Politechnika Białostocka

Systemy operacyjne

Dokumentacja projektu nr 2

Temat: **Problem śpiącego fryzjera**

Autorzy projektu: Adam Bajguz
Magdalena Kalisz
Krzysztof Kiełczewski

Studia dzienne
Kierunek: Informatyka
Semestr: IV Grupa zajęciowa: **PS1**

Prowadzący przedmiot: **mgr inż. Mirosław Marzewski**

20 czerwca 2018r.
Data oddania projektu

1 TREŚĆ ZADANIA PROJEKTOWEGO

3. Problem śpiącego fryzjera

Salon fryzjerski składa się z gabinetu z jednym fotelem oraz z poczekalni zawierającej n krzeseł. W danym momencie w gabinecie może być strzyżony tylko jeden klient, reszta czeka na wolnych krzesłach w poczekalni. Fryzjer po skończeniu strzyżenia prosi do gabinetu kolejnego klienta, lub ucina sobie drzemkę, jeśli poczekalnia jest pusta. Nowy klient budzi fryzjera jeśli ten śpi, lub siada na wolne miejsce w poczekalni jeśli fryzjer jest zajęty. Jeśli poczekalnia jest pełna, to klient nie wchodzi do niej i rezygnuje z wizyty.

Napisz program koordynujący pracę gabinetu. Zsynchronizuj wątki klientów i fryzjera:

- bez wykorzystania zmiennych warunkowych (tylko mutexy/semafory) [17 p]
- wykorzystując zmienne warunkowe (condition variables) [17 p]

Aby móc obserwować działanie programu, każdemu klientowi przydziel numer. Program powinien wypisywać komunikaty według poniższego przykładu:

```
Res:2 WRomm: 5/10 [in: 4]
```

Oznacza to, że dwóch klientów zrezygnowało z powodu braku miejsc (Res), w poczekalni (WRoom) zajętych jest 5 z 10 krzeseł, a w gabinecie obsługiwany jest klient o numerze 4. Po uruchomieniu programu z parametrem -debug należy wypisywać cała kolejka klientów czekających, a także lista klientów, którzy nie dostali się do gabinetu. Komunikat należy wypisywać w momencie zmiany którejkolwiek z tych wartości.

Uwagi dot. sprawozdania:

- Rozwiązania wykorzystujące zmienne warunkowe muszą posiadać kolejkę FIFO dla czekających wątków.
- Proszę koniecznie zaznaczyć wybraną wersję projektu (tym samym oczekiwaną ilość punktów).
- Zamieścić w sprawozdaniu tą część kod programu, wyróżniając (np. pogrubioną czcionką) fragmenty korzystające z mechanizmów synchronizacji.
- Opisać konkretne przeznaczenie i sposób wykorzystania każdego mechanizmu synchronizacji (semafora, mutexu, zmiennej warunkowej). Można to zrobić w formie komentarzy do kodu umieszczonych w miejscach, gdzie używany jest któryś z tych mechanizmów).
- Za każdy przypadek potencjalnego wyścigu -3p.

2 LISTA PLIKÓW

a) PR1_mutexy_i_semafory:

Pliki z kodem źródłowym:
- barber.c
- barber.h
- clients_queue.c
- clients_queue.h
- main.c
- utils.c
- utils.h

Plik Makefile:
- Makefile

Plik wykonywalny (skompilowany projekt):
- main

b) PR2_mutexy_i_zmienne_warunkowe:

Pliki z kodem źródłowym:
- barber.c
- barber.h
- clients_queue.c
- clients_queue.h
- main.c
- utils.c
- utils.h

Plik Makefile:
- Makefile

Plik wykonywalny (skompilowany projekt):
- main

3 WYBRANA WERSJA PROJEKTU

Przygotowano rozwiązanie w dwóch wersjach, zgodnie z treścią zadania projektowego, na maksymlanie 34 punkty (2*17punktów):

- bez wykorzystania zmiennych warunkowych (tylko mutexy/semafory) [17 p];
- wykorzystując zmienne warunkowe (condition variables) [17 p].

