<컴퓨터 네트워크 2차 프로젝트>

팀명: YonseiAlone

구성인원: 1명

구성원 학번: S20181623

구성원 이름: 김효민

1) 구현 환경

장치 사양

장치 이름	GK-gram
프로세서	Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz 1.80 GHz
설치된 RAM	8.00GB(7.87GB 사용 가능)
장치 ID	BC44F7F5-D5BE-49E5-AD88-DFAF35E0B902
제품 ID	00328-20160-00000-AA934
시스템 종류	64비트 운영 체제, x64 기반 프로세서
펜 및 터치	이 디스플레이에 사용할 수 있는 펜 또는 터치식 입력이 없습니다.

복사

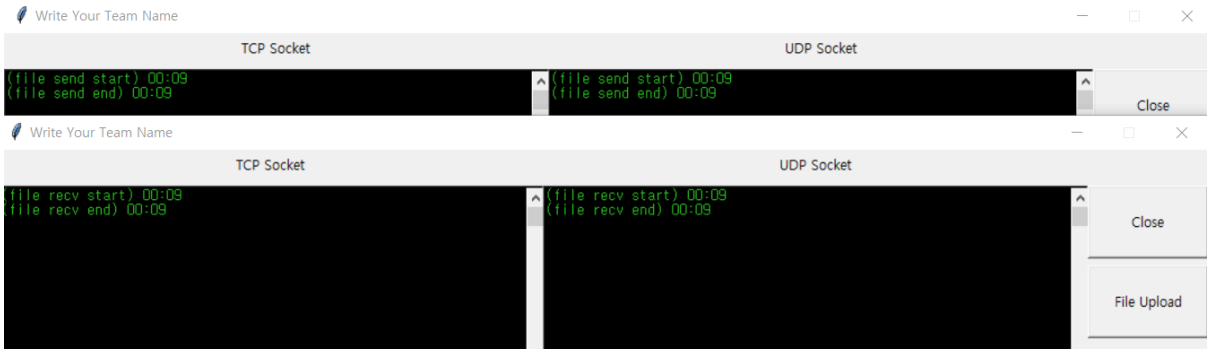
이 PC의 이름 바꾸기

Windows 사양

에디션	Windows 10 Education
버전	20H2
설치 날짜	2021-05-03
OS 빌드	19042.2006
경험	Windows Feature Experience Pack 120.2212.4180.0

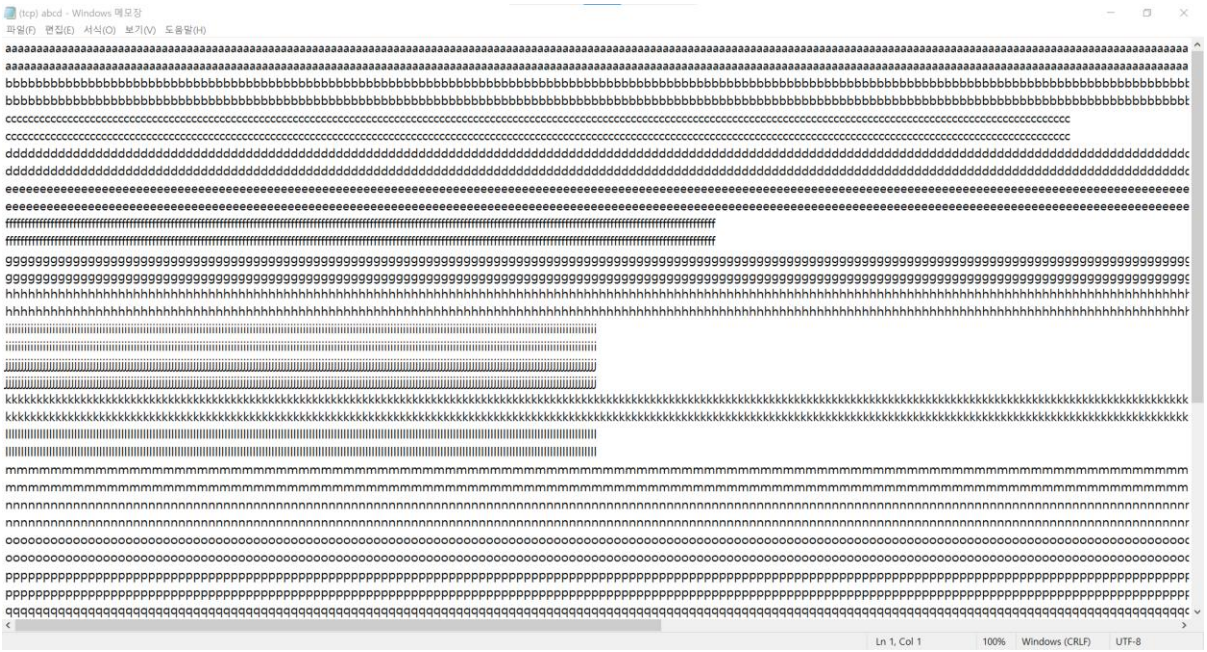
컴퓨터는 위와 같고 파이썬 버전은 3.11.0을 사용했다.

