Politechnika Białostocka

Sprawozdanie z pracowni specjalistycznej

Systemy operacyjne

Projekt numer: **2**

Temat: **Problem śpiącego fryzjera.**

Wykonujący ćwiczenie: **Karol Budlewski**
Maciej Więckowski

Studia dzienne

Kierunek: Informatyka

Semestr: IV

Grupa zajęciowa: PS4

Prowadzący ćwiczenie: **mgr inż. Daniel Reska**

.....
OCENA

Data
8 czerwca 2017

.....
Data i podpis prowadzącego

1 Wstęp

Celem projektu było rozwiązanie problemu fryzjera realizowanego na wątkach oraz synchronizowanego semaforami, mutexami i zmiennymi warunkowymi.
Program zrealizowany jest w wariantach:

a) z użyciem wyłącznie semaforów i mutexów.

b) z użyciem zmiennych warunkowych.

Oczekiwana liczba punktów to `34`.

2 Instalacja

Program jest instalowany z linii komend. Należy przejść do katalogu z plikiem `Makefile` i uruchomić komendę `make`. Po tym mamy już skompilowany program, który mieści się w katalogu `build`.

3 Sposób uruchamiania

Program jest uruchamiany z linii komend i może przyjmować następujące parametry:

a) `-t` - określa czas w `mikrosekundach` z jakim będą się pojawiać nowe wątki `Klient`

b) `-ts` - określa czas w `sekundach` z jakim będą się pojawiać nowe wątki `Klient`

c) `-tms` - określa czas w `milisekundach` z jakim będą się pojawiać nowe wątki `Klient`

d) `-size` - określa wielkość poczekalni

e) `-debug` - uruchamia tryb `debug`, wypisujący całą kolejkę klientów czekających oraz tych którzy nie dostali się do gabinetu

Do uruchomienia programu nie wymagane są żadne parametry. Program uruchomi się wówczas z ustawieniami domyślnymi, czyli czas ustawiony na 75 ms, rozmiar poczekalni na 10 oraz bez debuga. Parametry `-t`, `-ts`, `-tms` wpływają na siebie i zawsze będzie brany ostatni pod uwagę oraz jeżeli zostanie popełniony błąd w którym kolwiek z opcji program nie zostanie uruchomiony.

Przykłady uruchamiania programu

- ./build/fryzjer -t 150
- ./build/fryzjer -ts 150
- ./build/fryzjer -tms 150
- ./build/fryzjer -ts 150 -debug
- ./build/fryzjer -tms 150 -size 10
- ./build/fryzjer -t 150 -size 10 -debug

4 Zaimplementowane funkcje

4.1 Ogólne założenia funkcji

W obu programach występują te same funkcje:

1. `customer`,
2. `barber`,
3. `parser`,
4. `createCustomer`,
5. `state`,
6. `semaphoreInit`.

Funckje `parser`, `createCustomer`, `state`, `semaphoreInit` wykonują ten sam kod. Dodatkowo została zaimplementowana kolejka FIFO, która jest używana również jak zwykła lista.

4.1.1 parser

Jest to funkcja do sprawdzania użytych opcji. Może spowodować wykonanie się programu na domyślnych ustawieniach w przypadku błędnego użycia opcja.
np. po opcji `-size` nie zostanie podana żadna liczba.

4.1.2 state

Jest to funkcja do wyświetlania na ekranie aktualnego stanu salonu fryzjerskiego. Gdy w programie jest włączona opcja `debug` to ta funkcja powoduje również wypisanie na ekranie listy klientów, którzy zrezygnowali oraz przebywających w poczekalni.

4.1.3 semaphoreInit

Jest to funkcja do inicjalizacji semaforów.

4.1.4 createCustomer

Jest to funkcja do stworzenia wątku `Klienta`.

Pozostałe funkcje się różnią od wersji programu i ze względu na użycie synchronizacji zostano szczegółowo przedstawione.

4.2 Wersja bez użycia zmiennych warunkowych

4.2.1 Wątek klienta

```
void customer(void* arg) {
    // Zablokowanie mutexu odpowiedzialnego za wydarzeniami w poczekalni
    // Klient wszedl do poczekalni
    pthread_mutex_lock(&mutex_poczekalnia);
    // Klient pobiera numerek
    int id = licznik++;
    // Klient sprawdza czy jest wystarczajaca ilosc miejsc w poczekalni
    if(sem_trywait(&poczekalnia) == 0){
        // Klientowi udalo sie wejsc do poczekalni i zajac miejsce
        enqueue(oczekujacy_klienci, id, NULL);
        state();
        // Klient sygnalizuje fryzjerowi ze jest gotowy do strzyzenia
        // poprzez wyslanie do niego sygnalu
        sem_post(&budzenie);
        // Klient zostal zaproszony do strzyzenia
        // Mutex poczekalni zostal zwolniony
        pthread_mutex_unlock(&mutex_poczekalnia);

        // Klient wszedl do gabinetu fryzjera
        // Mutex gabinetu zostaje zablokowany
        pthread_mutex_lock(&mutex_gabinet);
        // Klient idzie na strzyzenie
        sem_wait(&strzyzenie_klienta);
        // Zwolnione zostaje miejsce w poczekalni
        sem_post(&poczekalnia);
        klient = id;
        oczekujacy_klienci = dequeue(oczekujacy_klienci);
        state();
        usleep(100000);
        // Klient zostaje ostrzyzony
        sem_post(&zajetosc_fryzjera);
        klient = -1;
        state();
        // Klient wychodzi od fryzjera
        // Mutex gabinetu zostaje odblokowany
        pthread_mutex_unlock(&mutex_gabinet);
    }
    else {
        // Klientowi nie udalo sie zajac miejsca wiec zrezygnowal z wizyty
        // u fryzjera
        ++zrezygnowani;
        enqueue(zrezygnowani_klienci, id, NULL);
        state();
        // Mutex poczekalni zostal zwolniony po wyjsciu Klienta
        pthread_mutex_unlock(&mutex_poczekalnia);
    }
}

