Michał Bronikowski

Systemy Operacyjne 2017
problem złodzieja jabłek

wersja oparta na wątkach

15 listopada 2017

Spis treści

1. Opis zadania

1.1 Problem złodzieja jabłek

Problem został przeze mnie wymyślony na potrzeby zadania. Wzorowałem się na klasycznym problemie synchronizacji tj. problemie producenta i konsumenta. Powiedzmy, że jesteśmy złodziejem jabłek i w naszej miejscowości są dwa sady A i B, w których w dość szybkim tempie przybywa jabłek. Kradniemy jabłka z sadu A lub z sadu B. W międzyczasie w sadach, dzięki działaniu właścicieli i natury jabłek przybywa. Problem polega na takim zsynchronizowaniu procesów sadów A i B oraz procesu złodzieja, żeby złodziej nie ukradł wszystkich jabłek z sadów, ponieważ to zwróciłoby uwagę właściciela i nasz złodziej mógłby mieć problemy.

1.2 Rozwiązanie

W rozwiązaniu korzystam z trzech semaforów:

- skradziono1
- skradziono2
- uroslo

Złodziej czeka aż w sadach przybędzie jabłek co będzie skutkowało opuszczeniem semafara uroslo. Po czym przystępują do kradzieży jabłek z losowo wybranego przez siebie sadu. Po kradzieży zwalnia się semafor skradziono1 albo skradziono1, a zamyka uroslo. Po zwolnieniu semaforów skradziono1 i skradziono2 w sadach ponownie rusza, produkcja jabłek.W celu zabezpieczenia dostępu do sekcjii krytycznych programu użyłem Mutex'ów.

2. Implementacja

Program należy uruchamiać na komputerze pod kontrolą systemu operacyjnego Linux. Program składa się z jednego pliku źródłowego sem.c. Załączony został również Makefile pomagający skompilować program.

```
$ make
```

1: Skompilowanie programu poleceniem make

Program uruchamiamy poleceniem:

```
$ ./sem
```

2: Uruchomienie programu

2.1 Struktura programu

Program składa się z jednego pliku źródłowego sem.c. W programie możemy podzielić na następujące części:

```c
  int main()
{
    srand(time(0));
    pthread_attr_t attr;
    pthread_attr_init(&attr);
    pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);

    pthread_mutex_init(&mutex, NULL);
    sem_init(&skradziono1, 0, 5);
    sem_init(&skradziono2, 0, 5);
    sem_init(&uroslo, 0, 0);

    pthread_t threads[3];
    pthread_create(&threads[0], &attr, sadA, NULL);
    pthread_create(&threads[2], &attr, sadB, NULL);
    pthread_create(&threads[1], &attr, zlodziej, NULL);
    pthread_create(&threads[2], &attr, sadB, NULL);
    pthread_join(threads[0], NULL);
    pthread_join(threads[1], NULL);
    pthread_join(threads[2], NULL);

    pthread_attr_destroy(&attr);
    pthread_mutex_destroy(&mutex);
    sem_destroy(&skradziono1);
    sem_destroy(&uroslo);
    sem_destroy(&skradziono2);

    pthread_exit(NULL);
    return 0;
}
```

3: Funkcja główna

```c
void *zlodziej()
{
    while(1)
    {
      sem_wait(&uroslo);
      pthread_mutex_lock(&mutex);
      int sad = (rand()%2)+1;
      if(sad == 1)
      {
          if(jablko_z_A < 1)
          {
              printf("Nie ukradne z sadu A - nie maja juz jablek\n");
              return 0;
          }
          else
          {
          --jablko_z_A;
          zA++;
          printf("Kradne jablko z sadu A mam juz %d jablek z sadu A i %d jablek z
    sadu B \n ",zA,zB);
          }
          sem_post(&skradziono2);
      }
      else
      {   if(jablko_z_B < 1)
          {
              printf("Nie ukradne z sadu B - nie maja juz jablek\n");
              return 0;
          }
          else
          {
          --jablko_z_B;
          zB++;
```

```
33            printf("Kradne jablko z sadu B mam juz %d jablek z sadu A i %d jablek z
    sadu B \n ",zA,zB);
34            }
35            sem_post(&skradziono1);
36          }
37            pthread_mutex_unlock(&mutex);
38      }
39  }
```

4: Funkcja odpowiadająca za złodzieja

```
1  void *sadB() {
2      while (1)
3      {
4          sem_wait(&skradziono1);
5          pthread_mutex_lock(&mutex);
6          printf("Jestem z sadu  B i mam %d jab ek\n", ++jablko_z_B);
7          pthread_mutex_unlock(&mutex);
8          sem_post(&uroslo);
9      }
10 }
```

5: Funkcja odpowiadająca za sad "B"

```
1  void *sadA() {
2      while (1)
3      {
4          sem_wait(&skradziono2);
5          pthread_mutex_lock(&mutex);
6          printf("Jestem z sadu A i mam %d jab ek\n", ++jablko_z_A);
7          pthread_mutex_unlock(&mutex);
8          sem_post(&uroslo);
9      }
10 }
```

6: Funkcja odpowiadająca za sad "A"

2.2 Jak działa program i czemu działa?

Na początku inicjalizuję semafory o nazwach: skradziono,uroslo,skradziono2. Inicjalizuję również mutex dający dostęp do zasobów tylko dla jednego wątku. Następnie tworzę trzy wątki: sadA,sadB,zlodziej. Procesy odpowiadające za „produkcję" jabłek w sadach są zwalnianie przez semafory skradziono i skradziono2. Każdy z procesów sadów informuje o tym, że zwiększyła się ilość dostępnych jabłek. Proces złodzieja losowo wybiera, z którego sadu chce ukraść jabłka, jeżeli w danym sadzie nie ma już jabłek oznacza to, że synchronizacja nie działa poprawnie i wtedy program kończy swoje działanie. Każdorazowo po kradzieży złodziej zwalnia odpowiedni semafor przypisany do sadu A lub sadu B. Na starcie w każdym sadzie jest dziesięć jabłek oraz wartości odpowiadających im semaforom ustawione są na pięć, a złodzieja na zero. Taki zabieg pozwala nam uniknąć sytuacji, w której złodziej będzie chciał ukraść jabłka z sadu, w którym tych jabłek nie ma.