2) 정상 동작 스크린샷

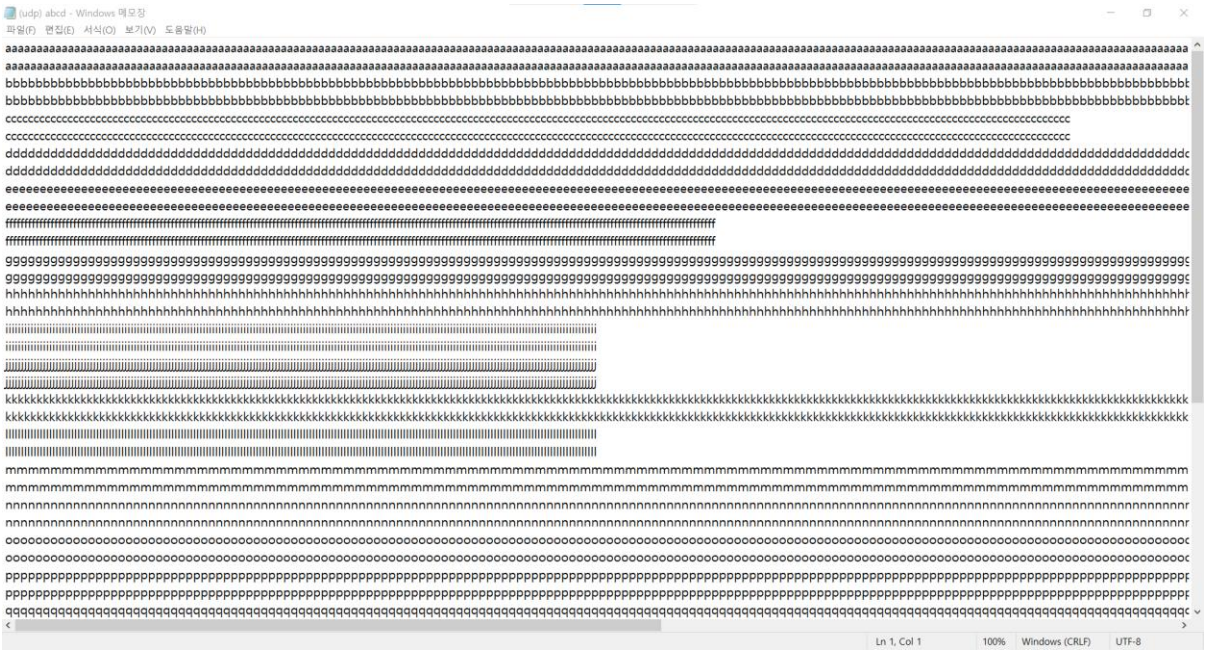


Both 옵션으로 파일을 하나만 전송했을 때의 모습이다.

