

Michał Bronikowski

Systemy Operacyjne 2017
problem złodzieja jabłek

wersja z procesami ciężkimi

15 listopada 2017

Spis treści

1. Opis zadania	3
1.1. Problem złodzieja jabłek	3
1.2. Rozwiązanie	3
1.3. Implementacja	3
2. Testowanie	5

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 dwóch semaforów:

- skradziono
- uroslo

Złodziej czeka aż w sadach przybędzie jabłek co będzie skutkowało opuszczeniem semafora `uroslo`. Po czym przystępują do kradzieży jabłek z losowo wybranego przez siebie sadu. Po kradzieży zwalnia się semafor `skradziono`, a zamyka `uroslo`. Po zwolnieniu semafora `skradziono` w sadach ponownie rusza, produkcja jabłek.

1.3 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.

```
1 $ make
```

1: Skompilowanie programu poleceniem make

Program uruchamiamy poleceniem:

```
1 $ ./sem
```

2: Uruchomienie programu

W programie zdefiniowałem strukturę `Zasob_Dzielony`

```
1 struct Zasob_Dzielony
2 {
3     sem_t skradziono ;
4     sem_t uroslo ;
5     int jablko_z_A ;
6     int jablko_z_B ;
7     int zA,zB;
8     int gdzie;
9 } *z_dz ;
```

3: struktura `Zasob_Dzielony`

Struktura zawiera elementy, które będą współdzielone przez wszystkie procesy programu.

Funkcja zarządzająca naszym złodziejem została zdefiniowana w następujący sposób:

```
1 void zlodziej()
2 {
3     while(1)
4     {
5         sem_wait(&z_dz->uroslo);
6
7         sad = rand()%2+1;
8         if(sad == 1)
9         {
10             if(z_dz->jablko_z_A < 1)
11             {
12                 printf("Nie ukradne z sadu A - nie maja ju  jablek\n");
13
14             }
15             else
16             {
17                 --z_dz->jablko_z_A;
18                 z_dz->zA++;
19                 printf("Kradne jablko z sadu A mam juz %d jab ek z sadu A i %d jablek z sadu B \n "
20 ,z_dz->zA,z_dz->zB);
21             }
22             sem_post(&z_dz->skradziono);
23             z_dz->gdzie=1;
24         }
25         else
26         {
27             if(z_dz->jablko_z_B < 1)
28             {
29                 printf("Nie ukradne z sadu B - nie maja juz jablek\n");
30
31             }
32             else
33             {
34                 --z_dz->jablko_z_B;
35                 z_dz->zB++;
36                 printf("Kradne jablko z sadu B mam juz %d jab ek z sadu A i %d jablek z sadu B \n "
37 ,z_dz->zA,z_dz->zB);
38             }
39             sem_post(&z_dz->skradziono);
40             z_dz->gdzie=2;
41         }
42     }
43 }
```

4: Funkcja zarządzająca złodziejem

Po każdej dokonanej kradzieży złodziej zaznacza, z którego sadu ukradł jabłko, poprzez ustawienie wartości współdzielonej zmiennej `gdzie` odpowiednio na 1 lub 2. Dzięki temu wiemy, w którym sadzie produkcja powinna ruszyć szybciej, aby w sad był konkurencyjny i nie zbankrutował (unikamy sytuacji, w której w sadzie nie będzie żadnego jabłka).

Przejdźmy do zarządzania sadami.

```
1 void rosnij()
2 {
3     while (1)
4     {
5         sem_wait(&z_dz->skradziono);
6         if((z_dz->gdzie) == 1)
7         {
8             printf("Jestem z sadu A i mam %d jablek\n", ++z_dz->jablko_z_A);
9         }
10        else
11        {
12            printf("Jestem z sadu B i mam %d jablek\n", ++z_dz->jablko_z_B);
13        }
14        sem_post(&z_dz->uroslo);
15    }
16 }
```

5: Funkcja zarządzająca produkcją jabłek w sadzie

Po zwolnieniu semafora `skradziono` sprawdzam, w którym sadzie ostatnio skradziono jabłka i tam przystępuję do produkcji kolejnych. W drugim sadzie jabłek nie powinno zostać, ponieważ reguluję wartość startową semaforów jako 5 zarówno dla złodzieja, jak i sadowników.

