22.11.2024 r.

PSI-Sprawozdanie-Zadanie 1.2

Autorzy
Daniel Machniak 325190
Natalia Pieczko 325208
Krzysztof Gólcz 325159

1. Treść zadania

Wychodzimy z kodu z zadania 1.1, tym razem pakiety datagramu mają stałą wielkość, można przyjąć np. 512B. Należy zaimplementować prosty protokół niezawodnej transmisji, uwzględniający możliwość gubienia datagramów. Rozszerzyć protokół i program tak, aby gubione pakiety były wykrywane i retransmitowane. Wskazówka – „Bit alternate protocol”. Należy uruchomić program w środowisku symulującym błędy gubienia pakietów. (Informacja o tym, jak to zrobić znajduje się w skrypcie opisującym środowisko Dockera).

To zadanie można wykonać, korzystając z kodu klienta i serwera napisanych w C lub w Pythonie (do wyboru). Nie trzeba tworzyć wersji w obydwu językach.

2. Rozwiązanie

Datagram w naszym rozwiązaniu składa się z dwóch części:

- Pierwszy bajt zawierający bit weryfikujący pełni rolę kontroli poprawności przesyłanych danych
- Przesyłana wiadomość o długości n - 1 dopełniona znakiem null, gdzie n to przesyłana liczba bajtów

Klient

```
def generate_datagram(no: int, length: int, seq_bit: bool):
    payload = (
        f"Message no. {no} with length {length} bytes and seq_bit {seq_bit}".encode()
    )

    datagram = seq_bit.to_bytes(1, "big") + payload.ljust(length - 1, b"\0")

    return datagram
```

Aby wygenerować nowy datagram klient używa powyższej funkcji, a funkcja główna wygląda następująco:

```python
def main():
    args = parse_args()

    host = args.host
    port = args.port
    bufsize = args.bufsize
    timeout = args.timeout

    no = 1
    seq_bit = 0
    ack_recv = True

    with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as s:
        s.settimeout(timeout)

        while work():
            if ack_recv:
                datagram = generate_datagram(no, bufsize, seq_bit)
            print("\n")
            print("-" * 50)
            print(
                f"Sending datagram #{no} (Seq: {seq_bit}) to server {host}:{port}\n"
                f"Datagram: {datagram.rstrip(b"\0")}\n",
                sep="",
            )
            s.sendto(datagram, (host, port))
            try:
                response, server = s.recvfrom(bufsize)
                response = response.rstrip(b"\0").decode()

                if response == f"ACK {seq_bit}":
                    print(
                        f"ACK received: Datagram #{no} acknowledged by server {server}"
                    )

                    seq_bit = 1 - seq_bit
                    no += 1
                    ack_recv = True
                else:
                    print(
                        f"Incorrect ACK received from server {server}\n",
                        f"Expected seq bit {seq_bit},but received {int(not seq_bit)}\n",
                        "Retrying...",
                        sep="",
                    )
                    ack_recv = False
            except TimeoutError:
                print(
                    f"Timeout: No ACK received for datagram #{no}\n",
                    "Retrying...",
                    sep="",
                )
                ack_recv = False
```

```
            print("-" * 50)
```

Na początku klient pobiera argumenty wywołania skryptu. Używając uzyskanego deskryptora gniazda przy pomocy socket(), klient sprawdza czy powinien wygenerować nowy datagram przy pomocy zmiennej ack_recv, która w pierwszej pętli zawsze będzie prawdziwa. Następnie wysyła aktualny datagram do serwera i próbuje uzyskać od niego odpowiedź przy użyciu recv(). Jeżeli odpowiedź serwera, a dokładniej bit weryfikujący jest zgodny z aktualnym, to zmienia ten bit, inkrementuje numer datagramu oraz zmienia zmienną ack_recv na True żeby w kolejnej iteracji wygenerować nowy datagram. Gdy odpowiedź serwera nie zgadza się z aktualnym bitem weryfikującym, to ustawia zmienną ack_recv żeby w następnej iteracji powtórzyć próbę wysłania datagramu i uzyskania poprawnej odpowiedzi. Takie same kroki podejmowane są, gdy otrzyma TimeoutError.