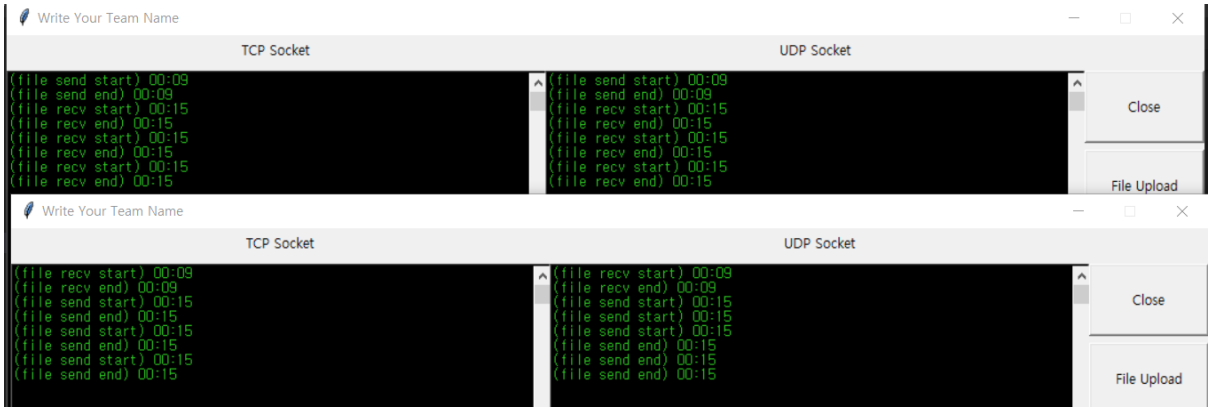
원본 파일을 이런 구성으로 이루어져 있었는데



tcp 통신으로 전해 받은 파일은 원본과 차이가 없다.



그리고 udp 통신으로 전해 받은 파일 또한 원본과 차이가 없음을 알 수 있다.



한번에 여러 개의 파일을 전송했을 때 전송 화면은 이와 같이 올바르게 나오고

(tcp) aaa	2022-11-19 오전 12:15	텍스트 문서	75KB
(tcp) aab	2022-11-19 오전 12:15	텍스트 문서	200KB
(tcp) aac	2022-11-19 오전 12:15	텍스트 문서	200KB
(tcp) abcd	2022-11-19 오전 12:09	텍스트 문서	10KB
(udp) aaa	2022-11-19 오전 12:15	텍스트 문서	75KB
(udp) aab	2022-11-19 오전 12:15	텍스트 문서	200KB
(udp) aac	2022-11-19 오전 12:15	텍스트 문서	200KB
(udp) abcd	2022-11-19 오전 12:09	텍스트 문서	10KB
aaa	2022-11-18 오후 1:30	텍스트 문서	75KB
aab	2022-11-18 오후 3:13	텍스트 문서	200KB
aac	2022-11-18 오후 3:13	텍스트 문서	200KB
abcd	2022-11-19 오전 12:03	텍스트 문서	10KB

download 폴더 안의 파일들을 원본 파일들과 비교해보면 보는 사진과 같이 용량들이 원본과 같음을 알 수 있다.

그러므로 이와 같은 코드는 udp 통신일 때 packet loss가 일어나면 잘 처리해서 data loss가 발생하지 않도록 함을 알 수 있다.

iii) 사용한 pipeline 방식

Selective Repeat 방식을 채용했다.

iv) 코드 설명

- tcp_file_send

```python
def tcp_file_send(self, filename: str, tcp_send_func: Callable)-> None: #파일 전송
    basename = os.path.basename(filename)
    self.file_pointer = open(filename, "rb")

    # packet의 파일 이름(basename)을 전송한다.
    name_packet = self.tcp_file_name_packet(basename)
    tcp_send_func(name_packet)
    # 이름 전송 종료

    # 파일을 구성하는 data를 전송한다.
    # tcp_file_data_packet이 생성하는 packet을 tcp를 이용해 전부 전송한다.
    isThis, data_packet = self.tcp_file_data_packet()
    while isThis == True:
        tcp_send_func(data_packet)
        isThis, data_packet = self.tcp_file_data_packet()
    # 파일 data 전송 종료

    # TCP_FILE_TRANSFER_END을 전송하여
    # 파일의 전송이 끝냈음을 알린다.
    tcp_send_func(TCP_FILE_TRANSFER_END)
    # TCP_FILE_TRANSFER_END을 전송 종료

    # 파일 닫기
    self.file_pointer.close()
    self.file_pointer = None
```

우선 tcp_file_name_packet 함수를 이용해서 name_packet을 생성한 다음 tcp_send_func 함수를 이용해서 전송한다. 그러고 tcp_file_data_packet 함수를 통해서 파일의 데이터가 없을 때까지 읽고 패킷을 보내는 행동을 반복한다. 마지막에는 TCP_FILE_TRANSFER_END 보내고 파일을 닫는다.

