Temat zajęć: Mechanizmy IPC: semafory

Czas realizacji zajęć:	*90 min.*
Zakres materiału, jaki zostanie zrealizowany podczas zajęć:	Zasada działania semaforów binarnych i uogólnionych, tworzenie semaforów, operacje na semaforach, parametry semaforów i ich inicjowanie, przykłady podstawowych problemów synchronizacji, implementacja przykładowych programów z synchronizacją wykorzystujących semafory

I. Semafory

Semafory są strukturami danych wspólnie użytkowanymi przez kilka procesów. Najczęściej znajdują one zastosowanie w synchronizowaniu działania kilku procesów korzystających ze wspólnego zasobu, przez co zapobiegają niedozwolonemu wykonaniu operacji na określonych danych jednocześnie przez większą liczbę procesów. Przez odpowiednie wykorzystywanie semaforów można zapobiec sytuacji w której wystąpi *zakleszczenie* (ang. *deadlock*) lub *zagłodzenie* (ang. *starvation*). Do zakleszczenia dochodzi wtedy, gdy każdy z procesów przetrzymuje zasób potrzebny innemu, czekając aż inny proces zwolni swój zasób – innymi słowy procesy wzajemnie na siebie czekają, a algorytmy uniemożliwiają im zwolnienie powstałych w ten sposób blokad. Zagłodzenie natomiast występuje wtedy, gdy gdy jakiś proces bezskutecznie próbuje uzyskać dostęp do współdzielonego zasobu.

Semafor skojarzony z danym zasobem jest początkowo ustawiany na wartość równą liczbie dostępnych zasobów tego typu. Proces, który żąda zasobu musi najpierw sprawdzić wartość wartość skojarzonego z tym zasobem semafora - dodatnia wartość semafora oznacza dostępność zasobu. Przed rozpoczęciem korzystania z zasobu proces zmniejsza wartość semafora w sposób *niepodzielny.* Zerowa wartość semafora oznacza, że nie ma wolnych zasobów i proces musi czekać aż proces zajmujący zasób przestanie z niego korzystać i zwiększy wartość semafora zwalniając zasób. Kiedy zasób zostanie zwolniony, pozostałe procesy, które na niego czekały są o tym powiadamiane przez system.

Wyróżniamy semafory **binarne**,sterujące dostępem do jednego zasobu i **uogólnione,** sterujące dostępem do wielu zasobów. Semafory binarne mogą przyjmować tylko dwie wartości: 0 i 1, natomiast semafory uogólnione przyjmują różne, nieujemne wartości.

Od strony implementacyjnej semafor jest nieujemną liczbą całkowitą przechowywaną w jądrze systemu, do której dostęp jest dostęp jest możliwy za pośrednictwem wywołań systemowych: **semget**, **semctl** i **semop**. Wywołania te powodują domyślne blokowanie procesów w sytuacjach, gdy wartość semafora wskazuje, że żądany zasób jest niedostępny. Kiedy zasób staje się dostępny system powiadamia o tym kolejny proces z kolejki oczekujących procesów.

W systemie Unix semafory są tworzone jako w postaci zestawów składających się z jednego lub kilku pojedynczych semaforów. W momencie tworzenia zestawu semaforów powstaje struktura danych o nazwie `semid_ds`, której system używa do identyfikowania semaforów i operowania na nich. Struktura ta jest zdefiniowana w pliku `<sys.sem.h>`.

Dodatkowo, z każdym semaforem z zestawu jest skojarzona struktura typu `sem`:

```
struct sem{
ushort semval     // wartość semafora
pid_t sempid      // identyfikator procesu ostatnio wykonującego operację na semaforze
ushort semncnt    //liczba procesów, które czekają aż wartość semafora będzie zwiększona
ushort semzcnt    //liczba procesów które czekają aż semafor osiągnie wartość 0
}
```

Funkcje umożliwiające synchronizację za pomocą semaforów zdefiniowane są w plikach `<sys/types.h>`, `<sys/ipc.h>` oraz `<sys/sem.h>`.

