Výpočet bodov - OSRT

Obsah

Úvod

Tento projekt predstavuje implementáciu servera a klienta, ktorý si vymieňa súradnice bodov a vypočíta vzdialenosť medzi nimi. Používa multithreading na spracovanie prichádzajúcich požiadaviek a zabezpečuje komunikáciu prostredníctvom socketov.

Funkcionalita

- Server: Počúva na prichádzajúce spojenia, prijíma súradnice bodov od klienta a vypočíta vzdialenosť medzi nimi.
- Klient: Odosiela súradnice bodov na server a prijíma výslednú vzdialenosť.
- Používa sa alarm na zabezpečenie správneho načítania vstupu.
- Implementované sú mutexy na synchronizáciu prístupu k zdieľaným premenným.

Inštalácia

Ako prvé treba si nainštalovať Linux distribúciu. Počas vypracovania bolo použité WSL Ubuntu z Microsoft Store

---

Ak toto máme splnené tak môžeme pokračovať nasledovne: 1. **Nainštalujeme GCC (GNU Compiler Collection)**, ak ho ešte nemáme nainštalovaný. Na Ubuntu môžeme spustiť nasledujúce príkazy: `bash sudo apt-get update sudo apt install gcc sudo apt install xterm`

2. **Naklonujeme repozitár** s kódom:

   ```
   git clone Gymoblig/OSRT_Semestralka
   ```

3. **Prejdeme do adresára projektu**:

   ```
   cd OSRT_Semestralka-main
   ```

Spustenie

1. Skontrolujeme a nastavíme premenné pre názvy xterm okien a aj súborov v súbore `Makefile`: `bash SERVER = Server CLIENT = Client`
2. Skontrolujeme nastavenie kompilátora **GCC** a aj jeho parametrov: `bash CC = gcc CFLAGS_SERVER = -lpthread -lm CFLAGS_CLIENT =`
3. Spustíme kompiláciu: `bash make` alebo môžeme rovno zkompilovať a spustiť server a následne klienta: `bash make spustit` — *Pri použití* ***`make clean`*** *sa vymaže zkompilovaný server a client*

Ovládanie

- Po spustení pomocou príkazu `make spustit` v okne xterm Client môžeme zadať súradnice.
- Klient odošle súradnice na server, ktorý vypočíta vzdialenosť a zobrazí ju vspäť v xterm klienta.
- Pre zastavenie servera a ukončenie programu stačí stlačiť `CTRL + C`.

Prvky

Najskôr si client vypýta od používateľa súradnice bodu 1 a bodu 2. Tieto procesy sú urobené cez fork a parametre sú ukladané pomcou pupefd1 a pipefd2.

```
pid_t pid1 = fork();
    if (pid1 == 0) { // Proces pre Bod 1
        nacitajFloat("Zadaj X pre bod 1: ", &bod1.X);
        nacitajFloat("Zadaj Y pre bod 1: ", &bod1.Y);

        // Zápis do pipe
        close(pipefd1[0]); // Zavrieť čítaciu časť
        write(pipefd1[1], &bod1, sizeof(Bod));
        close(pipefd1[1]); // Zavrieť zápisovú časť

        exit(0);
    }
    wait(NULL); // Čakanei na koniec procesu

    pid_t pid2 = fork();
    if (pid2 == 0) { // Proces pre Bod 2
        nacitajFloat("Zadaj X pre bod 2: ", &bod2.X);
        nacitajFloat("Zadaj Y pre bod 2: ", &bod2.Y);

        // Zápis do pipe
        close(pipefd2[0]); // Zavrieť čítaciu časť
        write(pipefd2[1], &bod2, sizeof(Bod));
        close(pipefd2[1]); // Zavrieť zápisovú časť
```

```
        exit(0);
    }
```

Následne rodič zoberie parametre z oboch rúr a vytvorí ďalšie deti pomocou **fork(0)** a pošle funkciou posliNaServerBod1 a posliNaServerBod2 pomocou socketov parametre na **server.c**

```
// Rodič číta hodnoty z pipes
    close(pipefd1[1]); // Zavrieť zápisovú časť pre Bod 1
    read(pipefd1[0], &bod1, sizeof(Bod));
    close(pipefd1[0]); // Zavrieť čítaciu časť pre Bod 1

    close(pipefd2[1]); // Zavrieť zápisovú časť pre Bod 2
    read(pipefd2[0], &bod2, sizeof(Bod));
    close(pipefd2[0]); // Zavrieť čítaciu časť pre Bod 2

    // Poslanie dát na server
    pid_t sendpid1 = fork();
    if (sendpid1 == 0) {
        posliNaServerBod1(bod1); // Odošle Bod 1
        exit(0);
    }
    wait(NULL);

    pid_t sendpid2 = fork();
    if (sendpid2 == 0) {
        posliNaServerBod2(bod2); // Odošle Bod 2
        exit(0);
    }

    wait(NULL); // Čakanie na koniec procesu
```

