

# CS 472 - Extra Credit

June 12, 2024

Satwik Shresth

## Implementation

- I created a UDP File Transfer Protocol server which can handle multiple clients sending in data for a file
- Original code given by prof was written in c with purely functional, I took most of the code and made very object oriented, so that concurrency is easier to implement
- Different classes to do different work
  - Connection
    - own socket created for udp and does does the main job of accepting the data and sending the acknowledgement
    - It is templated,so it can work with any new struct but they have to have the similar implementation
    - send function

```
template <typename PDU>
int Connection<PDU>::sendRaw(void* sbuff, int sbuff_sz)
{
    int bytesOut = 0;

    if (!outSockAddr.isAddrInit)
    {
        perror("sendRaw: connection not setup properly");
        return -1;
    }

    PDU* outPdu = (PDU*) sbuff;
    bytesOut = sendto(udpSock, (const char*) sbuff, sbuff_sz, 0, (const struct sockaddr*)
&(outSockAddr.addr), outSockAddr.len);

    outPdu->printOut(dbgMode);

    return bytesOut;
}
```

- recv function

```

    template <typename PDU>
int Connection<PDU>::recvRaw(void* buff, int buffSz)
{
    int bytes = 0;

    if (!inSockAddr.isAddrInit)
    {
        perror("recv: connection not setup properly - cli struct not init");
        return -1;
    }

    bytes = recvfrom(udpSock, (char*) buff, buffSz, MSG_WAITALL, (struct sockaddr*)
&(outSockAddr.addr), &(outSockAddr.len));

    if (bytes < 0)
    {
        perror("recv: received error from recvfrom()");
        return -1;
    }
    outSockAddr.isAddrInit = true;

    if (bytes > sizeof(PDU))
    {
        if (false)
        {
            PDU* inPdu = (PDU*) buff;
            char* payload = (char*) buff + sizeof(PDU);
            printf("DATA : %.*s\n", inPdu->dgram_sz, payload);
        }
    }

    PDU* inPdu = (PDU*) buff;
    inPdu->printIn(dbgMode);

    return bytes;
}

```

- FTPFileWriter

- Own the Channel Data structre and iterates over it until its closed
- Its main work is to take the buffer provided to ti and write it to the file
- `void writer::serverLoop()`

```
{
    while (!channel->isClosed())
    {
        std::string buff;

        try
        {
            buff = channel->receive();
        }
        catch (std::runtime_error e)
        {
            break;
        }
        FTP_PDU* pdu = reinterpret_cast<FTP_PDU*>(buff.data());
        std::cout << "filename: " << pdu->fileName << std::endl;

        auto mode = (pdu->status == Status::NEW) ? std::ios::trunc : std::ios::app;

        std::ofstream outFile{pdu->fileName, std::ios::out | mode};

        if (!outFile.is_open())
        {
            std::cerr << "ERROR: Cannot open file " << pdu->fileName << std::endl;
            exit(-1);
        }
        buff.erase(0, sizeof(FTP_PDU));
        outFile << buff;

        // fwrite(dataPtr, 1, dataSz, f);
        std::cout << "=====> \n" << buff << "\n=====>";
    }
    delete channel;
    closed = true;
}
```

- FTPServer
  - Owns connection and all the file writer
  - If a message comes with a request to connect creates a new filewriter, which intern creates a new channel
  - It creates a filewriter for each Sender, not for file, so that we can also limit the amount of load put on by a sender
  - listen function

```
void server::listen()
{
    ...
    rcvSz = dpc->recvRaw(dpc->_buffer, sizeof(dpc->_buffer));
    std::string address = inet_ntoa(dpc->getOutSockAddr()->addr.sin_addr);

    if (rcvSz == sizeof(PDU))
    {
        PDU pdu;
        pdu.seqnum = 0;
        pdu.mtype = MsgType::CNTACK;

        dpc->seqNums[inet_ntoa(dpc->getOutSockAddr()->addr.sin_addr)] = pdu.seqnum + 1;

        pdu.seqnum = dpc->seqNums[inet_ntoa(dpc->getOutSockAddr()->addr.sin_addr)];

        sndSz = dpc->sendRaw(&pdu, sizeof(PDU));

        if (sndSz != sizeof(PDU))
        {
            perror("listen: The wrong number of bytes were sent");
        }

        connected++;
        fw.push_back(new FTPFileWriter{address});

        pool->submit([&] {
            FTPFileWriter* writer = fw.back();
            fw.pop_back();
            writer->serverLoop();
            delete writer;
            ftpWriters.erase(writer->address);
        });

        ftpWriters[address] = fw.back();
        std::cout << "Connection established OK!" << std::endl;
    }
    else
    {
        ... handles the message accordingly
        ftpWriters[address]->pushToChannel(dpc->_buffer + sizeof(PDU), rcvSz - sizeof(PDU));
    }

    std::this_thread::sleep_for(std::chrono::seconds(3));
}
```