- tcp_file_recieve

```python
def tcp_file_receive(self, packet) -> int: #파일 수신 및 저장
    packet_type, data = self.tcp_packet_unpack(packet)

    if packet_type == PACKET_TYPE_FILE_START:
        # 파일의 이름을 받아 file_path 위치에 self.file_pointer를 생성한다.
        basename = data.decode(ENCODING)
        self.file_name = basename
        file_path = './downloads/(tcp) '+basename
        self.file_pointer = open(file_path, "wb")
        return 0

    elif packet_type == PACKET_TYPE_FILE_DATA:
        # self.file_pointer에 전송 받은 data를 저장한다.
        file_data = data.decode(ENCODING)
        self.file_pointer.write(data)
        return 1

    elif packet_type == PACKET_TYPE_FILE_END:
        # 파일 전송이 끝난 것을 확인하고 file_pointer를 종료한다.
        self.file_pointer.close()
        return 2
```

받은 패킷의 타입이 PACKET_TYPE_FILE_START면 파일 이름에 대한 패킷이 온 것이므로 지정한 경로에 건네 받은 파일 이름에 대한 파일을 생성한다.

받은 패킷의 타입이 PACKET_TYPE_FILE_DATA면 파일 내용에 대한 패킷이 온 것이므로 새로 생성한 파일에 내용을 적어준다.

받은 패킷의 타입이 PACKET_TYPE_FILE_END면 파일이 끝났다는 것을 알려주므로 열려 있던 파일을 닫아준다.

- udp_send_with_record

```python
def udp_send_with_record(self, packet_type: bytes, data: bytes, udp_send_func: Callable) -> None: #전송할 데이터를 패킷으로 변환 후, 패킷 정보(전송 time, packet) 저장
    #새로 전송할 패킷 넘버(self.udp_last_ack_num) 업데이트
    # GBN, SR을 통한 재전송을 위해 packet과 전송 시간을 self.udp_send_packet에 저장한다.
    # 또한 self.udp_lask_ack_num을 update하여 새로 전송할 packet의 ack_num를 update한다.
    packet = self.udp_packet_pack(packet_type, self.udp_last_ack_num, data)
    udp_send_func(packet)
    self.udp_send_packet[self.udp_last_ack_num] = (time(), packet)
    self.udp_last_ack_num = (self.udp_last_ack_num + 1) % UDP_MAX_ACK_NUM
```

이 함수는 전송할 데이터를 패킷으로 만들어 준 후에 전송하고 그에 따라 udp_send_packet에 내용을 기록해주고 udp_last_ack_num을 업데이트 해주는 함수다.

- udp_file_send

```python
def udp_file_send(self, filename: str, udp_send_func: Callable) -> None: #파일 전송
    basename = os.path.basename(filename)
    self.file_pointer = open(filename, "rb")
    # udp를 통해 파일의 basename을 전송하고 ack를 기다린다.
    # hint : self.udp_file_name_transfer 함수를 활용할 것
    with lock:
        self.udp_file_name_transfer(basename, udp_send_func)
    while self.udp_ack_num != self.udp_last_ack_num:
        with lock:
            if self.udp_time_out() == True:
                self.udp_file_name_transfer(basename, udp_send_func)
        sleep(UDP_WAIT)

    data_ready, data = self.udp_file_data()
    while data_ready:
        if len(self.udp_send_packet) < UDP_WINDOW_SIZE: #window의 크기보다 전송한 패킷의 양의 적은 경우
            diff = self.udp_last_ack_num - self.udp_ack_num
            if diff < 0:
                diff = UDP_MAX_ACK_NUM - self.udp_ack_num + self.udp_last_ack_num
            if diff < UDP_WINDOW_SIZE:
                with lock:
                    self.udp_send_with_record(PACKET_TYPE_FILE_DATA, data, udp_send_func)
                data_ready, data = self.udp_file_data() # 다음 전송할 data를 준비한다.
        else: # PIPELINE을 위한 window를 전체를 사용하여 ack를 기다리며 timeout에 대처한다.
            with lock:
                if self.udp_time_out() == True:
                    self.udp_pipeline(udp_send_func)
            sleep(UDP_WAIT)
    # 모든 파일 data의 ack를 기다리고 timeout에 대처한다.
    while self.udp_ack_num != self.udp_last_ack_num:
        with lock:
            if self.udp_time_out() == True:
                self.udp_pipeline(udp_send_func)
        sleep(UDP_WAIT)
    # 파일 전송이 완료되었음을 알리고 ack에 대비한다.
    with lock:
        self.udp_send_with_record(PACKET_TYPE_FILE_END, bytes(1), udp_send_func)
    while self.udp_ack_num != self.udp_last_ack_num:
        with lock:
            if self.udp_time_out() == True:
                self.udp_send_with_record(PACKET_TYPE_FILE_END, bytes(1), udp_send_func)
        sleep(UDP_WAIT)
    self.file_pointer.close()
    self.file_pointer = None
```

