Instrukcja do laboratorium Systemów Operacyjnych

(semestr drugi)

Ćwiczenie piąte (jedne zajęcia)

Temat: Semafory

Opracowanie:

mgr inż. Arkadiusz Chrobot

dr inż. Grzegorz Łukawski

Wprowadzenie

1. Semafory

Synchronizacja komunikujących się ze sobą procesów jest jednym z najważniejszych zagadnień w dziedzinie programowania współbieżnego, dlatego twórcy mechanizmów IPC dodali do nich semafory mimo, że nie służą one do przekazywania komunikatów. Semafor jest abstrakcyjnym typem danych, dla którego zdefiniowano dwie niepodzielne operacje: P – oczekuj i V – sygnalizuj. Rozważmy najprostszy rodzaj semafora, czyli semafor binarny. Jest to zmienna przyjmująca tylko dwie wartości: 0 i 1, dostępna tylko poprzez operacje V i P. Operacja P działa następująco: jeśli wartość semafora jest większa od zera, to zmniejsz ją o jeden. Jeśli wartość ta wynosi zero, to zawieś wykonanie procesu. Wykonanie operacji V przebiega w ten sposób: jeśli inny proces został zawieszony w oczekiwaniu na semafor, to wznów jego wykonanie. Jeśli żaden proces nie został zawieszony w oczekiwaniu na semafor, to zwiększ wartość semafora o jeden. Z powyższego opisu można wyciągnąć dwa wnioski. Po pierwsze semafor musi być dostępny dla wszystkich procesów ubiegających się o dostęp do zasobu strzeżonego przez ten semafor, a więc nie może być zmienną umieszczoną w przestrzeni adresowej do której dostęp ma tylko jeden proces. Po drugie zmiana wartości semafora musi się odbywać w sposób niepodzielny, tzn. jeśli proces rozpocznie operacje zmiany wartości semafora, to nie może zostać wywłaszczony i żaden inny proces w tym samym przedziale czasu nie może manipulować semaforem. Aby obydwa wymagania były spełnione semafory są tworzone w przestrzeni adresowej jądra i dostępne poprzez odpowiednie wywołania systemowe. Mechanizm IPC dostarcza interfejsu dla tych wywołań systemowych dla procesów pracujących w przestrzeni użytkownika. Semafory w Linuksie (Uniksie) mogą przyjmować większy zbiór wartości niż tylko 0 i 1. Należy oczywiście zadbać, aby wszystkie procesy, których działanie chcemy synchronizować przestrzegały odpowiedniego protokołu dostępu do zasobu (tzn. należy nie wolno dopuścić do sytuacji, w której proces uzyskuje dostęp do zasobu z pominięciem semafora lub kiedy nieodpowiednio posługuje się semaforem).

2. Funkcje i struktury danych

- funkcja semget() - funkcja ta tworzy zbiór semaforów i zwraca jego identyfikator lub zwraca identyfikator istniejącego zbioru semaforów. Jako argumenty wywołania przyjmuje klucz, który może być zwrócony przez funkcję ftok() (patrz poprzednia instrukcja), liczbę semaforów w zbiorze i flagi związane ze sposobem tworzenia i prawami dostępu do semafora. Szczegóły: man semget.

- funkcja semop() - umożliwia przeprowadzenie operacji na wartości semafora w sposób niepodzielny. Funkcja ta pobiera trzy argumenty. Pierwszym argumentem przyjmowanym przez tę funkcję jest identyfikator zbioru semaforów. Następnym jest wskaźnik na tablicę zawierającą elementy o następującej strukturze:

```
struct sembuf {
    unisgned sem_num;
    short sem_op;
    short sem_flg;
};
```

Pole *sem_num* zawiera numer semafora w zbiorze (semafory są numerowane od zera), którego ma dotyczyć operacja. Pole *sem_op* określa jaka operacja zostanie na semaforze przeprowadzona. Jeśli wartość tego pola jest dodatnia, to zostanie ona dodana do bieżącej wartości semafora. Jeśli wartość ta wynosi zero, to proces wykonujący tę operację będzie czekał do czasu aż semafor osiągnie wartość zero. Jeśli wartość pola *sem_op* jest ujemna, to proces będzie czekał do momentu kiedy semafor osiągnie wartość większą lub równą bezwzględnej wartości pola *sem_op*, a następnie dodaje tą ujemną wartość do bieżącej wartości semafora. Pole *sem_flg* może przyjmować dwie wartości SEM_UNDO i IPC_NOWAIT. Ostatnia flaga oznacza, że proces nie będzie czekał na zakończenie operacji, natomiast pierwsza oznacza, że operacja zostanie automatycznie cofnięta po zakończeniu procesu, który ją wykonał. Trzeci argument funkcji *semop ()* określa ile jest elementów w tablicy, której wskaźnik jest przekazywany jako drugi argument wywołania funkcji. Szczegóły: man semop.

