Systemy operacyjne (zaawansowane)

Lista zadań nr 5

Na zajęcia 16 listopada 2017

Należy przygotować się do zajęć czytając następujące rozdziały książek:

- Tanenbaum (wydanie czwarte): 2.3, 6.1, 6.8
- Stallings (wydanie ósme): 5.1 – 5.5, 6.1, 6.2

UWAGA! W trakcie prezentacji rozwiązań należy zdefiniować i wyjaśnić pojęcia, które zostały oznaczone **wytłuszczoną** czcionką. Zadania oznaczone jako **(P)** proszę rozwiązać samodzielnie (!!!), a następnie spisać na kartce formatu A4 i oddać prowadzącemu na początku zajęć. W trakcie prezentacji tych zadań analizowany kod będzie wyświetlany rzutnikiem.

Zadanie 1. Wymień cztery warunki konieczne do zaistnienia **zakleszczenia** (ang. *deadlock*). Czym różni się zakleszczenie od **uwięzienia** (ang. *livelock*) i **głodzenia** (ang. *starvation*)? W jaki sposób programista może przeciwdziałać zakleszczeniom (ang. *deadlock prevention*)?

Zadanie 2. Trzy procesy współzawodniczą o zasoby poetykietowane od A do F. Używając **grafu przydziału zasobów** (ang. *resource allocation graph*) wykaż możliwość wystąpienia zakleszczenia w poniższej implementacji. Zmieniając kolejność zakładania blokad w obrębie procesu, wyeliminuj możliwość powstania zakleszczenia. Rozwiązanie problemu podeprzyj zaktualizowanym grafem.

```
void P0() {
  forever {
    acquire(A);
    acquire(B);
    acquire(C);
    // use A, B, C
    release(A);
    release(B);
    release(C);
  }
}
```

```
void P1() {
  forever {
    acquire(D);
    acquire(E);
    acquire(B);
    // use D, E, B
    release(D);
    release(E);
    release(B);
  }
}
```

```
void P2() {
  forever {
    acquire(C);
    acquire(F);
    acquire(D);
    // use C, F, D
    release(C);
    release(F);
    release(D);
  }
}
```

Zadanie 3. Wyłączanie przerwań to najprostszy mechanizm implementacji sekcji krytycznych wewnątrz jądra systemu operacyjnego. Wykaż, że jest to właściwy mechanizm do wyeliminowania wyścigów między normalnym kodem jądra, a kodem wykonywanym w procedurze obsługi przerwania. Wyjaśnij jak analogiczny problem powstaje w przestrzeni użytkownika. Jak się go rozwiązuje?

Zadanie 4. Instrukcje **load-linked** i **store-conditional**, dostępne w procesorach `RISC`, to alternatywa dla instrukcji atomowych **compare-and-swap**, itp. Pokaż jak w asemblerze `MIPS` zaimplementować **blokadę wirującą** (ang. *spin-lock*). Jakie są zalety instrukcji `LL` i `SC` w porównaniu do `CAS`?

Zadanie 5. Na podstawie przykładu modyfikacji stanu kont bankowych z rozdziału 24[1] książki „Beautiful Code" wyjaśnij czemu złożenie ze sobą poprawnych współbieżnych programów używających blokad nie musi dać poprawnego programu (aka „locks are not composable").

[1] `https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/beautiful.pdf`

Zadanie 6 (P). W poniższym programie występuje **wyścig** (ang. *race condition*) na współdzielonej zmiennej `tally`. Wyznacz jej najmniejszą i największą możliwą wartość. Dyrektywa `parbegin` rozpoczyna współbieżne wykonanie podprocesów.

```
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; /* to samo co tally++ */
7 }
8
9 void main() { parbegin (total(), total()); }
```

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 7 (P). 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)); }
```

Zadanie 8 (P). Nowoczesne procesory stwarzają iluzję sekwencyjnego przetwarzania programów. Ze względu na pamięć podręczną i wykonanie instrukcji **poza porządkiem programu** (ang. *Out-of-Order execution*) inne procesory w systemie **SMP** mogą obserwować kolejność wprowadzania zapisów do pamięci głównej w innym porządku, niż wynikałoby to z kolejności instrukcji w programie. Nawet, gdy uda nam się napisać poprawny współbieżny program, to może się zdarzyć, że na maszynie wielordzeniowej będzie wykonywał się niepoprawnie.

Rozważmy wykonanie dowolnego przeplotu równoległego wykonania poniższych dwóch programów. Zmienne są przechowywane w pamięci, a `$t0` to rejestr tymczasowy. Początkowo `a = 1` i `b = 2`. Podaj możliwe końcowe wartości zmiennych `c` i `d` uwzględniając instrukcje **barier pamięciowych** (§6.8).

```
STORE(a, 3);
...
mb();
...
STORE(b, 4);
```

```
LOAD($t0, b);
STORE(c, $t0);
...
rmb();
...
LOAD($t0, a);
STORE(d, $t0);
```

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