

## 5 Semafor

### 5.1 Pojęcia ogólne

**Semafor** – pierwszy mechanizm synchronizacyjny w językach wysokiego poziomu (Dijkstra, 1965).

`semaphore`      abstrakcyjny typ danych;  
`semaphore S;`    zmienna `S` jest semaforem.

Operacje na semaforze:

- podniesienie (zwolnienie) semafora:    `V`    (hol. *vrijmaken*),
- opuszczenie (zajęcie) semafora:        `P`    (hol. *passeren*).

Poniżej podajemy tzw. *definicje praktyczne* podstawowych rodzajów semaforów.

#### (i) Semafor ogólny (liczący)

$S$  – zmienna całkowita nieujemna.

- $P(S)$ : jeśli  $S > 0$ , to  $S := S - 1$ , w przeciwnym razie wstrzymaj działanie procesu wykonującego tę operację.
- $V(S)$ : jeśli są procesy wstrzymane w wyniku operacji opuszczania semafora  $S$ , to wznów jeden z nich, w przeciwnym razie  $S := S + 1$ .

#### (ii) Semafor binarny

$S = 0/1$  (*false/true*).

Operacje:  $PB(S)$ ,  $VB(S)$ .

*Uwaga:* Semafor binarny nie pamięta liczby wykonanych operacji  $VB$ ;  
 czasem przyjmuje się, że  $VB(S = 1) \rightarrow$  błąd!

#### (iii) Semafor dwustronnie ograniczony

$0 \leq S \leq N$ ,

$$\left. \begin{array}{l} PD(S) - \text{opuszczanie} \\ VD(S) - \text{podnoszenie} \end{array} \right\} \text{symetryczne,}$$

tzn.  $VD(S = N)$  działa jak  $PD(S = 0)$ .

#### (iv) Semafor uogólniony

Zmiana  $S$  o dowolną liczbę naturalną  $n$ .

- $PG(S, n)$ : jeśli  $S \geq n$ , to  $S := S - n$ , w przeciwnym razie wstrzymaj działanie procesu wykonującego tę operację.
- $VG(S, n)$ : jeśli są procesy wstrzymane w wyniku wykonywania operacji  $PG(S, m)$ , przy czym  $m \leq S + n$ , to wznów jeden z nich i  $S := S - m + n$ , w przeciwnym razie  $S := S + 1$ .

**PRZYKŁAD: Wzajemne wykluczanie**

Zadanie zsynchronizowania procesów wymagających *wyłącznego dostępu* do pewnego *zasobu dzielonego* (fragment procesu, w którym korzysta on z zasobu dzielonego nazywa się *sekcją krytyczną* tego procesu).

```
binary semaphore S = 1; /* Semafor binarny do synchronizacji procesow */

/* Proces P_i */
do {
    /* Wlasne sprawy */
    PB(S);
    /* Sekcja krytyczna */
    VB(S);
    /* Reszta */
} while (1);
```

**5.2 Semaforey standardu POSIX**

W nowszych wersjach systemów uniksowych (np. w Linuksie od wersji jądra 2.6) dostępne są semaforey dla procesów i wątków standardu POSIX. Mają one prostszy i wygodniejszy interfejs niż wcześniejsze semaforey standardu UNIX System V (ciągle dostępne w systemach uniksowych), choć mają w stosunku do nich mniejszą funkcjonalność. Implementują one typ semafora ogólnego (liczącego), tzn. można je podnosić i opuszczać tylko o wartość 1. Semaforey te dostępne są w dwóch postaciach: **semaforey nienazwane** (ang. *unnamed semaphores*) i **semaforey nazwane** (ang. *named semaphores*). Do opuszczania i podnoszenia obu tych rodzajów semaforów służą te same funkcje, natomiast różne funkcje przeznaczone są do ich tworzenia i usuwania.

→ UWAGA: Aby można było używać funkcji semaforowych w programach w języku C, należy je linkować z opcją: **-pthread**.

Semafor nienazwany nie posiada nazwy, w związku z tym, aby mógł być użyty do synchronizacji procesów czy wątków, musi być umieszczony w ich pamięci wspólnej. Z tego względu ten typ semaforów jest wygodniejszy w zastosowaniach do synchronizacji wątków jednego procesu, jako że wątki takie współdzielą pamięć w ramach swojego procesu.