4 OPIS ISTOTNYCH FRAGMENTÓW KODU ŹRÓDŁOWEGO

4.1 WERSJA BEZ WYKORZYSTANIA ZMIENNYCH WARUNKOWYCH

```
#include "barber.h"

void barber_thread(struct BarberData *data) {
    while (1) {
        /* Fryzjer oczekuje na klienta. Jeśli nie ma żadnego klienta to śpi.
           Wykorzystano zasadę działania semafora.
        */
        sem_wait(&data->client_is_ready);

        /* Blokada dostępu do zmiany stanu poczekalni, ponieważ będziemy zmieniać stan krzeseł
           i nie chcemy aby inny wątek zmienił ich stan.

           Zmiana stanu krzeseł jest spowodowana wezwaniem kolejnego klienta na strzyżenie.
        */
        pthread_mutex_lock(&data->chair_access_lock);

        data->clients_waiting_num--;
        struct Client *new_client = queue_get_next_client(&data->clients_to_barber_queue);
        data->clients_in_barber_num = new_client->id;

        print_str("Next client for hair cut!\n");
        print_info(data);

        // Odblokowujemy dostęp do zmiany stanu poczekalni bo już wzieliśmy klienta.
        pthread_mutex_unlock(&data->chair_access_lock);

        // Ustawiamy w kliencie, że teraz jest jego kolej na strzyżenie.
        sem_post(&new_client->turn_for_cutting);

        // Blokujemy dostęp do zmiany stanu strzyżenia, bo fryzjer już strzyże naszego klienta.
        pthread_mutex_lock(&data->cutting_state_lock);

        sleep(rand() % 5);

        // Ustawiamy w kliencie że już został ostrzyżony oraz odblokowujemy dostęp do zmiany
stanu strzyżenia.
        sem_post(&new_client->was_already_cut);
        pthread_mutex_unlock(&data->cutting_state_lock);

        data->clients_in_barber_num = 0;
    }
}

void customer_thread(struct BarberData *data) {
    /* Blokada dostępu do zmiany stanu poczekalni (liczbę wolnych krzeseł), ponieważ będziemy
zmieniać stan krzeseł
      i nie chcemy aby inny wątek zmienił ich stan.

      Zmiana stanu krzeseł jest spowodowana przyjściem nowego klienta na strzyżenie.
    */
    pthread_mutex_lock(&data->chair_access_lock);

    data->total_clients_num++;

    // Jeśli nie ma miejsca w poczekalni
    if (data->clients_waiting_num >= data->chairs_num) {
        // Wpisujemy że klient zrezygnował ze strzyżenia
        queue_add_client(&(data->resigned_clients_queue), data->total_clients_num);
        data->resigned_clients_num++;

        print_str("New client resigned!\n");

        print_info(data);
```

```
        // Odblokowujemy dostęp do zmiany stanu poczekalni bo klient już przyszedł ale nie został w poczekalni.
        pthread_mutex_unlock(&data->chair_access_lock);
    } else {
        data->clients_waiting_num++;

        struct Client *newClient = queue_add_client(&(data->clients_to_barber_queue), data->total_clients_num);
        print_str("New client in waiting room!\n");
        print_info(data);

        // Informujemy fryzjera że klient jest gotowy na strzyżenie
        sem_post(&data->client_is_ready);

        // Odblokowujemy dostęp do zmiany stanu poczekalni bo klient już przyszedł i zgłosił że jest gotów na strzyżenie.
        pthread_mutex_unlock(&data->chair_access_lock);

        // Klient oczekuje aż nastąpi jego kolej na strzyżenie
        sem_wait(&newClient->turn_for_cutting);

        // Klient oczekuje aż skończy się jego strzyżenie
        sem_wait(&newClient->was_already_cut);
    }
}
```