- FTPClient
  - Connectes to the Server and send it file data

## Example output

```

satalish@shwini: ~/5mp /  v18: h.0  14:38
./as_udp
./bin/du-ftp -s
MODE 1
PORT 2080
FILE NAME: test.c
Waiting for a new connection...
====> PDU DETAILS [IN]
      Version: 0
      Msg Type: CONNECT
      Msg Size: 0
      Seq Num: 0

PDU DETAILS ==> [OUT]
      Version: 1
      Msg Type: CONNECT/ACK
      Msg Size: 1
      Seq Num: 1

Connection established OK!
Waiting for a new connection...
====> PDU DETAILS [IN]
      Version: 0
      Msg Type: SEND FRAGMENT
      Msg Size: 512
      Seq Num: 1

PDU DETAILS ==> [OUT]
      Version: 1
      Msg Type: SEND FRAGMENT/ACK
      Msg Size: 0
      Seq Num: 513

=====
Size :512
=====
filename: rfc768.txt
=====

RFC 768                                J. Postel
                                         ISI
                                         28 August 1980


User Datagram Protocol
-----

Introduction
-----

This User Datagram Protocol (UDP) is defined t
=====

```

- As you can see the server starts and accepts a file rfc768 from one client.

```

Introduction
=====
This User Datagram Protocol (UDP) is defined t
=====
Waiting for a new connection...
==> PDU DETAILS [In]
    Version: 0
    Msg Type: CONNECT
    Msg Size: 0
    Seq Num: 0

YOU DETAILS ==> [Out]
    Version: 1
    Msg Type: CONNECT/ACK
    Msg Size: 1
    Seq Num: 1

[connection established OK]
Waiting for a new connection...
==> PDU DETAILS [In]
    Version: 0
    Msg Type: SEND
    Msg Size: 212
    Seq Num: 513

YOU DETAILS ==> [Out]
    Version: 1
    Msg Type: SEND/ACK
    Msg Size: 0
    Seq Num: 213

=====
Size: 212
Filename: rfc768.txt
=====
# make available #
datagram mode of packet-switched computer communication in the
twir

=====
Waiting for a new connection...
==> PDU DETAILS [In]
    Version: 0
    Msg Type: SEND FRAGMENT
    Msg Size: 512
    Seq Num: 1

YOU DETAILS ==> [Out]
    Version: 1
    Msg Type: SEND FRAGMENT/ACK
    Msg Size: 0
    Seq Num: 725

=====
Size: 512
Filename: test.c
=====
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <unistd.h>
#include <sys/types.h>

#include "ds_proto.h"

static char *buffer[DP_MAX_FRAME_SIZE];
static int showPhn = 1;

static dp_conn_t initC() {
    dp_conn_t connession = malloc(sizeof(dp_connection));
    bzero(connession, sizeof(dp_connection));
    connession->remoteAddr.sin_addr.sint = false;
    return connession;
}

=====
Waiting for a new connection...