```

4.2.2 Wątek fryzjera

```
void barber(void* arg) {
    // Fryzjer pracuje caly czas
    while(1) {
        // Fryzjer zostaje powiadomiony  o tym ze jest klient
        sem_wait(&budzenie);
        // Fryzjer bierze Klienta z poczekalni
        // Mutex poczekalni zostaje zablokowny
        pthread_mutex_lock(&mutex_poczekalnia);
        // Klient zostaje ostrzyzony
        sem_post(&strzyzenie_klienta);
        // Fryzjer wypuszcza Klienta
        // Mutex zostaje odblokowny
        pthread_mutex_unlock(&mutex_poczekalnia);
        // Fryzjer mowi ze jest wolny
        sem_wait(&zajetosc_fryzjera);
        usleep(100000);
    }
}

```

4.3 Wersja z użyciem zmiennych warunkowych

4.3.1 Wątek fryzjera

```
void barber(void* arg) {
    // Fryzjer pracuje caly czas
    while(1) {
        // Fryzjer zostaje powiadomiony  o tym ze jest klient
        sem_wait(&budzenie);
        // Fryzjer bierze Klienta z poczekalni
        // Mutex poczekalni zostaje zablokowny
        pthread_mutex_lock(&mutex_poczekalnia);
        pthread_cond_t *zmienna = oczekujacy_klienci->next->zmienna;
        // Klient zostaje ostrzyzony
        // Wysyla sygnal przez zmienna warunkowa
        pthread_cond_signal(zmienna);
        // Fryzjer wypuszcza Klienta
        // Mutex zostaje odblokowny
        pthread_mutex_unlock(&mutex_poczekalnia);
        // Fryzjer mowi ze jest wolny
        sem_wait(&zajetosc_fryzjera);
        usleep(100000);
    }
}

```

4.3.2 Wątek klienta

```
void customer(void* arg) {
    // Zablokowanie mutexu odpowiedzialnego za wydarzeniami w poczekalni
    // Klient wszedl do poczekalni
    pthread_mutex_lock(&mutex_poczekalnia);
    // Klient pobiera numerek
    int id = licznik++;
    // Przypisanie zmiennej warunkowej do klienta
    pthread_cond_t *cond_klient = malloc(sizeof(pthread_cond_t));
    pthread_cond_init(cond_klient, NULL);
    // Klient sprawdza czy jest wystarczajaca ilosc miejsc w poczekalni
    if(sem_trywait(&poczekalnia) == 0){
        // Klientowi udalo sie wejsc do poczekalni i zajac miejsce
        enqueue(oczekujacy_klienci, id, cond_klient);
        state();
        // Klient sygnalizuje fryzjerowi ze jest gotowy do strzyzenia
        // poprzez wyslanie do niego sygnalu
        sem_post(&budzenie);
        // Klient czeka na zaproszenie do strzyzenia
        pthread_cond_wait(cond_klient, &mutex_poczekalnia);
        // Zwolnione zostaje miejsce w poczekalni
        sem_post(&poczekalnia);
        // Klient zostal zaproszony do strzyzenia
        // Mutex poczekalni zostal zwolniony
        pthread_mutex_unlock(&mutex_poczekalnia);

        // Klient wszedl do gabinetu fryzjera
        // Mutex gabinetu zostaje zablokowany
        pthread_mutex_lock(&mutex_gabinet);
        klient = id;
        oczekujacy_klienci = dequeue(oczekujacy_klienci);
        state();
        usleep(100000);
        // Klient zostaje ostrzyzony
        sem_post(&zajetosc_fryzjera);
        klient = -1;
        state();
        // Klient wychodzi od fryzjera
        // Mutex gabinetu zostaje odblokowany
        pthread_mutex_unlock(&mutex_gabinet);
    }
    else {
        // Klientowi nie udalo sie zajac miejsca wiec zrezygnowal z wizyty
        // u fryzjera
        ++zrezygnowani;
        enqueue(zrezygnowani_klienci, id, NULL);
        state();
        // Mutex poczekalni zostal zwolniony po wyjsciu Klienta
        pthread_mutex_unlock(&mutex_poczekalnia);
    }
}

```

5 Sposób odinstalowania programu

Demon jest odinstalowany z linii komend. Należy przejść do katalogu z plikiem `Makefile` i uruchomić komendę `make clean`.