PSI Zadanie 3

Paweł Spirydowicz, Hanna Zarzycka, Krzysztof Wojtaszko

4.12.2025 wersja 1

Treść zadania

Klient ma za zadanie odczytać plik z dysku (proszę wygenerować plik z losowymi 10000B) i wysłać do serwera jego zawartość w paczkach po 100B. Serwer ma zrekonstruować cały plik i obliczyć jego hash. Jako dowód działania proszę m.in. porównać hash obliczony przez serwer z hashem obliczonym przez klienta (może to być wydrukowane w konsoli klienta/serwera, hashe muszą być identyczne). Należy zaimplementować prosty protokół niezawodnej transmisji, uwzględniający możliwość gubienia datagramów. Gubione pakiety muszą być wykrywane i retransmitowane aby serwer mógł odtworzyć cały plik. Należy uruchomić program w środowisku symulującym błędy gubienia pakietów. (Informacja o tym, jak to zrobić znajduje się w skrypcie opisującym środowisko Dockera).

Rozwiązanie

Rozwiązanie składa się z dwóch komponentów, klienta i serwera, umieszczonych w kontenerach docker. Klient i serwer są podłączone do tego samego kontenera z siecią, dzięki czemu mogą przesyłać między sobą pakiety. Oba programy korzystają z tego samego algorytmu hashujacego – sha256. Paczki wysyłane do serwera mają dodane numery pozwalające na identyfikację pakietu, co pozwala na zagwarantowanie niezawodnej transmisji.

Klient python

Komunikacja UDP

```python
with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as sock:
    sock.settimeout(ACK_TIMEOUT)
    with open(FILE_PATH, "rb") as f:
        seq = 0
        while True:
            data = f.read(PACKET_SIZE)
            if not data:
                packet = struct.pack("!I", EOF)
                sock.sendto(packet, (server, port))
                break

            packet = struct.pack("!I", seq) + data
            retries = 0

            sock.sendto(packet, (server, port))
            print(f"[SEND] seq={seq}, bytes={len(data)}")

            while True:
                sock.sendto(packet, (server, port))
                try:
                    ack, _ = sock.recvfrom(4)
                    ack_seq = struct.unpack("!I", ack)[0]
                    print(f"[RECV ACK] {ack_seq}")

                    if ack_seq == seq:
                        print(f"[OK] Pakiet {seq} potwierdzony")
                        break
                    elif ack_seq < seq:
                        continue
                    else:
                        print(f"[BAD ACK] Otrzymano {ack_seq}, oczekiwano {seq}")

                except socket.timeout:
                    retries += 1
                    print(f"[TIMEOUT] Brak ACK dla {seq}, próba {retries}")
                    if retries > MAX_RETRIES:
                        print("[ERROR] Zbyt wiele retransmisji. Przerwano.")
                        return

            seq += 1
```

Paczki wysyłane do serwera są kawałkami pliku (10KB) o zadanej wielkości (100B) wysyłanymi do momentu w którym serwer potwierdzi odebranie paczki lub do momentu przekroczenia ilości dopuszczalnych prób.
Pakiety są wysyłane aż zostanie przesłany cały plik, a następnie wysyłany jest pakiet ze specjalną, wcześniej ustaloną wartością oznaczającą koniec pliku.
Ustawiony jest timeout, ponieważ bez tego serwer nie dawał sobie rady z ilością danych wysyłanych przez klienta (każdą paczkę musi odebrać i następnie na nią odpowiedzieć, a dodatkowo musi jeszcze zapisać zawartość paczki).

Serwer C

Komunikacja UDP

Przygotowanie serwera do nasłuchiwania

```c
//Open file in write mode
FILE *file = fopen(FILE_PATH, "wb");

if (argc != 2)
    errx(0, "Incorrect number of arguments. Correct usage: <port>", argv[0]);

//Create socket
if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
    bailout("Error opening socket\n");

server.sin_family = AF_INET;
server.sin_port = htons(atoi(argv[1]));
server.sin_addr.s_addr = INADDR_ANY;

//Bind socket to port
if ((s = bind(sock, (struct sockaddr *)&server, sizeof(server))) < 0)
    bailout("Bind unsuccessfull");

printf("[SERVER] Waiting for packets\n");
```