처음에 파일 이름에 대한 패킷을 보냈을 때는 udp_file_name_transfer 함수를 사용하고 여기서는 timeout이 발생하면 다시 udp_file_name_transfer 함수를 호출한다.

파일 데이터에 대한 패킷을 보낼 때에는 udp_file_data 함수를 이용해서 파일의 데이터를 읽어온다. 그리고 현재 윈도우 범위 밖에 있는 패킷은 보내지 말고 일단 기다린다. 해당 조건은 udp_last_ack_num과 udp_ack_num의 값의 차이가 윈도우 사이즈를 벗어나지 않는 지를 체크함으로써 알 수 있다. 그러고 여기서는 timeout이 발생하면 udp_pipeline 함수를 호출해준다.

파일에 대한 패킷을 전부 다 보냈다면 이미 보낸 패킷에 대한 ACK를 기다린다. 여기서 우리가 모든 패킷에 대해 ACK를 받았다는 사실을 알 수 있는 방법은 udp_last_ack_num과 udp_ack_num의 값이 같아질 경우에 조건을 만족했음을 알 수 있으므로 그 두 값이 다른 동안 계속해서 타임아웃을 체크해준다. 여기서는 타임아웃이 발생했을 때 udp_pipeline 함수를 사용한다.

마지막으로 파일 전송이 완료되었음을 알려줘야 하는데 이는 udp_send_with_record 함수를 이용해서 알려준다. 여기서는 timeout이 발생했을 때 udp_send_with_record 함수를 다시 호출해준다. 그러고 파일을 닫아준다.

- udp_file_reciever

```python
def udp_file_receive(self, packet: bytes, udp_send_func: Callable) -> int: #파일 수신 및 저장
    ack_bytes = self.udp_ack_bytes(packet)
    packet_type, ack_num, data = self.udp_packet_unpack(packet)
    if packet_type != PACKET_TYPE_FILE_ACK:
        # 받은 packet에 대한 ack를 전송한다.
        self.udp_ack_send(ack_bytes, udp_send_func)

    if packet_type == PACKET_TYPE_FILE_START:  # file transfer start
        # 파일의 이름을 받아 file_path 위치에 self.file_pointer를 생성하고.
        # 그다음 받을 파일의 data의 시작 packet의 ack_num를 self.file_packet_start에 저장하여
        # 연속된 packet을 받을 수 있게 준비한다.
        if self.file_pointer is not None:
            self.file_pointer.close()
        basename = data.decode(ENCODING)
        self.file_name = basename
        file_path = './downloads/(udp) '+basename
        self.file_pointer = open(file_path, "wb")
        self.file_packet_start = ack_num
        return 0

    elif packet_type == PACKET_TYPE_FILE_DATA:  # file transfer
        if not self.udp_recv_flag[ack_num]:
            # 처음 받은 packet인지 확인하고
            # 처음 받은 packet이라면 self.udp_recv_packet[ack_num]에 저장하고
            self.udp_recv_packet[ack_num] = data
            self.udp_recv_flag[ack_num] = True
        # self.udp_recv_packet에 self.file_packet_start에서 부터 연속된
        # 패킷이 저장되어 있다면 이를 self.file_pointer를 이용해 파일로 저장하고
        # self.udp_recv_flag를 update한다.
        # 또한 self.file_packet_start 역시 update한다.
        check = (self.file_packet_start + 1) % UDP_MAX_ACK_NUM
        while self.udp_recv_flag[check] == True:
            self.file_pointer.write(self.udp_recv_packet[check])
            self.udp_recv_flag[check] = False
            self.file_packet_start = check
            check = (check + 1) % UDP_MAX_ACK_NUM
        return 1


    elif packet_type == PACKET_TYPE_FILE_END:  # file transfer end
        # 파일 전송이 끝난 것을 확인하고 파일을 종료한다.
        if self.file_pointer is not None:
            self.file_pointer.close()
            self.file_pointer = None
        return 2
```