W funkcji głównej programu inicjalizuję semafony i inicjalizuję wartości odpowiadające za ilość jabłek w poszczególnych sadach na 5.

```
18 int main()
19 {
20     z_dz = mmap( NULL , sizeof( struct Zasob_Dzielony ) , PROT_READ | PROT_WRITE , MAP_SHARED
21     | MAP_ANONYMOUS , -1 , 0 );
22     sem_init( & z_dz->uroslo , 1 , 5 );
23     sem_init( & z_dz->skradziono , 1 , 5 );
24     z_dz->jablko_z_A = 5;
25     z_dz->jablko_z_B = 5;
26     pid = fork();
27
28     if (pid < 0)
29     {
30         perror ("Niestety cos sie zepsulo :( \n");
31         exit(1);
32     }
33     else if (pid == 0)
34     {
35         rosnij();
36         exit(0);
37     }
38
39     zlodziej();
40     return 1;
41 }
```

6: Funkcja główna

Wartość zmiennej `pid` jest identyfikatorem procesu `pid = 0` odpowiada za proces sadowników i `pid = 1` określa proces złodzieja. Sprawdzam również, czy nie nastąpił niespodziewany błąd związany z uruchomionymi procesami, czyli przypadek, gdy `pid < 0`.

2. Testowanie

Chcąc sprawdzić, czy program rzeczywiście korzysta z procesów. Uruchamiamy go i jednocześnie sprawdzamy aktualnie działające procesy poleceniem:

```
1 $ ps -A
```

7: Polecenie wyświetlające działające procesy

```
./sem
File Edit View Search Terminal Help
Jestem z sadu B i mam 153 jabłek
Kradnę jabłko z sadu B mam już 815604 jabłek z sadu A i 814262 jabłek z sadu B
Jestem z sadu A i mam 749 jabłek
Kradnę jabłko z sadu B mam już 815604 jabłek z sadu A i 814263 jabłek z sadu B
Jestem z sadu B i mam 152 jabłek
Kradnę jabłko z sadu A mam już 815605 jabłek z sadu A i 814263 jabłek z sadu B
Jestem z sadu B i mam 153 jabłek
Kradnę jabłko z sadu A mam już 815606 jabłek z sadu A i 814263 jabłek z sadu B
Jestem z sadu A i mam 748 jabłek
Kradnę jabłko z sadu A mam już 815607 jabłek z sadu A i 814263 jabłek z sadu B
Jestem z sadu A i mam 748 jabłek
Kradnę jabłko z sadu B mam już 815607 jabłek z sadu A i 814264 jabłek z sadu B
Jestem z sadu A i mam 749 jabłek
Kradnę jabłko z sadu B mam już 815607 jabłek z sadu A i 814265 jabłek z sadu B
Jestem z sadu B i mam 152 jabłek
Kradnę jabłko z sadu A mam już 815608 jabłek z sadu A i 814265 jabłek z sadu B
Jestem z sadu B i mam 153 jabłek
Kradnę jabłko z sadu A mam już 815609 jabłek z sadu A i 814265 jabłek z sadu B
Jestem z sadu A i mam 748 jabłek
Kradnę jabłko z sadu A mam już 815610 jabłek z sadu A i 814265 jabłek z sadu B
Jestem z sadu A i mam 748 jabłek
Kradnę jabłko z sadu B mam już 815610 jabłek z sadu A i 814266 jabłek z sadu B
Jestem z sadu A i mam 749 jabłek

9261 ? 00:00:04 python2
9268 ? 00:00:00 gnome-pty-helpe
9269 pts/2 00:00:00 zsh
9271 ? 00:00:00 gsettings <defunct>
9309 pts/2 00:00:01 mocp
9959 ? 00:00:00 kworker/3:0
10010 ? 00:00:00 kworker/2:0
10076 ? 00:00:00 kworker/0:1
10130 ? 00:00:01 kworker/u16:1
10204 ? 00:00:03 chromium-browser
10266 ? 00:00:00 kworker/3:2
10276 ? 00:00:00 kworker/2:2
10333 ? 00:00:00 kworker/u16:0
10390 ? 00:00:00 kworker/2:1
10396 ? 00:00:00 kworker/3:1
10442 ? 00:00:00 chromium-browser
10474 ? 00:00:05 gnome-terminal-
10478 pts/3 00:00:00 zsh
10610 pts/4 00:00:00 zsh
10709 pts/3 00:00:00 sem
10710 pts/3 00:00:00 sem
10711 ? 00:00:00 kworker/u16:3
10712 pts/4 00:00:00 ps
michal@michal-Lenovo-G580:~|
```

Procesy stworzone przez nasz program są widoczne jako:

```
10709 pts/3 00:00:00 sem
10710 pts/3 00:00:00 sem
```