Wielowątkowy serwer TCP

Wątek

- współbieżne działanie
- współdzielenie danych
- wykonywanie tego samego programu
- tańsze tworzenie w porównaniu do fork()
- join(), a detach()

pthread - interesujące fragmenty

- pthread_create(), pthread_detach(), pthread_exit(), pthread_join()
- pthread_mutex_t
  - PTHREAD_MUTEX_INITIALIZER
  - pthread_mutex_init() / pthread_mutex_destroy()
  - pthread_mutex_lock() / pthread_mutex_unlock()
- pthread_cond_t
  - PTHREAD_COND_INITIALIZER
  - pthread_cond_init() / pthread_cond_destroy()
  - pthread_cond_wait() / pthread_cond_signal() / pthread_cond_broadcast()

Współbieżny serwer TCP - schemat

```
socket()
setsockopt()
bind()
listen()

while() {
        accept()
        args = ...
        pthread_create(f_watku, args)
}
```

```
f_watku(args) {
        pthread_detach()
}
```

Problemy

Poprawne wykonanie programu != poprawny program

(brak objawów != brak choroby)

Problemy - zarządzanie pamięcią

```
funkcja() {
        int *licznik = malloc(...);
        pthread_create(..., f_watku, licznik)
}

f_watku(int *licznik) {
        printf(“%d”, *licznik)
        …
        pthread_exit(...)
}
```

Problemy - zarządzanie pamięcią

```
funkcja() {
        int *licznik = malloc(...);
        pthread_create(..., f_watku, licznik)
}

f_watku(int *licznik) {
        printf(“%d”, *licznik)
        …
        free(licznik);
        pthread_exit(...)
}
```

Problemy - warunek wyścigu

```
funkcja() {
        int *liczba = malloc(...)

        *liczba = 0
        pthread_create(..., f_watku, liczba)

        *liczba = 1
        pthread_create(..., f_watku2, liczba)
}
```

```
f_watku(int *liczba) {
        printf(“f: %d”, *liczba)
}

f_watku2(int *liczba) {
        printf(“f2: %d”, *liczba)
}
```

Problemy - warunek wyścigu

```
funkcja() {
    int *liczba = malloc(...)

    *liczba = 0
    pthread_create(..., f_watku, liczba)

    int *liczba2 = malloc(...)
    *liczba2 = 1
    pthread_create(..., f_watku2, liczba2)
}
```

```
f_watku(int *liczba) {
    printf("f: %d", *liczba)
}

f_watku2(int *liczba) {
    printf("f2: %d", *liczba)
}
```

Problemy - współbieżna modyfikacja

```
funkcja() {
        int *liczba = malloc(...)
        *liczba = 0
        pthread_create(..., f_watku, liczba)
        pthread_create(..., f_watku2, liczba)
}

zwieksz(int *liczba, int wartosc) {
        *liczba = *liczba + wartosc
}
```

```
f_watku(int *liczba) {
        zwieksz(liczba, 10)
}

f_watku2(int *liczba) {
        zwieksz(liczba, 20)
}
```

Problemy - współbieżna modyfikacja (zamek)

```
funkcja() {
        int *liczba = malloc(...)
        *liczba = 0
        pthread_create(..., f_watku, liczba)
        pthread_create(..., f_watku2, liczba)
}

zwieksz(int *liczba, int wartosc) {
        mutex_lock
        *liczba = *liczba + wartosc
        mutex_unlock
}
```

```
f_watku(int *liczba) {
        zwieksz(liczba, 10)
}

f_watku2(int *liczba) {
        zwieksz(liczba, 20)
}
```

Problemy - współbieżna modyfikacja (v. TCP)

```
funkcja() {
        int *desc = malloc(...)
        *desc = socket()
        pthread_create(..., f_watku, desc)
        pthread_create(..., f_watku2, desc)
}

wyslij(int *desc, ...) {
        write(desc, …)
}
```

```
f_watku(int *desc) {
        wyslij(desc, “msg1;”)
}

f_watku2(int *desc) {
        wyslij(desc, “msg2;”)
}
```

Problemy - współbieżna modyfikacja (v. TCP)

```
funkcja() {
        int *desc = malloc(...)
        *desc = socket()
        pthread_create(..., f_watku, desc)
        pthread_create(..., f_watku2, desc)
}

wyslij(int *desc, ...) {
        mutex_lock
        write(desc, …)
        mutex_unlock
}
```

```
f_watku(int *desc) {
        wyslij(desc, “msg1;”)
}

f_watku2(int *desc) {
        wyslij(desc, “msg2;”)
}
```

Problemy - granularność zamków

```
pthread_mutex_t m = PTHREAD...

funkcja() {
        int *liczba1 = malloc(...)
        *liczba1 = 10
        int *liczba2 = malloc(...)
        *liczba2 = 20
        pthread_create(..., f_watku, liczba1)
        pthread_create(..., f_watku, liczba1)
        pthread_create(..., f_watku2, liczba2)
        pthread_create(..., f_watku2, liczba2)
}
```

```
f_watku(int *liczba1) {
        mutex_lock(&m)
        *liczba1 = *liczba1 - 12
        mutex_unlock(&m)
}

f_watku2(int *liczba2) {
        mutex_lock(&m)
        *liczba2 = *liczba2 - 7
        mutex_unlock(&m)
}
```

Problemy - granularność zamków

```
pthread_mutex_t m1 = PTHREAD...
pthread_mutex_t m2 = PTHREAD...

funkcja() {
        int *liczba1 = malloc(...)
        *liczba1 = 10
        int *liczba2 = malloc(...)
        *liczba2 = 20
        pthread_create(..., f_watku, liczba1)
        pthread_create(..., f_watku, liczba1)
        pthread_create(..., f_watku2, liczba2)
        pthread_create(..., f_watku2, liczba2)
}
```

```
f_watku(int *liczba1) {
        mutex_lock(&m1)
        *liczba1 = *liczba1 - 12
        mutex_unlock(&m1)
}

f_watku2(int *liczba2) {
        mutex_lock(&m2)
        *liczba2 = *liczba2 - 7
        mutex_unlock(&m2)
}
```

Problemy - aktywne czekanie

pthread_mutex_t m = PTHREAD…

```
funkcja() {
        int *produkty = malloc(...)
        *produkty = 0
        pthread_create(..., f_watku, produkty)
        pthread_create(..., f_watku2, produkty)
}
f_watku(int *produkty) {
  while(1 == 1) {
        mutex_lock(&m)
        *produkty = *produkty + 1
        mutex_unlock(&m)
  } }
```

```
f_watku2(int *produkty) {
  while(1 == 1) {
        mutex_lock(&m)
        if (*produkty > 0) {
                *produkty = *produkty - 1
        }
        mutex_unlock(&m)
  }
}
```

Problemy - aktywne czekanie (zmienna warunkowa)

```
pthread_mutex_t m = PTHREAD…
pthread_cond_t c = PTHREAD...
funkcja() {
        int *produkty = malloc(...)
        *produkty = 0
        pthread_create(..., f_watku, produkty)
        pthread_create(..., f_watku2, produkty)
}
f_watku(int *produkty) {
  while(1 == 1) {
        mutex_lock(&m)
        *produkty = *produkty + 1
        mutex_unlock(&m)
        pthread_cond_signal(&c)
  } }
```

```
f_watku2(int *produkty) {
  while(1 == 1) {
        mutex_lock(&m)
        while (*produkty <= 0) {
          pthread_cond_wait(&c, &m)
        }
        *produkty = *produkty - 1
        mutex_unlock(&m)
  }
}
```