

# Systemy operacyjne

## Sprawozdanie - laboratorium 8 „Komunikacja międzyprocesowa 3”

Andrzej Kołakowski  
296586

### 1) Pamięć dzielona i semaforey

#### 1. Analiza działania programu `shm.c`

Program symuluje dwa konta, które wykonują przelewy. Podzielony jest na 3 procesy:

- **Pm** – proces główny, tworzy dwa inne procesy potomne i czeka na ich zakończenie. Oprócz tego alokuje i zwalnia obszar pamięci współdzielonej oraz w nieskończonej pętli przelewa losową kwotę z konta 1 na konto 2.
- **Pp** – proces potomny. Podobnie jak **Pm** w nieskończonej pętli przelewa losową kwotę z konta 1 na konto 2.
- **Pp2** – proces potomny. W nieskończonej pętli co sekundę wypisuje sumę stanów kont.

Z racji wykorzystania pętli nieskończonych, aby zakończyć działanie program przechwytuje sygnał **SIGINT**.

- Co się dzieje z sumą obydwu kont? Dlaczego?

W trakcie działania programu suma obydwu kont powinna być stała (ile przelejemy z jednego tyle pojawi się na drugim) i równa 1234567. Obserwujemy jednak inny efekt:

```
root@localhost:~/Desktop/so/lab8
File Edit View Search Terminal Help
[root@localhost lab8]# ./shm
pm: poczatek; getpid()=21354
pp: poczatek; getpid()=21355
pm: uzyskujemy id wspólnej pamieci ...
pp2: poczatek; getpid()=21356
pp: uzyskujemy id wspólnej pamieci ...
pp2: uzyskujemy id wspólnej pamieci ...
pp2: konto1+konto2=-7711545873971842
pp2: konto1+konto2=-16742797961073067
pp2: konto1+konto2=-24889766034811067
pp2: konto1+konto2=-32644715214570152
pp2: konto1+konto2=-41872556789673591
```

Problemem jest jednoczesny dostęp do stanu konta z dwóch procesów. Część bitów opisujących saldo może być już zmieniona (a część jeszcze nie) przez pierwszy proces w momencie gdy drugi uzyskuje do nich dostęp.

## 2. Analiza zmian w programie `sem.c`

Do kodu z poprzedniego zadania dodane zostały semaforey. Powinny one pilnować, aby jednocześnie tylko jeden proces modyfikował stany kont.

```
root@localhost:~/Desktop/so/lab8
File Edit View Search Terminal Help
[root@localhost lab8]# ./sem
pm: poczatek; getpid()=23318
pp: poczatek; getpid()=23319
pm: uzyskujemy id wspolnej pamieci ...
pm: uzyskujemy id zbioru semaforow ...
pm: inicjujemy semaforey ...
pp2: poczatek; getpid()=23320
pp: uzyskujemy id wspolnej pamieci ...
pp: uzyskujemy id zbioru semaforow ...
pp2: uzyskujemy id wspolnej pamieci ...
pp2: konto1+konto2=1234567
pp2: konto1+konto2=1234567
pp2: konto1+konto2=1234567
pp2: konto1+konto2=1234567
```

Niestety na skutek błędnej inicjalizacji doprowadziły one do sytuacji, gdy stan konta nigdy nie jest modyfikowany. Można się o tym przekonać dodając wywołania `printf()` w odpowiednich miejscach (plik `sem_debug.c`).

```
root@localhost:~/Desktop/so/lab8
File Edit View Search Terminal Help
[root@localhost lab8]# ./sem_debug
pm: poczatek; getpid()=23343
pm: uzyskujemy id wspolnej pamieci ...
pm: uzyskujemy id zbioru semaforow ...
pm: inicjujemy semafony ...
Petla w pm
pp2: poczatek; getpid()=23345
pp: poczatek; getpid()=23344
pp: uzyskujemy id wspolnej pamieci ...
pp: uzyskujemy id zbioru semaforow ...
Petla w pp
pp2: uzyskujemy id wspolnej pamieci ...
pp2: konto1+konto2=1234567
pp2: konto1+konto2=1234567
pp2: konto1+konto2=1234567
pp2: konto1+konto2=1234567
^CSIGINT: getpid()=23344, konto1+konto2=1234567
SIGINT: getpid()=23343, konto1+konto2=1234567
pm: semop(-1); Bład !!!: Interrupted system call
```

`printf("To sie nigdy nie wypisze\n")` nie zostało ani razu wywołane

Wywołanie w obu procesach zatrzymuje się na funkcji `semop()`. Oczekuje ona na moment, w którym wartość semafora będzie większa lub równa wartości bezwzględnej jej argumentu (-1).