Przed użyciem semafor nienazwany musi być zainicjowany przy użyciu funkcji **sem\_init**. Funkcja ta inicjalizuje semafor nienazwany pod adresem **sem** wartością przekazywaną

Pliki włączane	<semaphore.h>		
Prototyp	int sem_init(sem_t *sem, int pshared, unsigned int value);		
Zwracana wartość	Sukces	Porażka	Czy zmienia errno
	0	-1	Tak

przez parametr **value**. Jeżeli przez parametr **pshared** zostanie przekazana wartość 0, to semafor będzie współdzielony przez wątki w ramach procesu i musi być umieszczony pod

adresem widocznym dla wszystkich takich wątków (np. zmienna globalna lub zmienna utworzona dynamicznie na stercie). Natomiast dla niezerowej wartości `pshared`, semafor jest współdzielony przez procesy i musi być umieszczony w pamięci dzielonej (ang. *shared memory*) tych procesów, utworzonej przy pomocy mechanizmów standardu POSIX lub Systemu V.

Semafor nienazwany może być usunięty przy użyciu funkcji `sem_destroy`. Usuwa ona semafor pod adresem `sem` zainicjowany wcześniej funkcją `sem_init`. Usunięcie semafora,

Pliki włączane	<semaphore.h>		
Prototyp	<code>int sem_destroy(sem_t *sem);</code>		
Zwracana wartość	Sukces	Porażka	Czy zmienia <code>errno</code>
	0	-1	Tak

który jest używany przez jakiś inny proces lub wątek prowadzi do niezdefiniowanego zachowania. Semafor nienazwany powinien być usunięty funkcją `sem_destroy` zanim pamięć, w której się znajduje zostanie zdealokowana – w przeciwnym razie może dojść do wycieku zasobów.

Semafor nazwany jest identyfikowany nazwą postaci `/somename`, tj. napisem zakończonym znakiem zerowym, który rozpoczyna się ukośnikiem `/` i zawiera jeden lub więcej znaków (maksymalnie `NAME_MAX - 4`, tj. 251) nie będących ukośnikami. W systemie Linux semafony nazwane tworzone są w wirtualnym systemie plików (ang. *virtual file system - VFS*) z nazwami postaci `sem.somename` i zwykle montowane pod `/dev/shm`.

Różne procesy mogą mieć dostęp do tego samego semafora przez przekazanie tej samej nazwy do funkcji `sem_open`. Funkcja ta tworzy lub otwiera istniejący semafor nazwany

Pliki włączane	<fcntl.h>, <sys/stat.h>, <semaphore.h>		
Prototyp	<code>sem_t *sem_open(const char *name, int oflag);</code> <code>sem_t *sem_open(const char *name, int oflag, mode_t mode, unsigned int value);</code>		
Zwracana wartość	Sukces	Porażka	Czy zmienia <code>errno</code>
	adres semafora	SEM_FAILED	Tak

o nazwie przekazanej przez parametr `name` i w przypadku sukcesu zwraca jego adres jako wskaźnik typu `sem_t*`, natomiast w przypadku porażki zwraca wartość `SEM_FAILED`. Funkcja `sem_open` występuje w dwóch postaciach: pierwsza z nich ma dwa parametry i służy do otwierania istniejącego semafora o danej nazwie, natomiast druga ma cztery parametry i służy do tworzenia nowego semafora nazwanego. Aby utworzyć nowy semafor, należy parametr `oflag` ustawić na `O_CREAT` – jeżeli dodatkowo zostanie on połączony sumą bitową z `O_EXCL`, to funkcja zakończy się błędem jeśli semafor o danej nazwie już istnieje. W przypadku użycia flagi `O_CREAT`, do funkcji `sem_open` muszą być przekazane dwa dodatkowe argumenty: prawa dostępu do nowego semafora przez parametr `mode` (tak jak dla funkcji `open` dla plików) oraz wartość początkowa semafora przez parametr `value`. Jeżeli zostanie użyta flaga `O_CREAT`, a semafor o danej nazwie już istnieje, to parametry `mode` i `value` są ignorowane.

Jeżeli proces zakończył używanie semafora nazwanego, to powinien go zamknąć wywołując funkcję `sem_close` – zwalnia ona zasoby przydzielone do semafora o adresie przekazanym przez parametr `sem` w danym procesie.

Pliki włączane	<semaphore.h>		
Prototyp	<code>int sem_close(sem_t *sem);</code>		
Zwracana wartość	Sukces	Porażka	Czy zmienia <code>errno</code>
	0	-1	Tak

Kiedy wszystkie procesy korzystające z jakiegoś semafora nazwanego przestaną go używać, to można go on usunąć przy użyciu funkcji `sem_unlink`, przekazując jego nazwę przez parametr `name`. Semaforey nazwane standardu POSIX są obiektami trwałymi jądra (ang. *kernel persistence*), więc jeśli jakiś semafor nie zostanie usunięty funkcją `sem_unlink`, to będzie istniał aż do zamknięcia systemu (ang. *shutdown*).