4.2 WERSJA WYKORZYSTUJĄCA ZMIENNE WARUNKOWE

```
#include "barber.h"

void barber_thread(struct BarberData *data) {

    while (1) {
    /* Blokada dostępu do zmiany stanu poczekalni (liczbę wolnych krzeseł), ponieważ będziemy zmieniać stan krzeseł
       i nie chcemy aby inny wątek zmienił ich stan.
     */
    pthread_mutex_lock(&data->chair_access_lock);

    // Fryzjer oczekuje na klienta. Jeśli nie ma żadnego klienta to śpi.
    while (data->clients_waiting_num == 0) {
        data->clients_in_barber_num = 0;
        pthread_cond_wait(&data->client_is_ready, &data->chair_access_lock);
    }

    // Gdy wyjdziemy z pętli, oznacza to że przyszedł klient na strzyżenie
    data->clients_waiting_num--;

    // Pobieramy klienta z poczekalni
    struct Client *client = queue_get_next_client(&data->clients_to_barber_queue);
    data->clients_in_barber_num = client->id;

    print_str("Next client for hair cut!\n");
    print_info(data);

    // Odblokowujemy możliwość zmianu stanu poczekalni
    pthread_mutex_unlock(&data->chair_access_lock);

    // Ustawiamy w kliencie, że teraz jest jego kolej na strzyżenie.
    pthread_cond_signal(&client->turn_for_cutting);

    // Blokujemy dostęp do zmiany stanu strzyżenia, bo fryzjer już strzyże naszego klienta.
    pthread_mutex_lock(&data->cutting_state_lock);

    // Ustawiamy zmienną warunkową że strzyżemy
    data->cutting = 1;

    // Sygnalizujemy że strzyżemy klienta
    pthread_cond_signal(&data->barber_is_cutting);
```

```
    sleep(rand() % 5);

    // Kończymy strzyżenie
    data->cutting = 0;

    // Sygnalizujemy że fryzjer skończył strzyżenie klienta
    pthread_mutex_unlock(&data->cutting_state_lock);
}
}

void customer_thread(struct BarberData *data) {
    /* Blokada dostępu do zmiany stanu poczekalni (liczbę wolnych krzeseł), ponieważ będziemy
zmieniać stan krzeseł
        i nie chcemy aby inny wątek zmienił ich stan.

        Zmiana stanu krzeseł jest spowodowana przyjściem nowego klienta na strzyżenie.
    */
    pthread_mutex_lock(&data->chair_access_lock);

    ++data->total_clients_num;

    // Jeśli nie ma miejsca w poczekalni
    if (data->clients_waiting_num >= data->chairs_num) {
        // Wpisujemy że klient zrezygnował ze strzyżenia
        queue_add_client(&data->resigned_clients_queue, data->total_clients_num);
        data->resigned_clients_num++;

        print_str("New client resigned!\n");

        print_info(data);

        // Odblokowujemy dostęp do zmiany stanu poczekalni bo klient już przyszedł ale nie
został w poczekalni.
        pthread_mutex_unlock(&data->chair_access_lock);
    } else {
        data->clients_waiting_num++;

        struct Client *client = queue_add_client(&data->clients_to_barber_queue, data-
>total_clients_num);
        print_str("New client in waiting room!\n");
        print_info(data);

        // Informujemy fryzjera że klient jest gotowy na strzyżenie
        pthread_cond_signal(&data->client_is_ready);

        // Odblokowujemy dostęp do zmiany stanu poczekalni bo klient już przyszedł i zgłosił że
jest gotów na strzyżenie.
        pthread_mutex_unlock(&data->chair_access_lock);

        /* Klient oczekuje aż nastąpi jego kolej na strzyżenie, blokujemy dostęp do
           fryzjera i czekamy aż będzie możliwy wstęp do fryzjera.
        */
        pthread_mutex_lock(&data->barber_access_lock);
        while (data->clients_in_barber_num != data->total_clients_num)
            pthread_cond_wait(&client->turn_for_cutting, &data->barber_access_lock);

        // Klient dostał się do fryzjera więc zdejmujemy blokadę
        pthread_mutex_unlock(&data->barber_access_lock);

        // Klient oczekuje aż skończy się jego strzyżenie, blokujemy dodatkowo najpierw stan
strzyżenia
        pthread_mutex_lock(&data->cutting_state_lock);
        while (data->cutting)
            pthread_cond_wait(&data->barber_is_cutting, &data->cutting_state_lock);

        // Klient został ostrzyżony zdejmujemy blokadę
        pthread_mutex_unlock(&data->cutting_state_lock);
    }
}
```

5 SPOSÓB URUCHOMIENIA

- W przypadku braku pliku *main* należy zbudować go z plików źródłowych. W tym celu należy w domyślnym terminalu (np. bash) przejść do katalogu zawierającego pliki, a następnie wykonać polecenie *make* lub *make all* w katalogu zawierającym kod źródłowy i plik Makefile.

  Uwaga:
  Aby przebudować projekt należy wykonać polecenie *make clean* a następnie *make* lub *make all*.

- Uruchomienie obydwu programów następuję poprzez wprowadzenie polecenia *./main N*, gdzie N to liczba krzeseł, w odpowiednim katalogu z jednym z rozwiązań.

- Obydwa programy można uruchomić również z dodatkowym parametrem *-debug* w celu wypisywania kolejki klientów czekających, a także listy klientów, którzy nie dostali się do gabinetu.