Systemy operacyjne

Lista zadań nr 14

Na zajęcia 2 i 3 lutego 2022

Należy przygotować się do zajęć czytając następujące materiały: [3, 28, 31 i 32], [1, 2.3.5 i 2.3.6].

UWAGA! W trakcie prezentacji należy być gotowym do zdefiniowania pojęć oznaczonych **wytłuszczoną** czcionką.

WAŻNE! We wszystkich zadaniach zakładamy, że implementacja muteksów, zmiennych warunkowych i planisty zadań jest sprawiedliwa (np. działa zgodnie z polityką `FIFO`). Rozwiązania używające aktywnego czekania uznajemy za błędne!

Zadanie 1. Przypomnij z wykładu na czym polega problem **odwrócenia priorytetów** oraz metodę jego rozwiązywania o nazwie **dziedziczenie priorytetów**? W jakim celu **mutex** pamięta właściciela, tj. wątek który trzyma blokadę? W jaki sposób należy rozszerzyć implementację operacji «`mutex_lock`» i «`mutex_unlock`», żeby nie dopuścić do odwrócenia priorytetów? Czy semafory są odporne na problem odwrócenia priorytetów?

Zadanie 2. Zdefiniuj pojęcie **spójności pamięci podręcznych** (ang. *cache coherence*). Uruchamiamy program ze slajdu pt. „*Thread Function: Memory Accumulation*" do wykładu na komputerze z wieloma procesorami. Czemu program działa szybciej, jeśli zmienna `spacing` jest równa 8 zamiast 1? Posługując się diagramem przejść stanów, zaprezentuj działanie protokołu `MSI`[1] na przykładzie dwóch procesorów, z których każdy zwiększa n razy zawartość komórki pamięci x o 1. Każdy procesor wykonuje w pętli trzy instrukcje: załadowanie wartości z pamięci do rejestru, wykonanie operacji arytmetycznej i zapisanie wartości do pamięci. Chcemy prześledzić jak zmienia się stan linii pamięci podręcznej przechowującej zmienną x, w zależności od akcji procesora i żądań przychodzących z magistrali pamięci.

Zadanie 3. Zdefiniuj pojęcie **modelu spójności pamięci** (ang. *memory consistency model*). Następnie opowiedz jakie gwarancje daje programiście **spójność sekwencyjna**. Niestety procesory wykonujące instrukcje poza porządkiem programu potrafią zmienić kolejność żądań odczytu i zapisu pamięci, co może zostać zaobserwowane przez inne procesory. Artykuł „*Memory ordering*"[2] opisuje model pamięci implementowany przez architekturę `x86-64` i `ARMv7`. Wyjaśnij gwarancje, które daje programiście każdy z tych modeli. Wybierz ciąg instrukcji `load` i `store`, a następnie powiedz, które z nich procesor może zmieniać kolejnością wysyłając do magistrali pamięci.

Zadanie 4 (bonus). Uzasadnij, że poniższy program (algorytm Petersona) zachowuje się poprawnie, gdy wykonywany jest na architekturze `x86-64` (*total store order model*). Wskaż kontrprzykład, który pokona ten algorytm na architekturze `ARMv7` (*weak memory model*). Następnie pokaż gdzie trzeba wstawić instrukcje bariery pamięciowej[3] (więcej niż jedną!), aby program zachowywał się poprawnie.

```
1  shared boolean blocked[2] = { false, false };
2  shared int turn = 0;
3
4  void P (int id) {
5    while (true) {
6      blocked[id] = true;
7      turn = 1 - id;
8      while (blocked[1 - id] && turn == (1 - id))
9        continue;
10     /* put code to execute in critical section here */
11     blocked[id] = false;
12   }
13 }
14
15 void main() { parbegin (P(0), P(1)); }
```

[1] `https://en.wikipedia.org/wiki/MSI_protocol`
[2] `https://en.wikipedia.org/wiki/Memory_ordering`
[3] `https://en.wikipedia.org/wiki/Memory_barrier`

Zadanie 5. Podaj w pseudokodzie implementację **blokady współdzielonej** z operacjami «`init`», «`rdlock`», «`wrlock`» i «`unlock`» używając wyłącznie muteksów i zmiennych warunkowych. Nie definiujemy zachowania dla następujących przypadków: zwalnianie blokady do odczytu więcej razy niż została wzięta; zwalnianie blokady do zapisu, gdy nie jest się jej właścicielem; wielokrotne zakładanie blokady do zapisu z tego samego wątku. Twoje rozwiązanie może dopuszczać głodzenie pisarzy.

