|  |  |
| --- | --- |
| Wydział Informatyki Politechniki Białostockiej  Pracownia Specjalistyczna Systemów Operacyjnych | Data: 05.06.2024 |
| Projekt: Problem śpiącego fryzjera  Grupa: Ps 9 sem. IV  Zespół:   1. Konopka Maciej 2. Mróz Kamil | Prowadzący:  dr inż. dr inż. Daniel Reska  Ocena: |

1. Treść projektu:

Salon fryzjerski składa się z gabinetu z jednym fotelem (zasób wymagający synchronizacji) oraz z poczekalni zawierającej n krzeseł. W danym momencie w gabinecie może być strzyżony tylko jeden klient (wątek), reszta czeka na wolnych krzesłach w poczekalni. Fryzjer po skończeniu strzyżenia (może być stały czas) 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 w dwóch wersjach programu (razem **34 p**):

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

Aby móc obserwować działanie programu, każdemu klientowi (klientowi) przydziel numer. Każdy klient powinien przychodzić w losowym czasie z przedziału od 0 do wielokrotności czasu strzyżenia. Program powinien wypisywać komunikaty według poniższego przykładu:

Rezygnacja: 2 Poczekalnia: 5/10 [Fotel: 4]

Oznacza to, że dwóch klientów zrezygnowało z powodu braku miejsc (Rezygnacja), w poczekalni (Poczekalnia) zajętych jest 5 z 10 krzeseł, a w gabinecie obsługiwany jest klient o numerze 4. Po uruchomieniu programu z parametrem **-info** 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.

2. Wybrana wersja projektu:

Wybraliśmy implementację dwóch wersji programu (razem **34 p**):

1. bez wykorzystania zmiennych warunkowych (tylko mutexy i semafory) **[17 p]**
2. wykorzystując zmienne warunkowe (condition variables) **[17 p]**

3.1. Fragmenty programu wymagające synchronizacji (część 1)

Procedura pracy wątku fryzjera:

void \*barber\_thread\_routine(void \*arg)

{

while (1)

{

**sem\_wait(&customer\_ready); // 1.**

**pthread\_mutex\_lock(&mutex); // 2.**

// Double-check to ensure the queue isn't empty before proceeding

if (queue\_is\_empty(customer\_queue))

{

**pthread\_mutex\_unlock(&mutex); // 3.**

continue; // Go back to waiting if no customer is actually ready

}

current\_customer\_on\_barberchair = dequeue(customer\_queue);

available\_seats++;

print\_barbershop\_status();

if (print\_info)

{

print\_waiting\_queue(customer\_queue);

}

**pthread\_mutex\_unlock(&mutex); // 4.**

simulate\_work(5);

**pthread\_mutex\_lock(&mutex); // 5.**

current\_customer\_on\_barberchair = 0;

print\_barbershop\_status();

if (print\_info)

{

print\_waiting\_queue(customer\_queue);

}

**pthread\_mutex\_unlock(&mutex); // 6.**

// Signal that the haircut is done

**sem\_post(&haircut\_done); // 7.**

}

return NULL;

}

Mechanizm synchronizacji:

1. Wątek fryzjera jest uśpiony, aż jakiś klient zinkrementuje semafor, dając znać wątkowi fryzjera, że ktoś czeka w kolejce.
2. Blokuje mutex wykluczający jednoczesne modyfikowanie wspólnych zmiennych i sprawdza czy został słusznie wybudzony sprawdzając czy ktoś jest w kolejce klientów.
3. Jeśli nie to wraca do stanu uśpienia (krok 1) jednocześnie zwalniając mutex.
4. Jeśli rzeczywiście jest jakiś klient w kolejce to aktualizuje stan kolejki, zmiennej current\_customer\_on\_barberchair oraz zwalnia mutex.
5. Po wykonaniu pracy aktualizuje stan fotelu blokując przed tym mutex.
6. Po skończeniu korzystaniu z wspólnego zasobu odblokuj mutex.
7. „Wypuść” klienta wywołując funkcję sem\_post.

Procedura pracy wątku klienta:

void **\***customer\_thread\_routine**(**void **\***arg**)**

**{**

int id **=** **\*(**int **\*)**arg**;**

free**(**arg**);**

**pthread\_mutex\_lock(&mutex); *// 1.***

*// Check if the waiting room is full*

**if** **(**available\_seats **==** 0**)**

**{**

resigned\_customers**[**resigned\_customers\_counter**++]** **=** id**;**

**if** **(**print\_info**)**

**{**

print\_resigned\_customers**(**resigned\_customers\_counter**,** resigned\_customers**);**

**}**

**pthread\_mutex\_unlock(&mutex); *// 2.***

**return** **NULL;**

**}**

*// Enqueue the customer since there are available seats*

enqueue**(**customer\_queue**,** id**);**

available\_seats**--;**

print\_barbershop\_status**();**

**if** **(**print\_info**)**

**{**

print\_waiting\_queue**(**customer\_queue**);**

**}**

*// Signal the barber that a customer is ready*

**sem\_post(&customer\_ready); *// 3.***

**pthread\_mutex\_unlock(&mutex); *// 4.***

*// Wait for the haircut to complete*

**sem\_wait(&haircut\_done); *// 5****.*

**return** **NULL;**

**}**

Mechanizm synchronizacji:

1. Zablokuj mutex, by sprawdzić stan poczekalni i ewentualnie jak nie jest pełna, zająć miejsce w kolejce.
2. Jeśli poczekalnia jest pełna aktualizuj tablicę klientów którzy zrezygnowali z wizyty, odblokuj mutex oraz zakończ procedurę.
3. Po zajęciu miejsca w kolejce wywołaj funkcje sem\_post, która zinkrementuje wartość semafora.
4. Odblokuj mutex, by pozwolić innym wątkom na pracę.
5. Czekaj na inkrementację semafora przez wątek fryzjera, sygnalizujący nam, że nasze strzyżenie dobiegło końca.

