Systemy operacyjne

Lista zadań nr 11

Na zajęcia 14 i 15 stycznia 2020

Należy przygotować się do zajęć czytając następujące rozdziały książek:

- Tanenbaum (wydanie czwarte): 2.3.1 – 2.3.3, 6.1 – 6.3

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

Zadanie 1. Zdefiniuj, w możliwie jak najbardziej formalny sposób, zjawisko **zakleszczenia** (ang. *deadlock*), **uwięzienia** (ang. *livelock*) i **głodzenia** (ang. *starvation*). Rozważmy ruch uliczny – kiedy na skrzyżowaniach może powstać każde z tych zjawisk? Zaproponuj metodę (a) **wykrywania i usuwania** zakleszczeń (b) **zapobiegania** zakleszczeniom. Pokaż, że nieudana próba zapobiegania zakleszczeniom może zakończyć się wystąpieniem zjawiska uwięzienia lub głodzenia.

UWAGA! Interesujące fragmenty poniższych programów należy opisywać z użyciem trójek Hoare'a[1].

Niech zapis $\{P\}I\{Q\}$ oznacza, że formuły P i Q są prawdziwe odpowiednio przed i po wykonaniu instrukcji I. Formuły te nazywamy kolejno *warunkami wstępnymi* (ang. *preconditions*) i *warunkami końcowymi* (ang. *postconditions*). Program $\{P_1\}I_1\{Q_1\};\{P_2\}I_2\{Q_2\}$ może zostać wywłaszczony między instrukcjami I_1 i I_2, zatem nie musi zachodzić $Q_1 \equiv P_2$!

Zadanie 2. W poniższym programie występuje **sytuacja wyścigu** (ang. *race condition*) dotycząca dostępów do współdzielonej zmiennej «`tally`». Wyznacz jej najmniejszą i największą możliwą wartość.

```
1  const int n = 50;
2  shared int tally = 0;
3
4  void total() {
5    for (int count = 1; count <= n; count++)
6      tally = tally + 1;
7  }
8
9  void main() { parbegin (total(), total()); }
```

Dyrektywa «`parbegin`» rozpoczyna współbieżne wykonanie procesów. Maszyna wykonuje instrukcje arytmetyczne wyłącznie na rejestrach – tj. kompilator musi załadować wartość zmiennej «`tally`» do rejestru, przed wykonaniem dodawania. Jak zmieni się przedział możliwych wartości zmiennej «`tally`», gdy wystartujemy k procesów zamiast dwóch? Odpowiedź uzasadnij.

Zadanie 3. Poniżej znajduje się propozycja[2] programowego rozwiązania problemu **wzajemnego wykluczania** dla dwóch procesów. Znajdź kontrprzykład, w którym to rozwiązanie zawodzi. Okazuje się, że nawet recenzenci renomowanego czasopisma „Communications of the ACM" dali się zwieść.

```
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      while (turn != id) {
8        while (blocked[1 - id])
9          continue;
10       turn = id;
11     }
12     /* put code to execute in critical section here */
13     blocked[id] = false;
14   }
15 }
16
17 void main() { parbegin (P(0), P(1)); }
```

[2] Harris Hyman, „Comments on a Problem in Concurrent Programming Control", January 1966.

Zadanie 4. Algorytm Petersona[3] rozwiązuje programowo problem wzajemnego wykluczania. Zreferuj poniższą wersję implementacji tego algorytmu dla dwóch procesów. Wykaż jego poprawność.

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

Zadanie 5 (bonus). Na podstawie strony Memory Ordering[4] wyjaśnij kiedy programista musi dobrze rozumieć w jakiej kolejności procesor realizuje operacje na pamięci.

Algorytm wzajemnego wykluczania z poprzedniego zadania powinien działać bez modyfikacji na architekturze x86-64 realizującej **model pamięci TSO** (ang. *Total Store Order*). Pokaż, że algorytm ten może zawieść na architekturze ARMv8 realizującej **osłabiony model pamięci** (ang. *weakly-ordered memory model*). Napraw go wstawiając w odpowiednie miejsca instrukcje bariery pamięciowej[5] (`DMB ISH*`) dla pamięci współdzielonej między rdzenie procesora (inner shareable).

Ściągnij ze strony przedmiotu archiwum «`so19_lista_11.tar.gz`», następnie rozpakuj i zapoznaj się z dostarczonymi plikami. **UWAGA!** Można modyfikować tylko te fragmenty programów, które zostały oznaczone w komentarzu napisem «`TODO`».

Zadanie 6 (P). Program «`echoclient-thread`» wczytuje plik tekstowy zadany z wiersza poleceń i dzieli go na linie zakończone znakiem `'\n'`. Następnie startuje podaną liczbę wątków, z których każdy wykonuje w pętli: nawiązanie połączenia, wysłanie «`ITERMAX`» losowych linii wczytanego pliku, zamknięcie połączenia. Wątki robią to tak długo, aż do programu nie przyjdzie sygnał «`SIGINT`».

Twoim zadaniem jest tak uzupełnić kod, by program uruchamił «`nthreads`» wątków i poczekał na ich zakończenie. Czemu nie można go łatwo przerobić w taki sposób, żeby główny wątek czekał na zakończenie pozostałych wątków tak, jak robi się to dla procesów przy pomocy `wait(2)`?

Zadanie 7 (P). Program «`echoserver-poll`» implementuje serwer usługi «`echo`» przy pomocy wywołania `poll(2)`, ale bez użycia wątków i podprocesów. W kodzie wykorzystano wzorzec projektowy **pętli zdarzeń** (ang. *event loop*) do współbieżnej obsługi wielu połączeń sieciowych. Program «`echoserver-select`» robi to samo, ale przy pomocy wywołania `select(2)`.

Twoim zadaniem jest uzupełnienie pliku źródłowego «`echoserver-poll.c`». W pętli zdarzeń należy obsłużyć połączenia przychodzące, nadejście nowych danych na otwartych połączeniach i zamknięcie połączenia. Do przetestowania serwera użyj programu «`echoclient-thread`» z poprzedniego zadania.

Zadanie 8 (P). Dla jakich parametrów wiersza poleceń obserwujesz niepoprawne zachowanie programów «`bug-1`» i «`bug-2`»? Przeanalizuj kod źródłowy programów i wytypuj przyczyny błędów. Następnie zweryfikuj swoje podejrzenia: skompiluj programy z opcją «`-fsanitize=thread`» i uruchom je. Wyjaśnij znaczenie komunikatów diagnostycznych pochodzących z Thread Sanitizer[6], po czym pokaż jak naprawić błędy.

[3] `https://en.wikipedia.org/wiki/Peterson's_algorithm`
[4] `https://developer.arm.com/docs/den0024/a/memory-ordering`
[5] `https://developer.arm.com/docs/den0024/a/memory-ordering/barriers`
[6] `https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.308.3963`