Systemy operacyjne

Sprawozdanie - laboratorium 8
„Komunikacja międzyprocesowa 3”

Andrzej Kołakowski
296586

1) Pamięć dzielona i semafory

1. Analiza działania programu `shm.c`

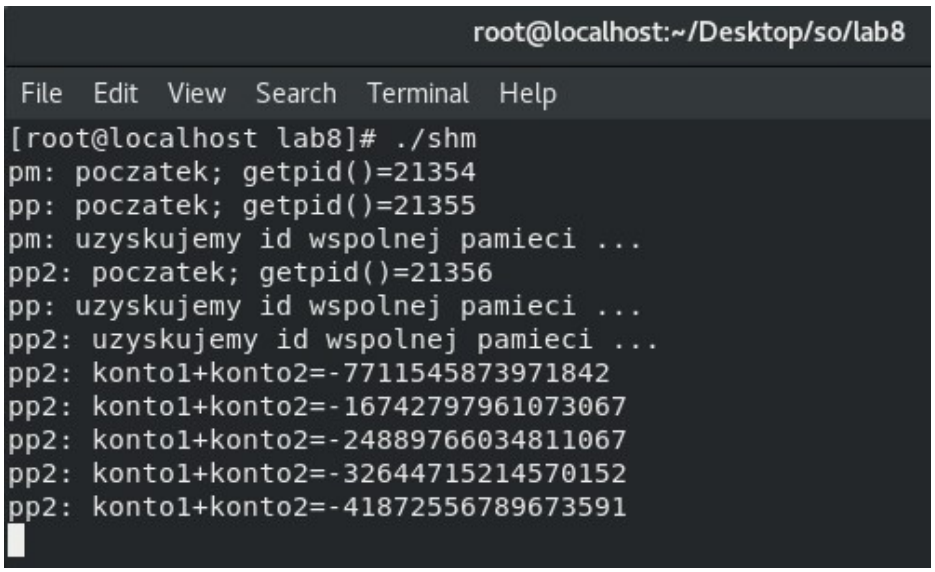
Program symuluje dwa konta, które wykonują przelewy. Podzielony jest na 3 procesy:

- **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?

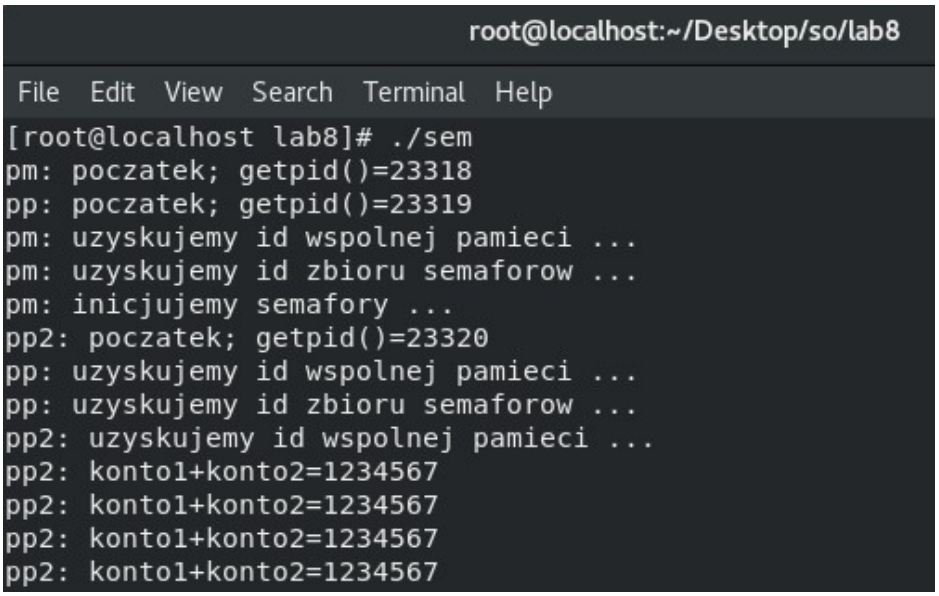
W trakcie działania programu suma obydwu kont powinna być stała (ile przelejemy z jednego tyle pojawi się na drugim) i równa 1234567. Obserwujemy jednak inny efekt:



