# 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 acknoledgement
    - It is templated,so it can work with any new struct but they have to have the similar implementation
    - send funciton

```
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========================>
    \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});

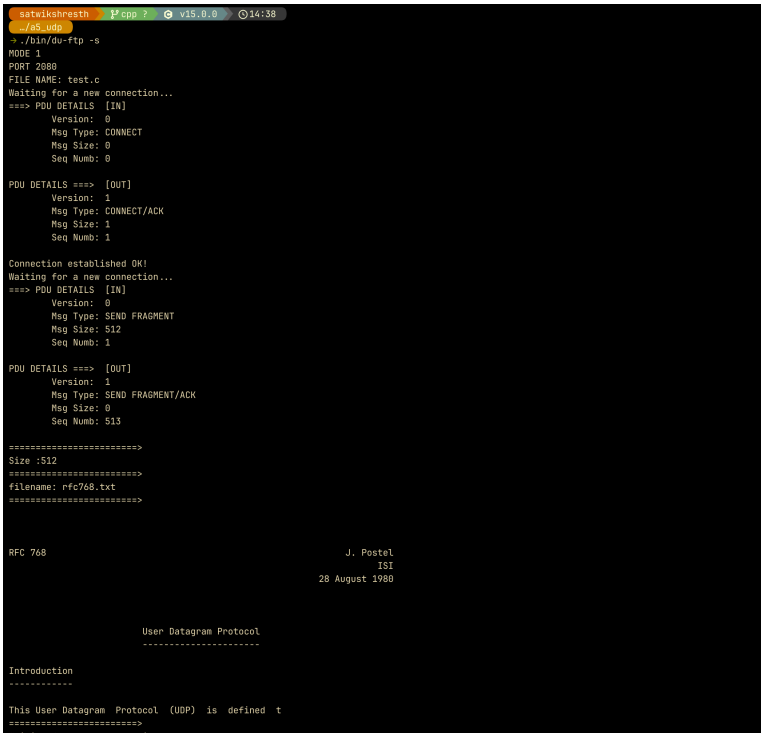
      pool->submit([&] {
         FTPFileWriter* writer = fw.back();
         fw.pop_back();
         writer->serverLoop();
         delete writer;
         ftpWriters.erase(writer->address);
      });

      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));
   }

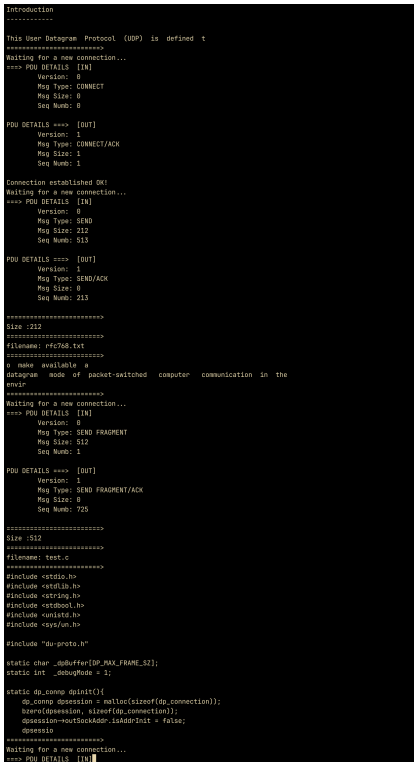
   std::this_thread::sleep_for(std::chrono::seconds(3));
}
```

