

# Sprawozdanie:

## 8. Zaawansowana komunikacja międzyprocesowa - semaforey i pamięć wspólna

Dominik Bober 303099

7 maja 2020

### 1 Pamięć dzielona i semaforey

Pobierz, rozpakuj, przeanalizuj, skompiluj i uruchom plik: `shm.zip`

Program symuluje dwa konta, które wykonują przelewy.

Co się dzieje z sumą obydwu kont? Dlaczego?

Pobierz, rozpakuj, przeanalizuj, skompiluj i uruchom plik: `sem.zip`

Co się zmieniło?

Programy nie do końca działały u mnie poprawie, ale analizując kod, można zauważyć, że program `shm.zip` tworzy 3 procesy: dwa służące do wykonywania przelewów (`pp1`, `pm`) i jeden wątek kontrolny (`pp2`), który wypisuje sumę kont. Procesy współdzielą pamięć poprzez *Shared memory* (`shm`).

Występuje problem, ponieważ, procesy jednocześnie modyfikują wartości zmiennych `konto1`, `konto2` w strukturze `KONTA`, dlatego może dość do błędów: początkowa suma wartości kont nie jest równa końcowej.

Program `sem.zip` wprowadza funkcjonalność semaforów, które pozwalają na zmianę danych struktury `KONTA` tylko przez jeden proces w danym czasie, dlatego dane pozostają prawidłowe, a proces `pp2` wypisuje stałą sumę. W procesie kontrolnym nie zaimplementowano semafora, ponieważ on jedynie czyta dane, co nie wywoła błędów.

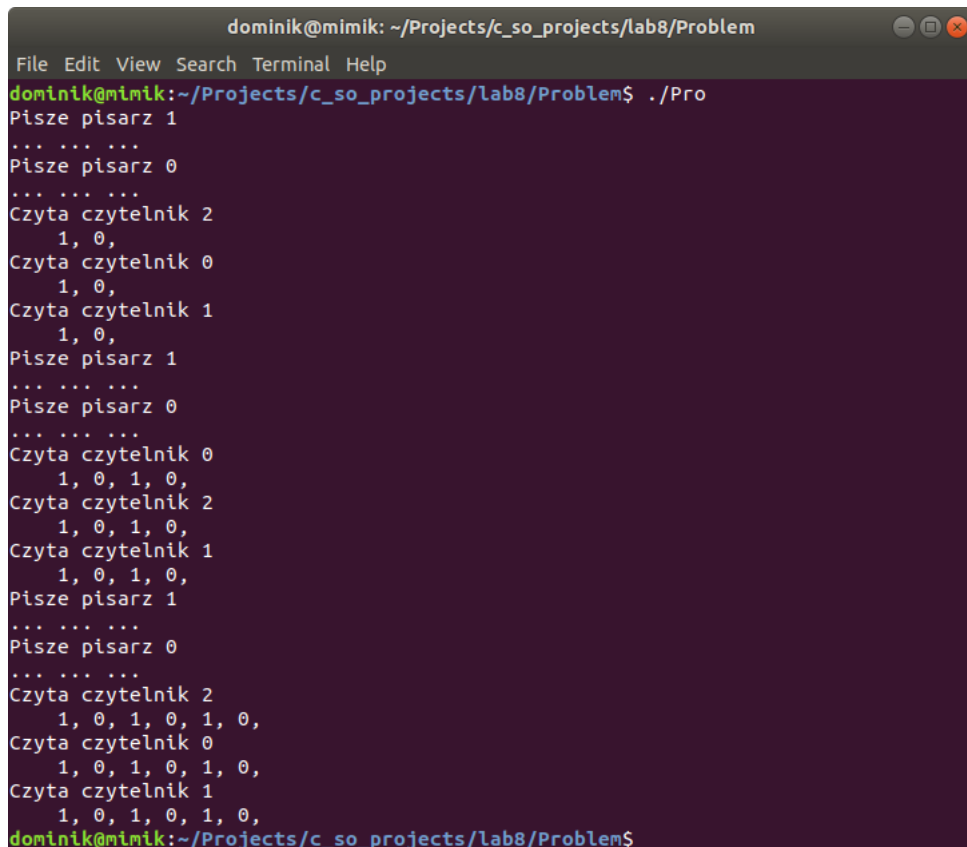
### 2 Zadania

Proszę wybrać jedno z poniższych zadań. Zadania polegają na napisaniu wieloprotocowego programu rozwiązującego dany problem synchronizacji, za pomocą dostępnych w systemie operacyjnym środków wymuszania wyłącznego dostępu (zwykle będą to semaforey). W rozwiązaniach należy unikać zakleszczenia i głodzenia!

