Systemy operacyjne

Lista zadań nr 14

Na zajęcia 4 lutego 2020

Należy przygotować się do zajęć czytając następujące rozdziały książek:

- Arpaci-Dusseau: 28 (Locks[1]), 31 (Semaphores[2]), 32 (Common Concurrency Problems[3])

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

Zadanie 1. Zdefiniuj **sprawiedliwość** (ang. *fairness*), która jest jedną z właściwości środków synchronizacji. Standard `POSIX.1` oferuje muteksy, semafory i zmienne warunkowe, które są sprawiedliwe. Czasami sprawiedliwe semafory określa się również mianem **silnych** (ang. *strong semaphore*). Dodatkowo wymienione środki są odporne na zjawisko **odwrócenia priorytetów**. Wyjątkiem są blokady wirujące, które ani nie zapobiegają odwróceniu priorytetów, ani nie muszą być sprawiedliwe. Wyjaśnij skąd to wynika! W jaki sposób pozostałe środki synchronizacji zapewniają sprawiedliwość i zapobiegają odwróceniu priorytetów?

Zadanie 2. Podaj implementację (w języku `C`) **semafora** z operacjami «`init`», «`wait`» oraz «`post`» używając wyłącznie muteksów i zmiennych warunkowych standardu `POSIX.1`. Pamiętaj, że wartość semafora musi być zawsze nieujemna.

Podpowiedź: `typedef struct Sem { pthread_mutex_t mutex; pthread_cond_t waiters; int value; } Sem_t;`

Zadanie 3. 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}
```

Zadanie 4 (bonus). Opisz semantykę operacji «`FUTEX_WAIT`» i «`FUTEX_WAKE`» mechanizmu `futex(2)` wykorzystywanego w systemie `Linux` do implementacji środków synchronizacji w przestrzeni użytkownika. Podaj w pseudokodzie[4] implementację funkcji «`lock`» i «`unlock`» **semafora binarnego** korzystając wyłącznie z **futeksów** i atomowej instrukcji `compare-and-swap`. Odczyty i zapisy komórek pamięci są atomowe. Operacja «`unlock`» ma prosić jądro o wybudzenie wątków tylko wtedy, gdy istnieje **rywalizacja** o blokadę.

Podpowiedź: Wartość futeksa wyraża stany: (0) $unlocked$, (1) $locked \wedge |waiters| = 0$, (2) $locked \wedge |waiters| \geq 0$.

Ściągnij ze strony przedmiotu archiwum «`so19_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! Należy użyć muteksów i zmiennych warunkowych.

Zadanie 5 (P). Restauracja ramen.

W malutkiej restauracji ramen[5] 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 fryzjera i klientów w programie «`ramen.c`» zgodnie z powyższymi założeniami.

[1] `http://pages.cs.wisc.edu/~remzi/OSTEP/threads-locks.pdf`
[2] `http://pages.cs.wisc.edu/~remzi/OSTEP/threads-sema.pdf`
[3] `http://pages.cs.wisc.edu/~remzi/OSTEP/threads-bugs.pdf`
[4] „Python is executable pseudocode. Perl is executable line noise.” – Bruce Eckel
[5] `https://pl.wikipedia.org/wiki/Ramen`

Zadanie 6 (P, 2, bonus). 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 7 (P, 2, bonus). 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 – pojeżdża 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 zamnkięciu drzwiczek pasażerowie zostają unieruchomieni ze względów bezpieczeństwa. Po unieruchomieniu mogą machać rękoma, krzyczeć i wykonywać inne czynności, które nie zagrażają ich niebezpieczeństwu. Po pełnej emocji przejażdżce wózek podjeżdża do platformy i odblokowuje mechnizm unieruchomiający pozwalając pasażerom opuścić swoje siedzenia.

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