems

Group Thimbles

July 28, 2015

1 Networking syscalls

1.1 connect()

Port mapping Ports are mapped to sockets for both incoming and outgoing connections.

Pseudocode:

Attempt to initiate a new connection to the specified port on the specified remote host, and return a new file descriptor referring to the connection. connect() does not give up if the remote host does not respond immediately.

Returns the new file descriptor, or -1 if an error occurred.

```
int connect(host, port){
        disable interrupts
        create a new socket s and assign it to a free port in (0,127)
        s.state = SYN_SENT
        send SYN packet
        block until SYN/ACK recv'd // timeout breaks this
        s.state = ESTABLISHED
        enable interrupts
        return s.fileDescriptor
}
```

1.2 accept()

Pseudocode:

```
int acccept(port){
        disable interrupts
        if there are connections waiting on port
                create a new socket s, assign it that port
        else return -1
```

```
        s.state = ESTABLISHED
        send SYN/ACK
        enable interrupts
        return s.fileDescriptor
}
```

1.3 write()

write() allows the connection to write to the network. Attempts to write a buffer of bytes to the socket. If the socket is not Established it returns 1.

```
int write(fileDescriptor, buffer, count){

        if(state == ESTABLISHED && offset + length <= buf.length))
        {
                new packet
                int bytePos = offset;
                int endPos = offset + length;
                while(bytePos < endPos){

                System.arraycopy(buf, bytePos, contents, 8, amountSend);
                }
        }

        ...
        <netcode>
        ...
}
```

1.4 read()

read() allows the connection to read from the network. Attempts to read a number of bytes from the socket. If it is closed and there are no bytes in the buffer, it will return 1, otherwise return the number of bytes read. It does not block.

```
int read(fileDescriptor, buffer, count){
        ...
        // for a socket
        if (s.isOpen){
                read count bytes
                return bytes successfully read
        } else {
                if (socket isn't empty){
                        read count bytes
                        if (socket is empty)
```

```
                                delete socket
                            return bytes successfully read
                    }
            }
}
```

2 Threads

2.1 Send thread

2.2 Receive thread

2.3 Timeout thread

This thread works like waitUntil, where it loops through the existing sockets and checks for any that have lived past their timeout value. If they have, it closes that socket.

3 Test cases

3.1 connect()

- Attempt to open a connection to a node that doesn't exist
  Check that connect() blocks
- Open a connection to an existing node
  Check that connect() returns
- Close an already-open connection
  Verify that socket is closed on both sides
- Open multiple connections to the same receiving port
  Check that they all send/receive data
- Open a connection, close it and re-open it
- packet drop during connect()

3.2 accept()

- Accept a waiting connection
- Accept multiple waiting connections on the same port
- Accept multiple waiting connections to different ports
- Return from accept() on a port that doesn't have a connection waiting
- packet drop during accept()

3.3 close()

- Close a connection that doesn't exist
- Close a connection that exists

  Check that it's actually closed
- Close a connection twice in a row

3.4 read/write()

- simple read write.
- read returns -1 during remote host disconnection
- read returns even after disconnect
- write returns -1 during remote host disconnect
- write flushes data after close

3.5 title