Odbiór pakietów

```c
uint32_t seq, seq_be;
char host[NI_MAXHOST], service[NI_MAXSERV];
uint8_t *data;
ssize_t data_size;

peer_addrlen = sizeof(peer_addr);
//Recieve packet
pkt_size = recvfrom(sock, buf, BUF_SIZE, 0,
                    (struct sockaddr *)&peer_addr, &peer_addrlen);
if (pkt_size < 4)
{
    fprintf(stderr, "Recvfrom not a valid package\n");
    continue;
}

s = getnameinfo((struct sockaddr *)&peer_addr, peer_addrlen, host, NI_MAXHOST,
                service, NI_MAXSERV, NI_NUMERICSERV);
if (s == 0)
{
    printf("[SERVER] Received %zd bytes from %s:%s\n", pkt_size, host, service);
    fflush(stdout);
}
else
{
    fprintf(stderr, "Error while getting client data\n");
    continue;
}
```

Wysyłanie pakietów

```
while(1)
    if (seq == count){
        data = buf + 4;
        data_size = pkt_size - 4;
        fwrite(data, 1, data_size, file);
        sendto(sock, &seq_be, sizeof(seq_be), 0, (struct sockaddr *)&peer_addr, peer_addrlen);
        printf("[SERVER] Send confirmation of recevieng %zd packet\n", seq);
        fflush(stdout);
        count = count + 1;
    }
    else if (seq < count)
    {
        sendto(sock, &seq_be, sizeof(seq_be), 0, (struct sockaddr *)&peer_addr, peer_addrlen);
        printf("[SERVER] Send confirmation of recevieng %zd packet\n", seq);
        fflush(stdout);
        continue;
    }
    else if (seq == EOF_UDP){
        printf("[SERVER] End of file\n");
        fflush(stdout);
        //Save file
        fflush(file);
        fclose(file);

        break;
    }
    else
    {
        bailout("Missing packets\n");
    }
}

//Get hash
unsigned char file_hash[SHA256_DIGEST_LENGTH];
hash(file_hash);
int i=0;
for(i=0; i < RETRIES; ++i){
    sendto(sock, &file_hash, sizeof(file_hash), 0, (struct sockaddr *)&peer_addr, peer_addrlen);
usleep(100000);
}
printf("[SERVER] ");
for(i=0; i<SHA256_DIGEST_LENGTH; ++i)
{
    //Print in hex
    printf("%02x", file_hash[i]);
}
printf("\n");
fflush(stdout);
```

Wysyłanie są albo wiadomości potwierdzające odbiór paczki, co zapewnia niezawodną transmisję, oraz na samym końcu jeden pakiet z wyliczonym hashem pliku, do porównania po stronie klienta z hashem pliku jaki przesyłał klient.
Serwer tutaj również zawartości paczek do pliku (odtwarza plik wysyłany przez klienta) oraz wylicza jego hash po zakończeniu transferu.

Konfiguracja docker

Serwer - z37_server

Klient - z37_client

Sieć - z37_network

Wywołanie programów

./app3-compose.sh

Testy

Testujemy program w środowisku symulującym błędy gubienia pakietów (30% pakietów jest gubionych) poprzez sprawdzenie czy serwer otrzymuje i odbiera wszystkie wymagane dane, co sprawdzane jest poprzez porównanie hashy pliku oryginalnego i powstałego po stronie serwera.

Konfiguracja

Port 8000

IP przydzielone automatycznie

Wydruki serwera i klienta

z37_server | [SERVER] Send confirmation of recevieng 92 packet
z37_server | [SERVER] Received 104 bytes from z37_client.cw3_default:59276
z37_server | [SERVER] Send confirmation of recevieng 92 packet
z37_client | [RECV ACK] 91
z37_server | [SERVER] Received 104 bytes from z37_client.cw3_default:59276
z37_server | [SERVER] Send confirmation of recevieng 92 packet
z37_client | [RECV ACK] 92 z37_client | [OK] Pakiet 92 potwierdzony
z37_client | [SEND] seq=93, bytes=100
z37_server | [SERVER] Received 104 bytes from z37_client.cw3_default:59276
z37_server | [SERVER] Send confirmation of recevieng 93 packet
z37_server | [SERVER] Received 104 bytes from z37_client.cw3_default:59276
z37_server | [SERVER] Send confirmation of recevieng 93 packet
z37_client | [RECV ACK] 93 z37_client | [OK] Pakiet 93 potwierdzony
z37_client | [SEND] seq=94, bytes=100
z37_server | [SERVER] Received 104 bytes from z37_client.cw3_default:59276
z37_server | [SERVER] Send confirmation of recevieng 94 packet
z37_server | [SERVER] Received 104 bytes from z37_client.cw3_default:59276
z37_server | [SERVER] Send confirmation of recevieng 94 packet
z37_client | [RECV ACK] 94 z37_client | [OK] Pakiet 94 potwierdzony