Jest to działanie zgodne z tym opisanym w manualu dla `semop(2)`:

*If semval is greater than or equal to the absolute value of sem\_op, the operation can proceed immediately: the absolute value of sem\_op is subtracted from semval, [...]*

*Otherwise, semncnt (the counter of threads waiting for this semaphore's value to increase) is incremented by one and the thread sleeps until one of the following occurs:*

- *semval becomes greater than or equal to the absolute value of sem\_op: the operation now proceeds, as described above.*
- *The semaphore set is removed from the system: semop() fails, with errno set to EIDRM*
- *The calling thread catches a signal: the value of semncnt is decremented and semop() fails, with errno set to EINTR.*

Wartość semafora nigdy nie będzie większa lub równa wartości bezwzględnej argumentu funkcji `semop()` (-1), bo jako początkowa wartość semafora w `union semun` podane zostało 0. Aby program działał poprawnie konieczna jest zmiana linii:

```
arg.val = 0 na arg.val = 1.
```

Po zmianach program zaczyna działać zgodnie z oczekiwaniami. Gdy jeden proces zmniejszy wartość semafora wtedy drugi nie może zmodyfikować stanu konta, musi czekać na podniesienie wartości semafora przez pierwszy.

## 2) Zadania do samodzielnego wykonania

### Problem czytelników i pisarzy

Dowolna liczba procesów czyta lub pisze do jednego pliku. Dowolne z nich mogą jednocześnie czytać. Jeśli jakiś pisze to inne nie piszą ani nie czytają. W rozwiązaniu nie wolno zagłodzić żadnego wątku.

Problem czytelników i pisarzy posiada kilka rozwiązań.

- Wariant faworyzujący czytelników

Czytelnicy nie mają obowiązku czekania na otrzymanie dostępu do zasobu, jeśli w danym momencie nie otrzymał go pisarz. Ponieważ pisarz może otrzymać tylko dostęp wyłączny, musi czekać na opuszczenie zasobu przez wszystkie inne procesy. Jeśli czytelnicy przybywają odpowiednio szybko, może dojść do zagłodzenia pisarza: w tej sytuacji będzie on w nieskończoność czekał na zwolnienie zasobu przez wciąż napływających nowych czytelników.

- Wariant faworyzujący pisarzy

Czytelnicy nie mogą otrzymać dostępu do zasobu, jeżeli oczekuje na niego pisarz. W tej sytuacji oczekujący pisarz otrzymuje dostęp najwcześniej, jak to jest możliwe, czyli zaraz po opuszczeniu zasobu przez ostatni proces, który przybył przed nim. W tym wariancie może dojść do zagłodzenia oczekujących czytelników.

- Wariant optymalny

Równoczesne wyeliminowanie możliwości zagłodzenia czytelników i pisarzy.

Initialisation	Reader	Writer
<pre>in = Semaphore(1) mx = Semaphore(1) wrt = Semaphore(1) ctr = Integer(0)</pre>	<pre>- Wait in - Wait mx - if (++ctr)==1, then Wait wrt - Signal mx - Signal in  [Critical section]  - Wait mx - if (--ctr)==0, then Signal wrt - Signal mx</pre>	<pre>- Wait in - Wait wrt  [Critical section]  - Signal wrt - Signal in</pre>

Źródło: <https://arxiv.org/ftp/arxiv/papers/1309/1309.4507.pdf>

Gdzie:

- Semaphore(n) – semafor z wartością początkową równą n
- Wait sem – dekrementacja wartości semafora sem
- Signal sem – inkrementacja wartości semafora sem

```
root@localhost:~/Desktop/so/lab8
File Edit View Search Terminal Help
[root@localhost lab8]# ./4
[reader] pid 2161 start
[reader] pid 2161 got value: 0
[reader] pid 2162 start
[reader] pid 2162 got value: 0
[writer] pid 2172 start
[writer] pid 2172 set value: 840
[reader] pid 2163 start
[reader] pid 2163 got value: 840
[reader] pid 2169 start
[reader] pid 2169 got value: 840
[reader] pid 2164 start
[reader] pid 2170 start
[reader] pid 2170 got value: 840
[writer] pid 2171 start
[writer] pid 2171 set value: 529
[reader] pid 2165 start
[reader] pid 2165 got value: 529
[writer] pid 2173 start
[reader] pid 2168 start
[reader] pid 2167 start
[reader] pid 2166 start
[reader] pid 2164 got value: 529
[writer] pid 2173 set value: 242
[reader] pid 2168 got value: 242
[reader] pid 2167 got value: 242
[reader] pid 2166 got value: 242
[root@localhost lab8]#
```

Jak widać program został napisany poprawnie - ani czytelnicy ani pisarze nie ulegają zagłodzeniu.