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

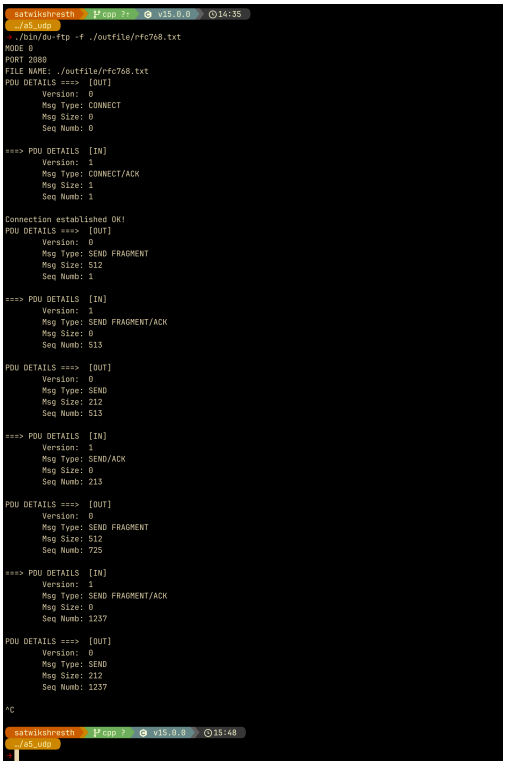
## Example output



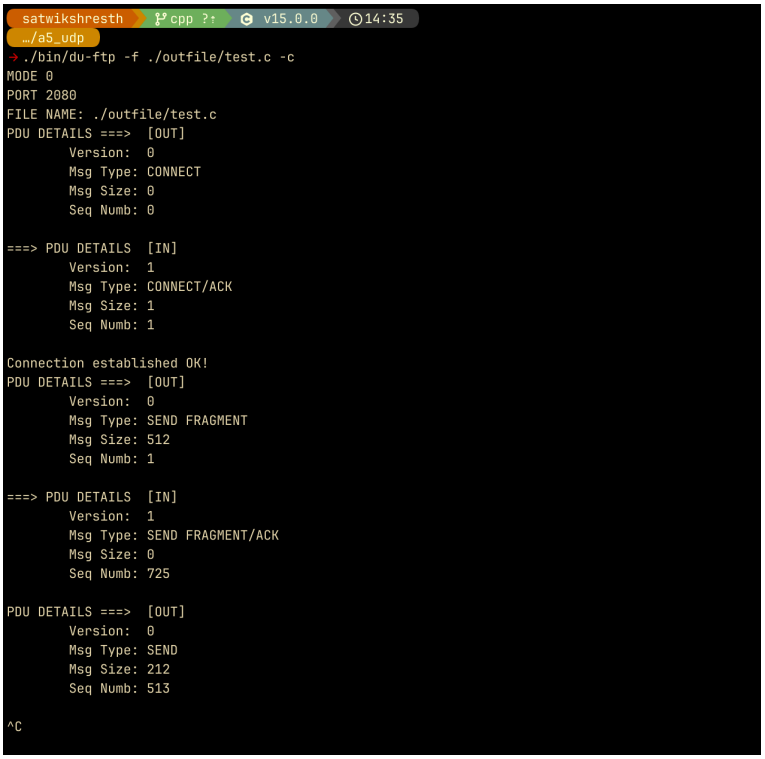

- As you can see the server starts and accepts a file rfc768 from one client




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

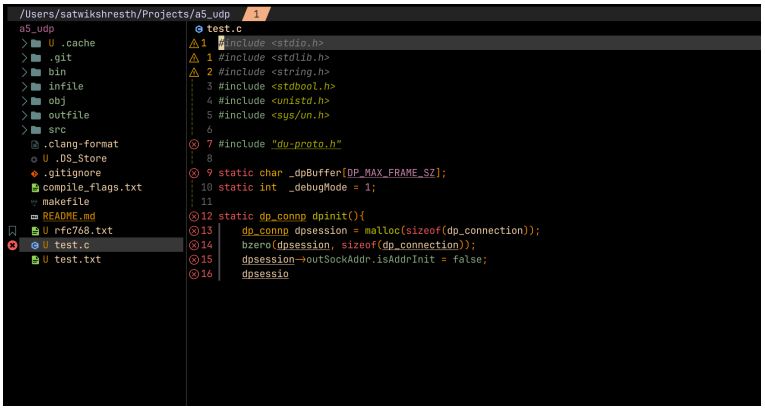



- here you can see me starting and sending rfc768.txt connecting ans sending data to server




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


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




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