- Problem stołówki studenckiej:** Student przebywając w stołówce wywołuje kolejno dwie procedury: `dine` i `leave` (zjada obiad i wychodzi). Po wywołaniu `dine` a przed wywołaniem `leave` student jest w stanie gotowym do wyjścia. By zapewnić studentowi komfort psychiczny nie może on obiadować w samotności. Taki stan ma miejsce, gdy każdy inny student, który wywołał `dine`, wywołał też `leave` zanim rozważany student zakończył procedurę `dine`.
- Problem orangutanów:** Nad głębokim kanionem, gdzieś w Ameryce Południowej, rozpięta jest lina. Używają jej orangutany by przekroczyć kanion. Lina wytrzymuje ciężar pięciu małp a dwa orangutany nie mogą jednocześnie przechodzić po niej z przeciwnych stron kanionu. Po wejściu na linę nie można zawrócić z drogi. Każda małpa oczekująca na przejście musi kiedyś zostać obsłużona.
- Problem golibrody w wersji Hilzera:** Zakład fryzjerski ma trzy stanowiska z trzema fryzjerami, kanapę na cztery osoby i poczekalnię. W zakładzie może przebywać 20 osób jednocześnie a klient nie wejdzie do zapełnionego zakładu. Będąc w środku klient siada na kanapie, jeśli jest wolna, lub czeka w poczekalni w przeciwnym przypadku. Jeśli jest jakiś wolny fryzjer to klient siedzący najdłużej na kanapie zostaje wybrany do strzyżenia, a na jego miejsce siada czekający najdłużej w poczekalni. Gdy klient zostanie ostrzyżony musi zapłacić. Opłatę uiszcza się w jedynej kasie, obsługiwanej przez aktualnie wolnego fryzjera. Fryzjerzy dzielą swój czas na strzyżenie klientów, sen (w przypadku braku klientów) i obsługę kasy.
- 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.

Wybrałem **Problem czytelników i pisarzy**.

Zdecydowałem się nie korzystać z pisania i czytania do pliku, ale zastąpić to pamięcią dzieloną - struktura `library`. Mój program tworzy `WRTS_NUM + RDS_NUM` procesów, są to odpowiednio *pisarze* i *czytelnicy*. Każdy proces wykonuje `ITERS_NUM` operacji, w czasie których *pisarze* modyfikują tablicę `book[]` dopisując do niej swoje id - `this_writerid`, operacja ta jest zabezpieczona semaforem (dodatkowo opóźniam to zadanie funkcjami `sleep()` by lepiej zaopserować działanie zabezpieczeń), *czytelnicy* natomiast modyfikują wartość `reader_num` oraz czytają z `book[]`, obie czynności są zabezpieczane osobnymi semaforami.



```
dominik@mimik: ~/Projects/c_so_projects/lab8/Problem
File Edit View Search Terminal Help
dominik@mimik:~/Projects/c_so_projects/lab8/Problem$ ./Pro
Pisze pisarz 1
... ..
Pisze pisarz 0
... ..
Czyta czytelnik 2
  1, 0,
Czyta czytelnik 0
  1, 0,
Czyta czytelnik 1
  1, 0,
Pisze pisarz 1
... ..
Pisze pisarz 0
... ..
Czyta czytelnik 0
  1, 0, 1, 0,
Czyta czytelnik 2
  1, 0, 1, 0,
Czyta czytelnik 1
  1, 0, 1, 0,
Pisze pisarz 1
... ..
Pisze pisarz 0
... ..
Czyta czytelnik 2
  1, 0, 1, 0, 1, 0,
Czyta czytelnik 0
  1, 0, 1, 0, 1, 0,
Czyta czytelnik 1
  1, 0, 1, 0, 1, 0,
dominik@mimik:~/Projects/c_so_projects/lab8/Problem$ _
```

Rysunek 1: Program wykonujący problem czytelników i pisarzy z parametrami  
`WRTS_NUM = 2`  
`RDS_NUM = 3`  
`ITERS_NUM = 3`