Serwer

```
def main():
    args = parse_args()

    port = args.port
    host = args.host
    bufsize = args.bufsize

    seq_bit = None

    with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as s:
        s.bind((host, port))

        print(f"UDP server up and listening on {host}:{port}")

        while Work():
            data, address = s.recvfrom(bufsize)

            recv_seq_bit = data[0]

            print("\n")
            print("-" * 50)
            print(
                f"Datagram received from {address}",
                f"Seq bit: {recv_seq_bit}\n",
                f"Datagram: {data.rstrip(b"\0")}\n",
                sep="",
            )

            if not seq_bit or seq_bit != recv_seq_bit:
                print("New data. Updating seq bit")
                seq_bit = recv_seq_bit
            else:
                print("Data duplicated or out-of-order. Sending ACK again")

            ack_datagram = f"ACK {seq_bit}".encode().ljust(bufsize, b"\0")
```

```
            s.sendto(ack_datagram, address)
            print(f"Sent acknowledgment: {ack_datagram.rstrip(b"\0")}")
            print("-" * 50)
```

Po uruchomieniu klienta, podobnie jak przy uruchomieniu serwera, pobieramy argumenty wywołania programu oraz uzyskujemy deskryptor gniazda używając socket(). Następnie otrzymuje datagram oraz adres, z którego pochodzi przy pomocy recv(). Wiedząc, że w używanym protokole pierwszy bajt zawiera bit weryfikujący, zapisuje go w zmiennej recv_seq_bit. W kolejnym kroku, jeżeli jest to pierwszy otrzymany datagram lub bit weryfikujący serwera jest inny niż datagramu, to do bit weryfikujący serwera ustawiamy na ten datagramu. W następnym kroku tworzymy wiadomość zwrotną oraz wysyłamy ją do klienta.

3. Konfiguracja testowa

Klient

Czas oczekiwania na potwierdzenie datagramu: 1s

Serwer

Adres IP: 172.21.33.31
Port: 8000

4. Testowanie

Zostały przeprowadzone testy na dwóch środowiskach:

1. Dwa osobne kontenery komunikujące się na ww. adresach i portach bez zakłóceń sieci
2. Dwa osobne kontenery komunikujące się na ww. adresach i portach, gdzie do kontenera wysyłającego datagramy wstrzykiwany jest program tc z opóźnieniem 1000 ms z rozrzutem (jitter) 500 ms i prawdopodobieństwem „zagubienia” pakietu równym 50%, który zakłóca działanie sieci

Środowisko 1.

Klient

```
------------------------------------------------
Sending datagram #1 (Seq: 0) to server 172.21.33.31:8080
Datagram: b'\x00Message no. 1 with length 512 bytes and seq_bit 0'

ACK received: Datagram #1 acknowledged by server ('172.21.33.31', 8080)
------------------------------------------------



------------------------------------------------
Sending datagram #2 (Seq: 1) to server 172.21.33.31:8080
Datagram: b'\x01Message no. 2 with length 512 bytes and seq_bit 1'

ACK received: Datagram #2 acknowledged by server ('172.21.33.31', 8080)
------------------------------------------------



------------------------------------------------
Sending datagram #3 (Seq: 0) to server 172.21.33.31:8080
Datagram: b'\x00Message no. 3 with length 512 bytes and seq_bit 0'

ACK received: Datagram #3 acknowledged by server ('172.21.33.31', 8080)
------------------------------------------------



------------------------------------------------
Sending datagram #4 (Seq: 1) to server 172.21.33.31:8080
Datagram: b'\x01Message no. 4 with length 512 bytes and seq_bit 1'

ACK received: Datagram #4 acknowledged by server ('172.21.33.31', 8080)
------------------------------------------------



------------------------------------------------
Sending datagram #5 (Seq: 0) to server 172.21.33.31:8080
Datagram: b'\x00Message no. 5 with length 512 bytes and seq_bit 0'

ACK received: Datagram #5 acknowledged by server ('172.21.33.31', 8080)
------------------------------------------------
```