- funkcja *semctl()* służy do sterowania zbiorem semaforów. Jako pierw-

szy argument pobiera identyfikator zbioru semaforów. Drugim argumentem jest jest numer semafora w zbiorze (patrz opis funkcji *semop*). Trzeci argument określa rodzaj operacji jaka ma być wykonana. Wartość IPC_RMID powoduje usunięcie zbioru semaforów z systemu (drugi argument jest ignorowany). Wartość GETVAL powoduje, ze wywołanie funkcji *semctl()* zwróci wartość określonego w jej argumentach semafora. Wartość SETVAL może być użyta do zainicjowania semafora określoną wartością. Jej użycie wymaga przekazania do funkcji czwartego argumentu o typie określonym przez następującą unię:

```
union semun {
   int val;
   struct semid_ds  *buff;
   unsigned short   *array;
   struct seminfo *__buf;
} arg;
```

Pierwsze pole jest wykorzystywane przez operację SETVAL, drugie pole jest wykorzystywane przez IPC_STAT i IPC_SET, trzecie przez SETALL i GETALL, natomiast ostatnie jest specyficzne dla Linuksa i używane przez IPC_INFO. Pełny opis operacji wymienionych w opisie unii oraz szczegóły dotyczące działanie *semctl()* znajdują się w podręczniku systemowym: man semctl.

Do zarządzania semaforami można użyć tych samych poleceń systemowych co w przypadku kolejek komunikatów, tj. ipcs i ipcrm.

Zadania

1. Napisz dwa programy. Pierwszy stworzy semafor i zainicjuje go wartością dodatnią, a następnie poczeka, aż drugi program ustawi wartość tego semafor na zero i dopiero wtedy zakończy się.
2. Napisz program, który stworzy zbiór dziesięciu semaforów, o wartości początkowej równej jeden, a następnie stworzy dziesięć procesów potomnych, które wstępnie zostaną uśpione na sekundę, a następnie ustawią wartość odpowiadającego im semafora na zero. Proces rodzicielski może się zakończyć dopiero wtedy, kiedy ostatni semafor ze zbioru osiągnie wartość zero.

3. Zmodyfikuj zadanie drugie tak, aby tworzony był jeden semafor o wartości początkowej zero, a procesy potomne zwiększały go o jeden, w określonym porządku. Proces pierwszy bezwarunkowo zwiększy wartość semafora, proces drugi będzie czekał aż semafor będzie miał wartość dwa i zwiększy ją do trzech, itd. Wartość końcowa semafora powinna wynosić pięć.
4. Zademonstruj synchronizację operacji zapisu i odczytu dla łącza nazwanego, przy pomocy semaforów.
5. Pokaż w jaki sposób może dojść do zakleszczenia (*ang. deadlock*) procesów synchronizowanych przy pomocy semaforów.
6. Zademonstruj działanie operacji SEM_UNDO.
7. Zademonstruj działanie operacji SETALL, GETALL, IPC_STAT, GETPID, GETZCNT.
8. Stwórz kolejkę komunikatów z której będzie korzystało kilka procesów, z których część będzie pisarzami, a część czytelnikami. Operację zapisu i odczytu z tej kolejki należy zsynchronizować za pomocą dwóch semaforów, tzn.: semafor pierwszy będzie zajmowany przez proces przed operacją odczytu, a zwalniany zaraz po jej wykonaniu, natomiast semafor drugi będzie zajmowany przez proces przed operacją zapisu i podobnie jak wyżej, zwalniany po jej wykonaniu. Załóż, że wiele procesów może odczytywać współbieżnie dane z kolejki, ale tylko jeden może w danym czasie do niej pisać.
9. Napisz program, który podzieli się na dwa procesy komunikujące się przez łącze nienazwane (pipe). Oba te procesy będą również miały dostęp do wspólnego semafora, któremu będzie nadana wartość początkowa większa od zera. Proces pierwszy będzie wysyłał liczby od 1 do 10 do procesu drugiego. Proces drugi będzie wyświetlał je na ekran, a po otrzymaniu liczby 10 wyzeruje semafor. Po wyzerowaniu semafora oba procesy powinny się zakończyć.

Uwaga: We wszystkich programach tuż przed zakończeniem ich działania wszystkie semafory i inne zasoby IPC z jakich one korzystają powinny zostać usunięte.