Pri funckii posliNaServerBod1() sa vytvorí socket, potom sa pomocou AF_INET určí family a dá sa adresa servera spolu so základným portom **7777**. Nasleduje pripojenie na server a potom už ostáva len odoslať súradnice na **server.c**, celá funckia sa ukončí uzatvorením socketu

```
void posliNaServerBod1(Bod bod) {
    int sock_desc = socket(AF_INET, SOCK_STREAM, 0); // Vytvorenie socketu
    if (sock_desc == -1) {
        printf("Nemôžem vytvoriť socket pre Bod 1!\n");
        exit(1);
    }

    struct sockaddr_in server;
    memset(&server, 0, sizeof(server));
    server.sin_family = AF_INET;
```

```
    server.sin_addr.s_addr = inet_addr("127.0.0.1");
    server.sin_port = htons(PORT);

    if (connect(sock_desc, (struct sockaddr*)&server, sizeof(server)) != 0) {
        printf("Nemôžem sa pripojiť k serveru pre Bod 1!\n");
        close(sock_desc);
        exit(1);
    }

    // Odošle súradnice bodu
    float data[2] = {bod.X, bod.Y};
    send(sock_desc, data, sizeof(data), 0);
    sleep(1);
    close(sock_desc); // Zatvorenie socketu
}
```

Prijatie bodu na serveri sa uskutočňuje pomocou funkcie **prijmiBod()**. Tu sú využité vlákna kde z **recv()** sa zoberú parametre a pomocou zamknutia mutexu sa následne zistí či do bodu1 už boli zadané parametre. Ak neboli zapíšu sa do bodu1, ak informácie v bode1 už sú tak sa zapíšu do bodu2. Prebehne odomknutie mutexu a uzavretie dočasného socketu. Už sa len vypíše, že vlákno dokončilo prácu.

```
// Vlákno na prijímanie bodu
void* prijmiBod(void* sock_desc) {
    int temp_sock_desc = *(int*)sock_desc; // Dočasný socket
    float bod[2]; // Pre prijatý bod

    // Prijatie bodu
    recv(temp_sock_desc, bod, sizeof(bod), 0);

    pthread_mutex_lock(&mutex); // Zamykanie mutexu

    // Uloženie prijatého bodu
    if (bod1[0] == 0 && bod1[1] == 0) {
        bod1[0] = bod[0];
        bod1[1] = bod[1];
        printf("Prijatý bod 1: X = %.2f, Y = %.2f\n", bod1[0], bod1[1]);
    } else {
        bod2[0] = bod[0];
        bod2[1] = bod[1];
        printf("Prijatý bod 2: X = %.2f, Y = %.2f\n", bod2[0], bod2[1]);
    }

    pthread_mutex_unlock(&mutex); // Odomykanie mutexu

    close(temp_sock_desc); // Zatvorenie socketu
```

```
    printf("→ Vlákno na prijatie bodu dokončilo prácu.\n");

    return NULL; // Návrat z vlákna
}
```

V **main()** prebehne vytvorenie socketu a nastavenie servera. Keď sa prijatie bodov dokončí tak sa pomocou vypocitajVzdialenost() vypočíta vzdialenosť a server ju vypíše. Keď ju server zistí pošle ju späť klientovi.

Pri **client.c** ešte sú **signalik** a **nacitajFloat()**. - Kde handle_alarm() slúži pre funkcionalitu časovača kde pri nacitajFloat() ak používateľ do 10 sekúnd nedá parameter tak to vypíše znova žiadosť o súradnicu. - handle_sigint slúži na opätovné využitie signálu. Využitie spočíva pri stlačení kláves 'CTRL+C' kde sa zabijú všetky okná xterm, keďže sa **client.c** a **server.c** otvárajú v xterm okne čo je spúšťané cez Makefile.

Definovanie signalu pre sigint v main()

```
 signal(SIGINT, signalik); // Spracovanie Ctrl+C
```

Časovač:

```
// Funkcia na obsluhu alarmu
void timer_handler(int sig)
{
    printf("\nProsím zadaj súradnicu: ");
    fflush(stdout);
}

timer_t nastav_casovac() {
    struct sigevent sev;
    sev.sigev_notify = SIGEV_SIGNAL;
    sev.sigev_signo = SIGALRM; // Signál pri uplynutí času

    timer_t timer;
    timer_create(CLOCK_REALTIME, &sev, &timer);
    return timer;
}

void spustiCasovac(timer_t casovac, int sekundy)
{
  struct itimerspec casik;
  casik.it_value.tv_sec=sekundy;
  casik.it_value.tv_nsec=0;
  casik.it_interval.tv_sec=sekundy;
  casik.it_interval.tv_nsec=0;
  timer_settime(casovac,0,&casik,NULL);
}
```

signalik()

```
// Funkcia na obsluhu Ctrl+C
void signalik(int sig) {
    //Kontrola či deti existujú, ak áno zabiť SIGKILL
    if (pid1 > 0) {
        kill(pid1, SIGKILL);
    }
    if (pid2 > 0) {
        kill(pid2, SIGKILL);
    }
    system("pkill xterm"); // Zatvoriť terminál, keď je Ctrl+C stlačené
    exit(0);
}
```

nacitajFloat()

```
// Funkcia na načítanie float hodnoty
void nacitajFloat(const char* prompt, float* value) {
    char input[128];
    signal(SIGALRM, timer_handler); // Nastavenie signálu pre časovač

    // Nastavenie časovača
    timer_t timer = nastav_casovac();
    while (1) {
        spustiCasovac(timer, CAS);
        printf("%s", prompt);
        fflush(stdout); // Uistiť sa, že prompt sa zobrazí

        // Čítanie vstupu
        if (fgets(input, sizeof(input), stdin) != NULL) {

            // Pokúsiť sa previesť na float
            if (sscanf(input, "%f", value) == 1) {
                break; // Validný vstup
            } else {
                printf("Neplatný vstup! Skús to znova.\n");
                fflush(stdout);
            }
        } else {
            printf("Chyba pri načítaní vstupu.\n");
        }
    }
    spustiCasovac(timer, 0);
}
```