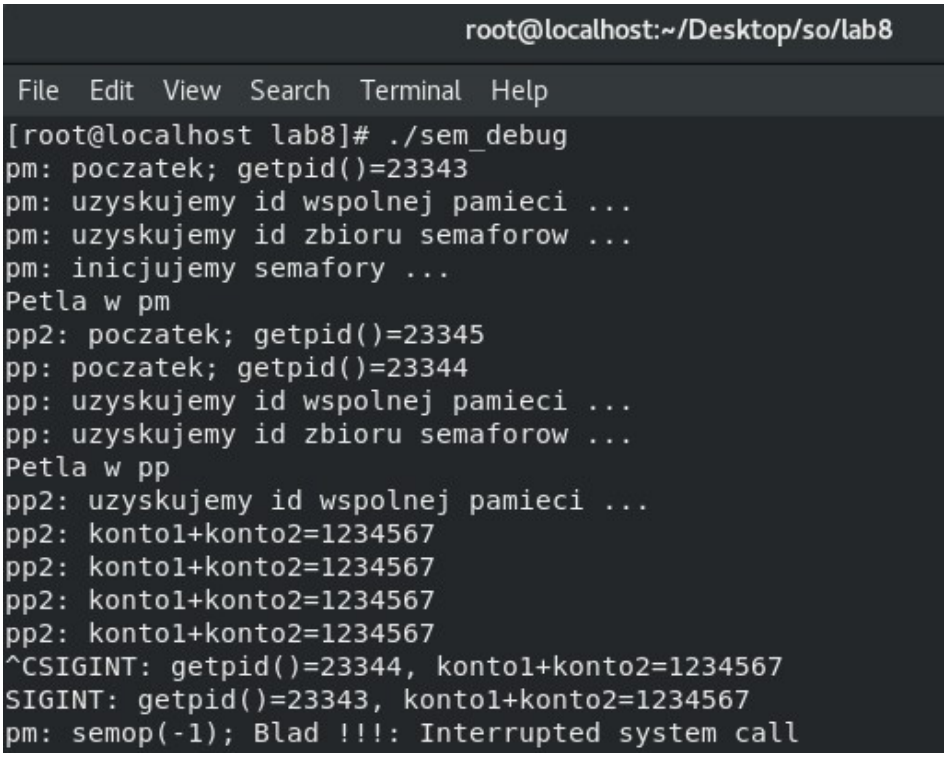
Problemem jest jednoczesny dostęp do stanu konta z dwóch procesów. Część bitów opisujących saldo może być już zmieniona (a część jeszcze nie) przez pierwszy proces w momencie gdy drugi uzyskuje do nich dostęp.

2. Analiza zmian w programie `sem.c`

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



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`).



`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.

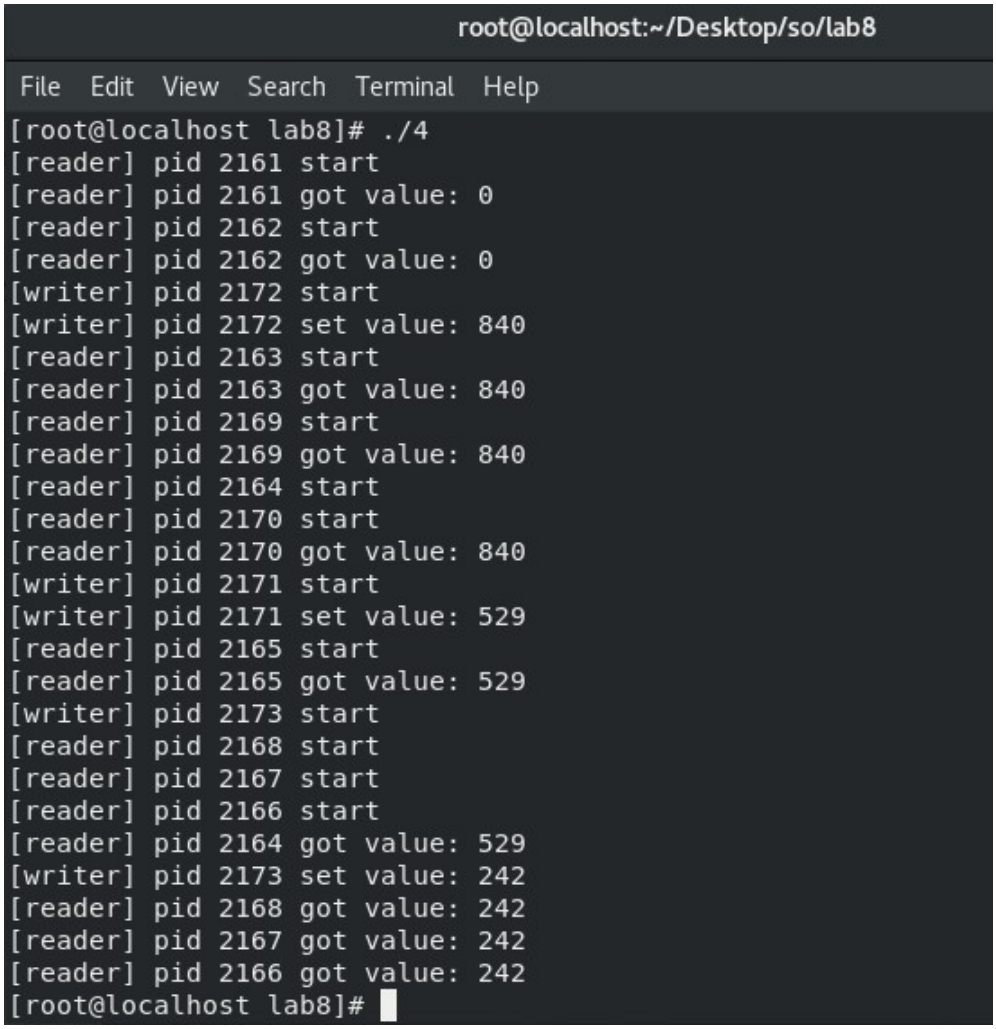
- Wariant optymalny

Równoczesne wyeliminowanie możliwości zagłodzenia czytelników i pisarzy.

Initialisation	Reader	Writer
in = Semaphore(1) mx = Semaphore(1) wrt = Semaphore(1) ctr = Integer(0)	- 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	- Wait in - Wait wrt [Critical section] - Signal wrt - Signal in

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



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