Podpowiedź:
```
RWLock = {owner: Thread, readers: int, critsec: Mutex,
          noreaders: CondVar, nowriter: CondVar, writer: Mutex}
```

Ściągnij ze strony przedmiotu archiwum «`so21_lista_14.tar.gz`», następnie rozpakuj i zapoznaj się z dostarczonymi plikami.

UWAGA! Można modyfikować tylko te fragmenty programów, które zostały oznaczone w komentarzu napisem «`TODO`». W poniższych zadaniach nie wolno używać semaforów! Można używać wyłącznie muteksów i zmiennych warunkowych!

Zadanie 6. Restauracja ramen.

W malutkiej restauracji ramen[4] prowadzonej przez starego Japończyka jest stolik z pięcioma krzesłami. Kiedy w restauracji pojawia się klient, a jedno z siedzeń jest puste, może on od razu je zająć. Jeśli wszystkie siedzenia są zajęte, nowy klient musi poczekać, aż wszystkie pięć osób zje swoje ramen i opuści restaurację. Kodeks honorowy zabrania Japończykom odchodzenia od stołu, jeśli je przy nim co najmniej jedna osoba – w przeciwnym wypadku tracą zdolność honorową i popełniają seppuku przy pomocy `assert(3)`.

Twoim zadaniem jest prawidłowe zsynchronizowanie wątków klientów w programie «`ramen.c`» zgodnie z powyższymi założeniami.

Zadanie 7 (2). Mały oddział pocztowy.

Oddział pocztowy liczy n krzeseł dla osób oczekujących na nadanie listu poleconego oraz jedno czynne okienko, przy którym może znajdować się co najwyżej jedna osoba. Klient wchodzi do oddziału i jeśli zobaczy, że nie ma żadnych wolnych krzeseł, to opuszcza budynek z niezadowoleniem. W przeciwnym wypadku zajmuje jedno wolne siedzenie i czeka na swoją kolej. Kiedy pracownik skończy obsługiwać klienta, wtedy woła do okienka następnego interesanta. Klient nie może opuścić budynku, dopóki nie zostanie obsłużony. Jeśli nie ma interesantów, to pracownik oddziału pocztowego powraca do czytania swojej ulubionej książki. Jeśli pojawi się klient, to pracownik odkłada książkę i powraca do pracy.

Twoim zadaniem jest prawidłowe zsynchronizowanie wątków pracownika i klientów w programie «`office.c`» zgodnie z powyższymi założeniami.

Podpowiedź: Możesz użyć systemu z wydawaniem biletów, jak w urzędzie miejskim.

Zadanie 8 (2). Problem kolejki górskiej w wesołym miasteczku.

Jedną z ciekawszych atrakcji wesołego miasteczka jest kolejka górska, która posiada dokładnie jeden wózek. Głodni wrażeń pasażerowie ustawiają się przed bramką wejściową czekając na przejażdżkę. Wózek jest całkowicie zautomatyzowany – podjeżdża do platformy i otwiera drzwiczki. W tym momencie wszyscy pasażerowie muszą opuścić wózek przez bramkę wyjściową. Następnie pasażerowie stojący przed bramką wejściową mogą zacząć wchodzić do środka. Wózek może przewieźć n pasażerów i może ruszyć tylko wtedy gdy jest pełen. Po zamknięciu drzwiczek pasażerowie zostają unieruchomieni ze względów bezpieczeństwa. Po unieruchomieniu mogą machać rękoma, krzyczeć i wykonywać inne czynności, które nie zagrażają ich bezpieczeństwu. Po pełnej emocji przejażdżce wózek podjeżdża do platformy i odblokowuje mechanizm unieruchamiający pozwalając pasażerom opuścić swoje siedzenia.

Twoim zadaniem jest prawidłowe zsynchronizowanie wątków pracownika i klientów w programie «`ride.c`» zgodnie z powyższymi założeniami.

[4] `https://pl.wikipedia.org/wiki/Ramen`

Literatura

[1] „*Systemy operacyjne*”
Andrew S. Tanenbaum, Herbert Bos
Helion; wydanie czwarte; 2015

[2] „*Operating System Concepts*”
Abraham Silberschatz, Peter Baer Galvin; Greg Gagne
Wiley; wydanie dziesiąte; 2018

[3] „*Operating Systems: Three Easy Pieces*”
Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau
https://pages.cs.wisc.edu/~remzi/OSTEP/