Sprawozdanie z Laboratorium PSI
Zadanie 2 – Komunikacja TCP (Serwer Współbieżny)

Zespół nr 39
Skład zespołu:
Łukasz Szydlik (lider)
Jan Szwagierczak
Dominik Śledziewski

26 listopada 2025

1 Treść zadania

Celem zadania było stworzenie zestawu dwóch programów (klienta i serwera) komunikujących się poprzez protokół TCP. Zgodnie z wymaganiami :

- Serwer został napisany w języku **C**, a klient w języku **Python**.
- Serwer ma konstrukcję współbieżną - obsługuje każdego klienta w osobnym procesie przy użyciu funkcji `fork()`.
- Wymagane jest poprawne sprzątanie procesów potomnych przy użyciu funkcji `wait()`.
- Klient wysyła wiadomość, a serwer odsyła jej skrót (hash), który klient następnie weryfikuje.

2 Opis rozwiązania

2.1 Architektura systemu

System składa się z dwóch obrazów Docker uruchamianych w dedykowanej sieci `z39_network`.

- **Serwer (C):** Nasłuchuje na porcie 8000. Główny proces w pętli akceptuje połączenia (`accept`) i dla każdego nowego klienta tworzy proces potomny (`fork`).
- **Klient (Python):** Łączy się z serwerem, wysyła ciąg znaków, odbiera obliczony hash i porównuje go z lokalnie obliczoną wartością (algorytm DJB2). Dodatkowo klient symuluje opóźnienie (`sleep(5)`), aby udowodnić współbieżność serwera.

2.2 Implementacja współbieżności (Serwer C)

Kluczowym elementem zadania była obsługa wielu klientów naraz. Zrealizowano to poprzez funkcję systemową `fork()`. Zaimlementowano usuwanie procesów "zombie".

Fragment kodu serwera odpowiedzialny za `fork` i `waitpid`:

```c
// Sprzatanie procesow zombie
void reap_children(int sig) {
    (void)sig;
    int status;
    pid_t pid;

    while ((pid = wait(&status)) > 0) {
        (void)pid;
    }
}

// Petla glowna serwera
while (1) {
    // ...
    int connfd = accept(sockfd, (struct sockaddr *)&cliaddr, &len);
    // ...
    pid_t pid = fork();
    // ..
    if (pid == 0) {
        /* PROCES POTOMNY */
        close(sockfd); // Zamkniecie gniazda nasluchujacego
        // Obsluga klienta: recv -> compute_hash -> send
        close(connfd);
        exit(EXIT_SUCCESS);
    } else {
        /* PROCES RODZICA */
        close(connfd); // Zamkniecie deskryptora klienta
    }
}
```

Listing 1: Obsługa procesów potomnych i sygnałów

2.3 Algorytm haszujący

Zastosowano algorytm DJB2. Jest to prosty i szybki algorytm haszujący, łatwy do zaimplementowania w obu językach w celu weryfikacji poprawności danych.

3 Konfiguracja testowa

Testy przeprowadzono w środowisku Docker na serwerze `bigubu`.

- **Sieć:** `z39_network` (bridge).
- **Serwer:** Kontener `z39_server_tcp`.
- **Klienci:** 5 instancji uruchamianych równolegle w tle przez skrypt powłoki.

4 Przebieg testów i wyniki

Przeprowadzono test polegający na równoległym uruchomieniu 5 klientów. Każdy klient łączy się z serwerem, a następnie usypia na 5 sekund przed zakończeniem transmisji. Gdyby serwer był iteracyjny (jednowątkowy), obsługa 5 klientów zajęłaby 25 sekund ($5 \times 5s$). Dzięki zastosowaniu `fork()`, serwer obsługuje ich równolegle.

4.1 Logi z działania (Client Side)

Poniżej przedstawiono logi z uruchomienia skryptu testowego . Widać, że 5 klientów uruchomiło się niemal jednocześnie i wszyscy otrzymali poprawny hash.

```
--- Running 5 client in parallel ---
[Client] Connecting to z39_server_tcp:8000...
[Client] Simulating 'slow client'. Sleeping for 5s...
[Client] Connected.
[Client] Simulating 'slow client'. Sleeping for 5s...
...
[Client] Resuming work after sleep.
[Client] Sending: 'Test PSI Message'
[Client] Server response: Hash: 10907319353902606518
[Test A] SUCCESS: Hashes are identical.
...
[Test A] SUCCESS: Hashes are identical.
```

Listing 2: Logi klientów (skrócone)

Wszystkie 5 instancji klienckich zakończyło test z sukcesem ("Hashes are identical"), co potwierdza poprawność przesyłania danych i obliczeń.

4.2 Logi z działania (Server Side)

Logi serwera potwierdzają nawiązanie połączeń z 5 różnymi adresami IP w bardzo krótkim odstępie czasu (nie czekając na rozłączenie poprzedniego klienta).

```
TCP server listens on 8000
New connection from 172.21.39.4:58514
New connection from 172.21.39.5:40292
New connection from 172.21.39.6:49554
New connection from 172.21.39.7:55190
New connection from 172.21.39.8:51184
...
Sent response 'Hash: 10907319353902606518' to 172.21.39.4:58514
Sent response 'Hash: 10907319353902606518' to 172.21.39.5:40292
```

Listing 3: Logi serwera

5 Wnioski

Zadanie zostało wykonane poprawnie.

1. Implementacja serwera w C poprawnie wykorzystuje funkcję `fork()` do tworzenia nowych procesów dla każdego połączenia przychodzącego.
2. Serwer skutecznie usuwa procesu "zombie", co jest kluczowe dla stabilności długotrwałego działania.
3. Testy wykazały, że serwer nie blokuje się podczas obsługi klienta, który symuluje opóźnienie.
4. Weryfikacja hasha DJB2 potwierdziła integralność przesyłanych danych między Pythonem a C.