3.2. Fragmenty programu wymagające synchronizacji (część 2)

Procedura pracy wątku fryzjera:

void **\***barber\_thread\_routine**(**void **\***arg**)**

**{**

**while** **(**1**)**

**{**

*//===== ENTRY SECTION =====//*

**pthread\_mutex\_lock(&mutex); *// 1.***

**while** **(**queue\_is\_empty**(**customer\_queue**))**

**{**

**pthread\_cond\_wait(&customer\_ready, &mutex); *// 2.***

**}**

*//===== CRITICAL SECTION =====//*

current\_customer\_on\_barberchair **=** dequeue**(**customer\_queue**);**

available\_seats**++;**

print\_barbershop\_status**();**

**if** **(**print\_info**)**

**{**

print\_waiting\_queue**(**customer\_queue**);**

**}**

**pthread\_cond\_broadcast(&barber\_ready); *// Wake up all the customer threads for a moment // 3.***

**pthread\_mutex\_unlock(&mutex); *// 4****.*

simulate\_work**(**5**);**

*//===== EXIT SECTION =====//*

**pthread\_mutex\_lock(&mutex); *// 5.***

current\_customer\_on\_barberchair **=** 0**;**

print\_barbershop\_status**();**

**if** **(**print\_info**)**

**{**

print\_waiting\_queue**(**customer\_queue**);**

**}**

**pthread\_cond\_signal(&haircut\_done); *// 6.***

**pthread\_mutex\_unlock(&mutex); *// 7.***

**}**

**return** **NULL;**

**}**

Mechanizm synchronizacji:

1. Zablokuj mutex.
2. Sprawdź czy kolejka jest pusta, to czekaj na sygnał na zmiennej warunkowej customer\_ready jednoczesniej odblokowując mutex.
3. Po rzeczywistej zmianie zmiennej warunkowej customer\_ready, wątek zablokuje mutex, przygotuje się do obsłużenia klienta, a następnie zawiadomi każdy wątek, że jest gotów do pracy (wybudzi się na dłużej tylko ten który był pierwszy w kolejce, wytłumaczone na następnej stronie dokumentacji).
4. Odblokuj mutex, by pozwolić innym wątkom na pracę.
5. Po obsłużeniu klienta zablokuj mutex i przejdź do zaktualizowania stanu fotelu.
6. Powiadom klienta, że strzyżenie dobiegło końca.
7. Odblokuj mutex i zacznij pętlę od nowa.

Procedura pracy wątku klienta:

void **\***customer\_thread\_routine**(**void **\***arg**)**

**{**

int id **=** **\*(**int **\*)**arg**;**

free**(**arg**);**

*//===== ENTRY SECTION =====//*

**pthread\_mutex\_lock(&mutex); *// 1.***

*//===== CRITICAL SECTION =====//*

*// Check if the waiting room is full*

**if** **(**available\_seats **==** 0**)**

**{**

resigned\_customers**[**resigned\_customers\_counter**++]** **=** id**;**

print\_resigned\_customers**(**resigned\_customers\_counter**,** resigned\_customers**);**

**pthread\_mutex\_unlock(&mutex); *// 2.***

**return** **NULL;**

**}**

*// Enqueue the customer since there are available seats*

enqueue**(**customer\_queue**,** id**);**

available\_seats**--;**

print\_barbershop\_status**();**

**if** **(**print\_info**)**

**{**

print\_waiting\_queue**(**customer\_queue**);**

**}**

*// Signal the barber that a custom is ready, if the barber is sleeping*

**if** **(**current\_customer\_on\_barberchair **==** 0**)**

**{**

**pthread\_cond\_signal(&customer\_ready); *// 3.***

**}**

*// Wait until the barber is ready for this customer*

**while** **(**current\_customer\_on\_barberchair **!=** id**)**

**{**

**pthread\_cond\_wait(&barber\_ready, &mutex); *// 4.***

**}**

*// Wait for the haircut to complete*

**while** **(**current\_customer\_on\_barberchair **==** 0**)**

**{**

**pthread\_cond\_wait(&haircut\_done, &mutex); *// 5.***

**}**

*//===== EXIT SECTION =====//*

**pthread\_mutex\_unlock(&mutex); *// 6.***

**return** **NULL;**

**}**

1. Zablokuj mutex, aby wejść do sekcji krytycznej.

2. Jeśli nie ma wolnych miejsc w poczekalni odblokuj mutex i zakończ procedurę.

3. Jeśli było wolne miejsce wejdź do kolejki oraz jeśli fryzjer śpi obudź go wysyłając sygnał na zmiennej warunkowej customer\_ready.

4. Uśpij wątek zwalniając przy tym mutex, do momentu zasygnalizowania przez wątek fryzjera gotowość do strzyżenia.

5. Poczekaj zwalniając mutex, aż strzyżenie dobiegnie końca.

6. Zwolnij mutex i zakończ procedurę.