Pliki włączane	<semaphore.h>		
Prototyp	<code>int sem_unlink(const char *name);</code>		
Zwracana wartość	Sukces	Porażka	Czy zmienia <code>errno</code>
	0	-1	Tak

Do wykonywania operacji podnoszenia  $V(S)$  i opuszczania  $P(S)$  o wartość 1 semaforów zarówno nazwanych, jak i nienazwanych służą odpowiednio funkcje `sem_post` i `sem_wait`. Jeżeli istnieją procesy wstrzymane pod semaforem, to wywołanie funkcji `sem_post` spo-

Pliki włączane	<semaphore.h>		
Prototyp	<code>int sem_post(sem_t *sem);</code> <code>int sem_wait(sem_t *sem);</code>		
Zwracana wartość	Sukces	Porażka	Czy zmienia <code>errno</code>
	0	-1	Tak

woduje uruchomienie jednego z nich. Z kolei wywołanie funkcji `sem_wait` dla semafora, którego wartość wynosi zero spowoduje wstrzymanie wywołującego procesu do momentu, aż wartość ta stanie się większa od zera.

Pliki włączane	<semaphore.h>		
Prototyp	<code>int sem_getvalue(sem_t *sem, int *sval);</code>		
Zwracana wartość	Sukces	Porażka	Czy zmienia <code>errno</code>
	0	-1	Tak

Do uzyskania informacji o wartości semafora w danej chwili służy funkcja `sem_getvalue`. Za pośrednictwem wskaźnika `sval` zwraca ona wartość semafora o adresie przekazanym przez parametr `sem`. Kiedy istnieją jakieś procesy wstrzymane pod semaforem, to standard POSIX.1-2001 dopuszcza dwie możliwości dla wartości zwracanej przez `sval`: zero

lub ujemną liczbę, której wartość bezwzględna odpowiada liczbie wstrzymanych procesów. Linux stosuje to pierwsze podejście.

Oprócz `sem_wait` istnieją jeszcze dwie funkcje do opuszczania semafora: `sem_trywait` i `sem_timedwait`. Informacje o nich, jak też więcej szczegółów na temat semaforów standardu POSIX można znaleźć w rozdziale podręcznika systemowego *on-line*: `man sem_overview`.

## ĆWICZENIE 6: WZAJEMNE WYKLUCZANIE DLA PROCESÓW: SEMAFORY

Przy pomocy **semaforów nazwanych** standardu POSIX zaimplementować zadanie **wzajemnego wykluczania** dla **procesów** podane w pseudokodzie na końcu podrozdziału [5.1](#). Czas operacji na *wspólnym zasobie* symulować używając np. funkcji `sleep`. Dla demonstracji poprawności działania programu użyć odpowiednich *komunikatów* wypisywanych przez poszczególne procesy *przed*, *w trakcie* i *po sekcji krytycznej* oraz funkcji podającej *wartość semafora*. Pamiętać o *zainicjowaniu semafora* odpowiednią wartością zaraz po jego utworzeniu.

- Stworzyć *własną bibliotekę* prostych w użyciu funkcji do tworzenia, otwierania, uzyskiwania wartości, operowania, zamykania i usuwania semafora – korzystających z odpowiednich funkcji systemowych/bibliotecznych i zawierających obsługę błędów.
- Napisać specjalny program do *powielania procesów* realizujących wzajemne wykluczanie – w oparciu o funkcje `fork` i `exec1p` (*nazwę programu* do inicjowania procesów, *liczbę procesów* oraz *liczbę sekcji krytycznych/prywatnych* każdego procesu przekazywać przez *argumenty* programu „powielacza”). Program ten powinien na początku utworzyć i zainicjować semafor, a na końcu – kiedy wszystkie jego procesy potomne zakończą swoje działanie – usunąć go.
- Semafor usuwać w funkcji rejestrowanej przez funkcję `atexit` (zwrócić uwagę, kiedy należy wywołać funkcję `exit`, a kiedy `_exit`). Dodać również możliwość usuwania semafora w funkcji obsługi sygnału `SIGINT`, na wypadek gdyby program trzeba było zakończyć sekwencją klawiszy `Ctrl-C`.
- W celu weryfikacji poprawności wzajemnego wykluczania procesów niech program „powielacz” na samym początku stworzy plik tekstowy `numer.txt` i wpisze w nim numer o wartości 0 (zero), a program implementujący *wzajemne wykluczanie* w swojej *sekcji krytycznej* odczytuje wartość zapisanego w tym pliku numeru, następnie zwiększa go o 1 i po losowej (np. funkcja `rand`) chwili czasu zapisuje z powrotem do pliku. Po zakończeniu działania wszystkich procesów realizujących wzajemne wykluczanie proces „powielacz” powinien sprawdzić poprawność końcowego numeru zapisanego w pliku i wypisać odpowiedni komunikat.  
→ Sprawdzić, jaka będzie końcowa wartość numeru bez użycia *sekcji krytycznej*.