PSI – Sprawozdanie z zadania 2 v1.0

Sofiya Yedzeika
Katsyaryna Anikevich
Matvii Ivashchenko

Treść zadania

Napisz zestaw dwóch programów – klienta i serwera komunikujących się poprzez TCP. Klient oraz serwer musi być napisany w konfiguracji C + Python (do wyboru co w czym). Zmodyfikować serwer tak, aby miał konstrukcję współbieżną, tj. obsługiwał każdego klienta w osobnym procesie. Dla C należy posłużyć się funkcjami fork() oraz (obowiązkowo) wait(). Dla Pythona należy posłużyć się wątkami, do wyboru: wariant podstawowy lub skorzystanie z ThreadPoolExecutor. Każdy połączony klient wysyła żądanie do serwera o obliczenie hasha przesłanej wiadomości. Przetestować dla kliku równolegle działających klientów.

Opis rozwiązania

- serwer TCP zaimplementowany w języku C,
- klient TCP zaimplementowany w języku Python.

Serwer

Serwer nasłuchuje na wszystkich interfejsach (`0.0.0.0`) na porcie `8000`. Po stronie serwera zastosowano model *process-per-connection*. Pętla główna akceptuje nowe połączenia, a następnie dla każdego wywoływana jest funkcja `fork()`:

- proces rodzicielski zamyka deskryptor nowego gniazda i wraca do `accept()`,
- proces potomny przejmuje obsługę pojedynczego klienta.

Dzięki temu kilku klientów może być obsługiwanych równolegle. W procesie potomnym wykonywane są następujące kroki:

1. Odczyt danych z gniazda do bufora bajtowego (`recv()`, maksymalnie 1023 bajty).
2. Dołączenie bajtu końca napisu `'\0'`.
3. Obliczenie wartości funkcji skrótu `djb2` dla odebranej wiadomości:

```
unsigned long djb2(unsigned char *str) {
    unsigned long hash = 5381;
    int c;

    while (c = *str++)
        hash = ((hash << 5) + hash) + c;

    return hash;
}
```

4. Konwersja wyniku na zapis szesnastkowy w postaci ASCII.

5. Odesłanie napisu zawierającego hash do klienta (`send()`) i zakończenie procesu.

Aby uniknąć powstawania procesów-zombie, zdefiniowano funkcję `KillZombies()` wywołującą `waitpid()` w trybie `WNOHANG` oraz powiązano ją z sygnałem `SIGCHLD`. Dodatkowo jest ona wywoływana w pętli głównej po zamknięciu gniazda klienta.

Klient

Funkcja `send_request()` realizuje pojedyncze żądanie do serwera: tworzy gniazdo TCP, łączy się z serwerem, wysyła napis w kodowaniu ASCII oraz odbiera odpowiedź zawierającą hash. Wątkowość zrealizowana jest przy pomocy `ThreadPoolExecutor` z maksymalną liczbą pięciu równolegle pracujących wątków.

Każdy wątek działa jak niezależny klient: nawiązuje własne połączenie TCP, wysyła jedną wiadomość i odbiera hash. Różne czasy życia wątków powodują, że serwer w C w danym momencie obsługuje kilka procesów potomnych.

Opis konfiguracji testowej

Kontener serwera

- Nazwa kontenera: `z33_server`
- Adres IP w sieci Docker: `172.21.33.2`
- Port TCP: `8000`

Kontener klienta

- Nazwa kontenera: `z33_client`
- Dołączenie do sieci: `--network z33_network`

Opis testowania i wydruków

Najpierw program drukuję komunikaty o zebraniu dokerów, później wyniki:

1. Informacja o adresie i porcie serwera, np.
   `Client will send to 172.21.33.2:8000`
2. Dla każdej wiadomości wynik w postaci:
   `Message: <wiadomość> : Hash: <wartość_hex>`

```
Client will send to 172.21.33.2:8000
Message: PSI            : Hash: b881231
Message: Anikevich      : Hash: 37798eb9d9fc677
Message: Katsyaryna     : Hash: 726c4c4a4b87076c
Message: Yedzeika       : Hash: 1ae685ef6cffdb
Message: Sofiya         : Hash: 652d18d8e30
Message: Matvii         : Hash: 652c29925ef
Message: Ivashchenko    : Hash: bfee51c2489c6c98
z33_server
```

Poprawność działania weryfikowano w następujący sposób:

- każda wiadomość powinna zostać obsłużona bez przekroczenia czasu,
- hash zwracany przez serwer miał stałą wartość dla danego napisu w kolejnych uruchomieniach,
- serwer po zakończeniu pracy nie pozostawiał procesów-zombie.