

# Systemy komputerowe

## Lista zadań nr C

Na zajęcia 3–6 czerwca 2019

Należy przygotować się do zajęć czytając następujące rozdziały książek:

- Tanenbaum (wydanie czwarte): 2.3, 6.1
- Stallings (wydanie dziewiąte): 5.1 – 5.4, 6.1, 6.2

**UWAGA!** W trakcie prezentacji należy być gotowym do zdefiniowania pojęć oznaczonych **wytfuszczoną** czcionką. Zadania oznaczone **(S)** proszę rozwiązać samodzielnie! Pozostałe zadania można rozwiązywać grupowo. Rozwiązanie zadania oznaczonego **(P)** należy pokazać z użyciem rzutnika.

**Zadanie 1.** Wyjaśnij różnice między **zakleszczeniem** (ang. *deadlock*), **uwięzieniem** (ang. *livelock*) i **głodzeniem** (ang. *starvation*). W podręcznikach pojęcia te odnoszą się do **zasobów**. Pokaż, że podobne problemy występują w przypadku **przesyłania komunikatów**.

**Zadanie 2.** Wymień cztery warunki konieczne do zaistnienia zakleszczenia. W jaki sposób programista może **przeciwdziałać** zakleszczeniom (ang. *deadlock prevention*)? Których z proponowanych rozwiązań nie implementuje się w praktyce i dlaczego?

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*). Zauważ, że z racji wyłączenia dla programu  $\{P_1\}I_1\{Q_1\}; \{P_2\}I_2\{Q_2\}$  nie musi zachodzić  $Q_1 \equiv P_2!$

**UWAGA!** W kolejnych zadaniach należy jawnie opisywać globalny stan przy pomocy formuł logicznych.

**Zadanie 3 (S).** W poniższym programie występuje **sytuacja wyścigu** (ang. *race condition*) na współdzielonej zmiennej «tally». Wyznacz jej najmniejszą i największą możliwą wartość. Dyrektywa «parbegin» rozpoczyna współbieżne wykonanie procesów.

```
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 4 (S).** Rozważmy poniższy kod ze slajdów do wykładu. Zakładamy, że kolejka «queue» przechwytuje do  $n$  elementów. Wszystkie operacje na kolejce są **atomowe** (ang. *atomic*). Startujemy po jednym wątku wykonującym kod procedury «producer» i «consumer». Procedura «sleep» usypia wołający wątek, a «wakeUp» budzi wątek wykonujący daną procedurę. Wskaż przeplot instrukcji, który doprowadzi do (a) błędu wykonania w linii 6 i 13<sup>1</sup> (b) zakleszczenia w liniach 5 i 12.

<sup>1</sup>A zatem uwaga poczyniona na wykładzie o poprawności kodu w tej sytuacji była błędna. Rozumowanie o programach z wątkami jest trudne nawet, gdy jest ich tylko dwa!

```

1     def producer():
2         forever:
3             item = produce()
4             if queue.full():
5                 sleep()
6             queue.push(item)
7             if not queue.empty():
8                 wakeup(consumer)
9     def consumer():
10        forever:
11            if queue.empty():
12                sleep()
13            item = queue.pop()
14            if not queue.full():
15                wakeup(producer)
16            consume(item)

```

**Zadanie 5.** Przeczytaj rozdział 2 artykułu „[Beautiful concurrency<sup>2</sup>](#)”. Na podstawie poniższego przykładu wyjaśnij czemu złożenie ze sobą poprawnych współbieżnych podprocedur używających blokad nie musi dać poprawne procedury (tj. „locks are not composable”). Jak poprawić procedurę transfer? Czemu według autorów artykułu blokady nie są dobrym narzędziem do strukturyzowania współbieżnych programów?

```

1     class Account {
2         private int balance;
3         synchronized void withdraw(int n) { balance -= n; }
4         synchronized void deposit(int n) { balance += n; }
5     }
6
7     void transfer(Account from, Account to, int amount) {
8         from.withdraw(amount);
9         to.deposit(amount);
10    }

```

**Zadanie 6 (S).** Poniżej znajduje się propozycja<sup>3</sup> programowego rozwiązania problemu wzajemnego wykluczenia 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 7 (S, P).** Podaj w pseudokodzie implementację **semafora** z operacjami «init», «down» i «up» używając wyłącznie muteksów i zmiennych warunkowych standardu POSIX.1. Dopuszczamy ujemną wartość semafora.

**Podpowiedź:** struct semaphore { pthread\_mutex\_t critsec; pthread\_cond\_t waiters; int count; };

<sup>2</sup><https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/beautiful.pdf>

<sup>3</sup>Harris Hyman, „Comments on a Problem in Concurrent Programming Control”, January 1966.