Systemy komputerowe

Lista zadań nr 14

Na ćwiczenia 11 czerwca 2025

Zadanie 1 (2pkty). Zaprezentuj klasyczny Uniksowy algorytm planowania procesów.

Wskazówka: Przeczytaj podrozdziały 8.1.1 – 8.1.4 książki "Maurice J. Bach. 1986. The design of the UNIX operating system. Prentice-Hall, Inc., USA" zamieszczonej w SKOSie.

Zadanie 2. Zaprezentuj algorytm planowania ze sprawiedliwym podziałem (ang. *fair share scheduler*), będący wariantem algorytmu z poprzedniego zadania.

Wskazówka: Przeczytaj podrozdział 8.1.5 książki wymienionej we wskazówce do zadania 1.

W poniższych zadaniach zakładamy, że programy uruchamiane są na komputerze z silnym modelem pamięci: procesor wykonuje instrukcje bez *reorderingu*, czyli zgodnie z porządkiem programu, modyfikacje pamięci wykonane przez jeden z procesorów są natychmiast widoczne dla innych.

Zadanie 3. Dwa procesy wykonują funkcję sum() operującą na współdzielonej zmiennej counter, o początkowej wartości równej 0.

```
void sum() {
  for (int i = 1; i <= 50; i++)
    counter = counter + 1;
}
```

Jaka jest najmniejsza i największa możliwa wartość tej zmiennej po zakończeniu pracy przez obydwa procesy?

Zadanie 4. Dwa procesy P_0 i P_1, współdzielą dwuelementową tablicę flag[], zainicjowaną początkowo wartościami false. Kod procesu P_i $(i \in \{0,1\})$ wygląda następująco

```
while (true) {
  lock(i);      /* sekcja wejściowa   */
  ...           /* sekcja krytyczna?  */
  unlock(i);  /* sekcja wyjściowa   */
}
```

Oto implementacja funkcji lock() i unlock():

```
void lock(int my_id) {
  flag[my_id] = true;
  while (flag[1 - my_id] == true);
}

void unlock(int my_id) {
  flag[my_id] = false;
}
```

Czy ten kod rozwiązuje problem sekcji krytycznej? Które z warunków poprawnego rozwiązania tego problemu są spełnione, a które nie? Odpowiedzi uzasadnij.

Zadanie 5. Pokaż, że algorytm Petersona poprawnie rozwiązuje problem sekcji krytycznej dla dwóch procesów, tzn. spełnia warunki wzajemnego wykluczania, postępu i ograniczonego czekania. W algorytmie tym procesy P_0 i P_1, współdzielą dwuelementową tablicę flag[] oraz zmienną turn, której wartość początkowa nie jest istotna. Oto funkcje lock() i unlock() tworzące ten algorytm:

```
void lock(int my_id) {
  flag[my_id] = true;
  turn = 1 - my_id;
  while (flag[1 - my_id] == true && turn == 1-my_id);
}

void unlock(int my_id) {
  flag[my_id] = false;
}
```

Zadanie 6. W algorytmie Petersona, w funkcji lock() zamieniamy miejscami instrukcje przypisania do flag[my_id] i zmiennej turn. Czy otrzymany algorytm nadal rozwiązuje problem sekcji krytycznej? Które z warunków poprawnego rozwiązania tego problemu są spełnione, a które nie? Odpowiedzi uzasadnij.

Zadanie 7. W algorytmie Petersona, w funkcji lock() zamieniamy instrukcję turn = 1 - my_id; na turn = my_id;. Czy otrzymany algorytm nadal rozwiązuje problem sekcji krytycznej? Które z warunków poprawnego rozwiązania tego problemu są spełnione, a które nie? Odpowiedzi uzasadnij.

Poniższe dwa zadania są bonusowe.

Zadanie 8. Oto implementacja protokołu wzajemnego wykluczania przy pomocy instrukcji maszynowej TestAndSet. Przeanalizuj ją i stwierdź, które z pozostałych dwóch warunków poprawnego rozwiązania problemu sekcji krytycznej zawodzą. Procesy współdzielą zmienną lock, o początkowej wartości false.

```
void lock(int my_id) {
  while (TestAndSet(&lock));
}

void unlock(int my_id) {
  lock = false;
}
```

Wskazówka: Instrukcja TestAndSet(&arg) ustawia argument arg na True i zwraca jego poprzednią wartość. Instrukcja działa atomowo, tzn. żaden inny proces nie uzyska dostępu do zmiennej arg podczas działania instrukcji TestAndSet. Zobacz też: Podrozdział 6.4 Abraham Silberschatz, Peter B. Galvin, and Greg Gagne. 2012. Operating System Concepts (**8th/10th. ed.**).

Zadanie 9. Udowodnij, że poniżej podane funkcje i tworzą rozwiązanie problemu sekcji krytycznej dla dowolnej liczby n procesów. Współdzielonymi zmiennymi są lock oraz tablica waiting[n] o początkowych wartościach false.

```
void lock(int my_id) {
  waiting[my_id] = true;
  bool key = true;
  while (waiting[my_id] && key)
    key = TestAndSet(&lock);
  waiting[my_id] = false;
}

void unlock(int my_id) {
  int j = (my_id + 1) % n;
```

```
   while ((j != my_id) && !waiting[j])
      j = (j + 1) % n;

   if (j == my_id)
      lock = false;
   else
      waiting[j] = false;
}
```

Wskazówka: Podrozdział 6.4 Abraham Silberschatz, Peter B. Galvin, and Greg Gagne. 2012. Operating System Concepts (**8th/10th. ed.**).