Serwer

```
------------------------------------------------
Datagram received from ('172.21.33.2', 57242)
Seq bit: 0
Datagram: b'\x00Message no. 1 with length 512 bytes and seq_bit 0'

New data. Updating seq bit
Sent acknowledgment: b'ACK 0'
------------------------------------------------



------------------------------------------------
Datagram received from ('172.21.33.2', 57242)
Seq bit: 1
Datagram: b'\x01Message no. 2 with length 512 bytes and seq_bit 1'

New data. Updating seq bit
Sent acknowledgment: b'ACK 1'
------------------------------------------------



------------------------------------------------
Datagram received from ('172.21.33.2', 57242)
Seq bit: 0
Datagram: b'\x00Message no. 3 with length 512 bytes and seq_bit 0'

New data. Updating seq bit
Sent acknowledgment: b'ACK 0'
------------------------------------------------



------------------------------------------------
Datagram received from ('172.21.33.2', 57242)
Seq bit: 1
Datagram: b'\x01Message no. 4 with length 512 bytes and seq_bit 1'

New data. Updating seq bit
Sent acknowledgment: b'ACK 1'
------------------------------------------------



------------------------------------------------
Datagram received from ('172.21.33.2', 57242)
Seq bit: 0
Datagram: b'\x00Message no. 5 with length 512 bytes and seq_bit 0'

New data. Updating seq bit
Sent acknowledgment: b'ACK 0'
------------------------------------------------
```

Środowisko 2.

Klient

Niepoprawny bit weryfikujący

```
------------------------------------------------
Sending datagram #7848 (Seq: 1) to server 172.21.33.31:8080
Datagram: b'\x01Message no. 7848 with length 512 bytes and seq_bit 1'

Incorrect ACK received from server ('172.21.33.31', 8080)
Expected seq bit 1,but received 0
Retrying...
------------------------------------------------



------------------------------------------------
Sending datagram #7848 (Seq: 1) to server 172.21.33.31:8080
Datagram: b'\x01Message no. 7848 with length 512 bytes and seq_bit 1'

ACK received: Datagram #7848 acknowledged by server ('172.21.33.31', 8080)
------------------------------------------------
```

Przekroczenie czasu oczekiwania na potwierdzenie datagramu

```
------------------------------------------------
Sending datagram #7834 (Seq: 1) to server 172.21.33.31:8080
Datagram: b'\x01Message no. 7834 with length 512 bytes and seq_bit 1'

Timeout: No ACK received for datagram #7834
Retrying...
------------------------------------------------



------------------------------------------------
Sending datagram #7834 (Seq: 1) to server 172.21.33.31:8080
Datagram: b'\x01Message no. 7834 with length 512 bytes and seq_bit 1'

ACK received: Datagram #7834 acknowledged by server ('172.21.33.31', 8080)
------------------------------------------------
```

Serwer

Datagram zduplikowany lub poza kolejnością

```
------------------------------------------------
Datagram received from ('172.21.33.2', 52816)Seq bit: 1
Datagram: b'\x01Message no. 7854 with length 512 bytes and seq_bit 1'

New data. Updating seq bit
Sent acknowledgment: b'ACK 1'
------------------------------------------------



------------------------------------------------
Datagram received from ('172.21.33.2', 52816)Seq bit: 1
Datagram: b'\x01Message no. 7854 with length 512 bytes and seq_bit 1'

Data duplicated or out-of-order. Sending ACK again
Sent acknowledgment: b'ACK 1'
------------------------------------------------



------------------------------------------------
Datagram received from ('172.21.33.2', 52816)Seq bit: 0
Datagram: b'\x00Message no. 7855 with length 512 bytes and seq_bit 0'

New data. Updating seq bit
Sent acknowledgment: b'ACK 0'
------------------------------------------------
```

5. Wnioski

Dzięki zastosowaniu Alternating Bit Protocol zapewniono retransmisję utraconych danych, co wyeliminowało problem ich gubienia, znacząco zwiększając niezawodność transmisji. Ponadto, zastosowanie bitu sekwencji pozwala protokołowi skutecznie identyfikować i odrzucać duplikaty datagramów, co zapobiega ich wielokrotnemu przetwarzaniu.