z37_client | [SEND] seq=95, bytes=100
z37_server | [SERVER] Received 104 bytes from z37_client.cw3_default:59276
z37_server | [SERVER] Send confirmation of recevieng 95 packet
z37_server | [SERVER] Received 104 bytes from z37_client.cw3_default:59276
z37_server | [SERVER] Send confirmation of recevieng 95 packet
z37_client | [RECV ACK] 95
z37_client | [OK] Pakiet 95 potwierdzony
z37_client | [SEND] seq=96, bytes=100
z37_server | [SERVER] Received 104 bytes from z37_client.cw3_default:59276
z37_server | [SERVER] Send confirmation of recevieng 96 packet
z37_server | [SERVER] Received 104 bytes from z37_client.cw3_default:59276
z37_server | [SERVER] Send confirmation of recevieng 96 packet
z37_client | [RECV ACK] 95
z37_server | [SERVER] Received 104 bytes from z37_client.cw3_default:59276
z37_server | [SERVER] Send confirmation of recevieng 96 packet
z37_client | [RECV ACK] 96
z37_client | [OK] Pakiet 96 potwierdzony
z37_client | [SEND] seq=97, bytes=100
z37_server | [SERVER] Received 104 bytes from z37_client.cw3_default:59276
z37_server | [SERVER] Send confirmation of recevieng 97 packet
z37_server | [SERVER] Received 104 bytes from z37_client.cw3_default:59276
z37_server | [SERVER] Send confirmation of recevieng 97 packet
z37_client | [RECV ACK] 96
z37_server | [SERVER] Received 104 bytes from z37_client.cw3_default:59276
z37_server | [SERVER] Send confirmation of recevieng 97 packet
z37_client | [RECV ACK] 97
z37_client | [OK] Pakiet 97 potwierdzony
z37_client | [SEND] seq=98, bytes=100
z37_server | [SERVER] Received 104 bytes from z37_client.cw3_default:59276
z37_server | [SERVER] Send confirmation of recevieng 98 packet
z37_server | [SERVER] Received 104 bytes from z37_client.cw3_default:59276
z37_server | [SERVER] Send confirmation of recevieng 98 packet
z37_client | [RECV ACK] 97
z37_server | [SERVER] Received 104 bytes from z37_client.cw3_default:59276
z37_server | [SERVER] Send confirmation of recevieng 98 packet
z37_client | [RECV ACK] 98
z37_client | [OK] Pakiet 98 potwierdzony
z37_client | [SEND] seq=99, bytes=100
z37_server | [SERVER] Received 104 bytes from z37_client.cw3_default:59276
z37_server | [SERVER] Send confirmation of recevieng 99 packet
z37_server | [SERVER] Received 104 bytes from z37_client.cw3_default:59276
z37_server | [SERVER] Send confirmation of recevieng 99 packet
z37_client | [RECV ACK] 98
z37_server | [SERVER] Received 104 bytes from z37_client.cw3_default:59276
z37_server | [SERVER] Send confirmation of recevieng 99 packet
z37_client | [RECV ACK] 99
z37_client | [OK] Pakiet 99 potwierdzony
z37_client | [CLIENT] Wszystkie pakiety wysłane, czekam na hash z serwera...
z37_server | [SERVER] Received 4 bytes from z37_client.cw3_default:59276
z37_server | [SERVER] End of file

z37_client | [CLIENT] Hash klienta :
a666d9ef994070ef7ba48948de4fadfff3f41ae6b1cc97f8d3a0658a899488e5
z37_client | [CLIENT] Hash serwera:
a666d9ef994070ef7ba48948de4fadfff3f41ae6b1cc97f8d3a0658a899488e5
z37_client | [RESULT] OK — hashe zgodne
z37_client exited with code 0
z37_server | [SERVER]
a666d9ef994070ef7ba48948de4fadfff3f41ae6b1cc97f8d3a0658a899488e5
z37_server exited with code 0

Ze względów na ilość danych drukowanych przez programy wstawiony został tylko końcówkę komunikacji pomiędzy serwerem a klientem.

Wnioski

Zagwarantowanie niezawodności transmisji wymaga znacznego zwiększenia nakładu zasobów, np. wielokrotnego wysyłanie tego samego pakietu, dodawanie identyfikatorów do pakietów i wysyłanie wiadomości zwrotnej o odebraniu pakietu.