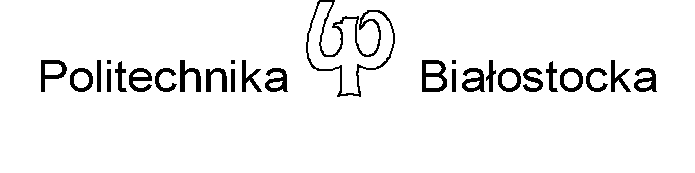
****

**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**

# 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.

# Lista plików

1. 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

1. 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

# 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].

# Opis istotnych fragmentów kodu źródłowego

## 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**);**

**}**

}

## 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**);**

**}**

**}**

# 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.