```

- Now you can see the server connecting to another client, receiving data for test.c and processing it

```
satevikshresth 2: v15.0.0 14:35
~/a5_udp
➔ ./bin/du-ftp -f ./outfile/rfc768.txt
MODE 0
PORT 2080
FILE NAME: ./outfile/rfc768.txt
PDU DETAILS ==> [OUT]
  Version: 0
  Msg Type: CONNECT
  Msg Size: 0
  Seq Num: 0

==> PDU DETAILS [IN]
  Version: 1
  Msg Type: CONNECT/ACK
  Msg Size: 1
  Seq Num: 1

Connection established OK!
PDU DETAILS ==> [OUT]
  Version: 0
  Msg Type: SEND FRAGMENT
  Msg Size: 512
  Seq Num: 1

==> PDU DETAILS [IN]
  Version: 1
  Msg Type: SEND FRAGMENT/ACK
  Msg Size: 0
  Seq Num: 513

PDU DETAILS ==> [OUT]
  Version: 0
  Msg Type: SEND
  Msg Size: 212
  Seq Num: 513

==> PDU DETAILS [IN]
  Version: 1
  Msg Type: SEND/ACK
  Msg Size: 0
  Seq Num: 213

PDU DETAILS ==> [OUT]
  Version: 0
  Msg Type: SEND FRAGMENT
  Msg Size: 512
  Seq Num: 725

==> PDU DETAILS [IN]
  Version: 1
  Msg Type: SEND FRAGMENT/ACK
  Msg Size: 0
  Seq Num: 1237

PDU DETAILS ==> [OUT]
  Version: 0
  Msg Type: SEND
  Msg Size: 212
  Seq Num: 1237

^C
satevikshresth 2: v15.0.0 15:48
~/a5_udp
```

- here you can see me starting and sending rfc768.txt connecting and sending data to server

```
satevikshresth 2: v15.0.0 14:35
~/a5_udp
➔ ./bin/du-ftp -f ./outfile/test.c -c
MODE 0
PORT 2080
FILE NAME: ./outfile/test.c
PDU DETAILS ==> [OUT]
  Version: 0
  Msg Type: CONNECT
  Msg Size: 0
  Seq Num: 0

==> PDU DETAILS [IN]
  Version: 1
  Msg Type: CONNECT/ACK
  Msg Size: 1
  Seq Num: 1

Connection established OK!
PDU DETAILS ==> [OUT]
  Version: 0
  Msg Type: SEND FRAGMENT
  Msg Size: 512
  Seq Num: 1

==> PDU DETAILS [IN]
  Version: 1
  Msg Type: SEND FRAGMENT/ACK
  Msg Size: 0
  Seq Num: 725

PDU DETAILS ==> [OUT]
  Version: 0
  Msg Type: SEND
  Msg Size: 212
  Seq Num: 513

^C
```

- here you can see me starting and sending test.c connecting and sending data to server

```

a5_udp
├── .cache
├── .git
├── bin
├── infile
├── obj
├── outfile
├── src
├── channel
├── dnxelprotocol
├── client.h
├── connection.h
├── ftp.h
├── msgtype.h
├── pdu.h
├── server.h
├── threadpool
├── jointhread.h
├── threadpool.h
├── threadqueue.h
├── workstealqueue.h
├── client.cpp
├── jointhread.cpp
├── main.cpp
├── msgtype.cpp
├── server.cpp
├── threadpool.cpp
├── .clang-format
├── .DS_Store
├── .gitignore
├── compile_flags.txt
├── makefile
├── README.md
├── rfc768.txt
├── test.c
└── test.txt

```

```

1
2
3 RFC 768
4 J. Postel
5 ISI
6 28 August 1980
7
8
9 User Datagram Protocol
10 -----
11
12 Introduction
13 -----
14
15 This User Datagram Protocol (UDP) is defined to make available a
16 datagram mode of packet-switched computer communication in the
17 environment of an interconnected set of computer networks. This
18 protocol assumes that the Internet Protocol (IP) [1] is used as the
19 underlying protocol.
20
21 This protocol provides a procedure for application programs to send
22 messages to other programs with a minimum of protocol mechanism. The
23 protocol is transaction oriented, and delivery and duplicate protection
24 are not guaranteed

```

- Here you can see server writing this file in a new file outside of outfile the data it was sent

```

/Users/satwikshreshth/Projects/a5_udp
├── .cache
├── .git
├── bin
├── infile
├── obj
├── outfile
├── src
├── .clang-format
├── .DS_Store
├── .gitignore
├── compile_flags.txt
├── makefile
├── README.md
├── rfc768.txt
├── test.c
└── test.txt

```

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <stdbool.h>
5 #include <unistd.h>
6 #include <sys/un.h>
7 #include "du_proto.h"
8
9 static char _dpBuffer[DP_MAX_FRAME_SZ];
10 static int _debugMode = 1;
11
12 static dp_conn dp_init(){
13     dp_conn dpession = malloc(sizeof(dp_connection));
14     bzero(dpession, sizeof(dp_connection));
15     dpession->outSockAddr.isAddrInit = false;
16     dpession

```

- Here you can see server writing this file in a new file outside of outfile the data it was sent for test.c