이 부분은 해당 함수에서 리시버 측이 사용하게 되는 기능들을 나타낸다.

우선 패킷의 타입이 PACKET_TYPE_FILE_ACK가 아니라면 받은 패킷에 대한 ACK를 udp_ack_send 함수를 이용해서 전송해준다.

그리고 패킷의 타입이 PACKET_TYPE_FILE_START인 경우, 전달 받은 파일 이름에 대해서 지정한 경로에 파일을 생성한다.

패킷의 타입이 PACKET_TYPE_FILE_DATA인 경우, 처음 받은 패킷이라면 udp_recv_packet에다가 해당 패킷을 저장한다. 그러고 check 변수를 이용해서 file_packet_start+1 값에서부터 하나씩 이동하면서 udp_recv_flag가 True인 경우에 해당 패킷을 파일에다가 써주고 그 flag 값을 False로 바꿔준다. 그러고 다시 file_packet_start 값을 check로 초기화 시켜준다. 이러한 과정을 반복하여 연속된 패킷들을 처리해주게 된다.

패킷의 타입이 PACKET_TYPE_FILE_END인 경우, 파일을 닫아준다.

```
elif packet_type == PACKET_TYPE_FILE_ACK:  # ack
    # GBN, SR을 위해 self.udp_ack_windows를 update한다.
    # hint: self.udp_ack_num으로 부터 연속되게 ack를 받은 경우
    # window를 옮겨준다 (self.udp_send_packet에 저장된 packet도 처리해줄 것)
    self.udp_ack_windows[ack_num] = True
    while self.udp_ack_windows[self.udp_ack_num] == True:
        with lock:
            self.udp_send_packet.pop(self.udp_ack_num)
        self.udp_ack_windows[self.udp_ack_num] = False
        self.udp_ack_num = (self.udp_ack_num + 1) % UDP_MAX_ACK_NUM
    return 1
```

이 부분은 sender측에서 사용하게 된다.

우선 전달받은 ack_num에 대하여 udp_ack_windows 값을 True로 만들어주고 udp_ack_num을 이용해서 윈도우 처음부터 udp_ack_windows의 값이 False가 될 때까지 udp_send_packet의 제일 앞에 있는 값을 pop해주고 udp_ack_windows의 값을 False로 만들어 준 다음 윈도우를 하나 옆으로 옮겨준다.

- udp_pipeline

```
def udp_pipeline(self, udp_send_func: Callable) -> None: #타임아웃 이후에 패킷 재전송
    # GBN, SR 중 하나의 알고리즘을 선택하여 ACK를 관리한다.
    # hint: self.udp_send_packet[ack_num]에 저장시
    # (send time, packet)형태로 저장할 것
    udp_send_func(self.udp_send_packet[self.udp_ack_num][1])
    self.udp_send_packet[self.udp_ack_num] = (time(), self.udp_send_packet[self.udp_ack_num][1])
```

이 함수는 SR 방식을 따르고 있다. 타임아웃이 발생하기 전에 오는 ACK에 대해서 다 기록해두었으므로 타임아웃이 발생하면 제일 앞에 있는 아직 ACK를 받지 못한 패킷을 다시 전송하고 그 패킷에 대한 udp_send_packet의 time을 새로 보낸 시간으로 바꿔준다.

v) 추가적인 설명

- %UDP_MAX_ACK_NUM을 이용해서 이 수를 벗어나는 ACK_NUM에 대해서도 계속해서 처리할 수 있게 만들어 주었다.

- threading 라이브러리를 import하고, with lock을 이용해서 동시성 문제를 해결하였다.

- txt 파일이 전송 가능하다.