II. Funkcje systemowe obsługujące zestawy semaforów i ich argumenty.

- **`int semget (key_t key, int nsems, int semflags)`**

  Wartości zwracane:
  poprawne wykonanie funkcji: identyfikator zestawu semaforów
  zakończenie błędne: -1

  Argumenty funkcji:
  key –liczba, która identyfikuje zestaw semaforów
  nsems – liczba semaforów w zestawie, używana przez system podczas rezerwowania miejsca na strukture `sem`, poindeksowana od 0
  semflags – wskazuje prawa dostępu i/lub dodatkowe opcje dla tworzonego zestawu semaforów (IPC_CREAT, IPC_EXCL, które połączone są operatorem **OR**)

  UWAGI:
  Utworzenie lub pobranie identyfikatora tablicy semaforów. Funkcja tworzy tablicę składającą się z *nsems* semaforów, jeśli tablica o kluczu *key* jeszcze nie istnieje i zwraca na podstawie klucza identyfikator tej tablicy. Parametr *semflg* umożliwia przekazanie praw dostępu do tablicy semaforów oraz pewnych dodatkowych flag definiujących sposób jej tworzenia (np. IPC_CREAT), połączonych z prawami dostępu operatorem sumy bitowej.
  IPC_PRIVATE nie jest flagą tylko szczególną wartością *key_t*. Jeśli wartość ta zostanie użyta jako wartość klucza, to system uwzględni jedynie bity uprawnień parametru *msgflg* i zawsze będzie próbować utworzyć nową kolejkę.

- **`int semctl (int semid, int semnum ,int cmd, union semun ctl_arg)`**

  Wartości zwracane:
  poprawne wykonanie funkcji: żądana wartość całkowita
  zakończenie błędne: -1

  Argumenty funkcji:
  semid – identyfikator zestawu semaforów
  semnum – liczba semaforów w zestawie
  cmd – całkowitoliczbowa wartość reprezentująca polecenie
  ctl_arg – w zależności od wykonywanej przez funkcję czynności jest to wskaźnik do struktury `semid_ds`, lub adres bazowy tablicy liczb typu `short int`

  UWAGI:
  Zastosowania funkcji **semctl**:
  - ustawienie początkowej wartości semafora
  - zbadanie zmiany właściciela semafora, praw dostępu do niego, czasu ostatniej zmiany, ilości procesów oczekujących na semafor i identyfikatora procesu ostatnio zmieniającego wartość semafora, itd.
  - Operacje kontrolne na pojedynczym semaforze lub ich zestawie.

  Drugim argumentem funkcji może być wartość 0, która pojawia się wówczas, gdy funkcja **semctl** ma wykonać operację, podczas której liczba semaforów w zestawie nie ma znaczenia.
  Wartość pola *cmd* może należeć do jednej z trzech grup:
  1. tradycyjne operacje IPC
     - IPC_STAT – zwraca wartości struktury `semid_ds` dla semafora (lub zestawu) o

identyfikatorze *semid*, informacja jest umieszczana w strukturze wskazywanej przez 4 argument funkcji **semctl**
- IPC_SET – modyfikuje wartości struktury `semid_ds`
- IPC_RMID – usuwa zestaw semaforów o identyfikatorze *semid* z systemu

2. operacje na pojedynczym semaforze (dotyczą semafora określonego przez *semnum*):
   - GETVAL – zwraca wartość semafora (*semval*), wskazywanego argumentem *semnum*
   - SETVAL – nadaje wartość określoną czwartym argumentem funkcji semaforowi o numerze *semnum*
   - GETPID – zwraca wartość *sempid*
   - GETNCNT – pobranie liczby procesów oczekujących na to, aż semafor wskazywany przez *semnum* zwiększy swoją wartość
   - GETZCNT – pobranie liczby procesów oczekujących na to, aż semafor wskazywany przez *semnum* osiągnie wartość zero
3. operacje na wszystkich semaforach:
   - GETALL – pobranie bieżących parametrów całego zestawu semaforów i zapisanie uzyskanych wartości w tablicy wskazanej czwartym argumentem funkcji
   - SETALL – zainicjowanie wszystkich semaforów z zestawu wartościami przekazanymi w tablicy określonej przez wskaźnik przekazany czwartym argumentem funkcji

Ostatni argument funkcji jest unią:

```
union semun (
int val;
struct semid_ds *buf;
unsigned short *array
)
```

- **`int semop (int semid, stuct sembuf *sops, unsigned nsops)`**

Wartości zwracane:
poprawne wykonanie funkcji: 0
zakończenie błędne: -1

Argumenty funkcji:
semid – identyfikator zestawu semaforów
sops – wskaźnik na adres bazowy tablicy operacji semaforowych, które zostaną wykonane na zestawie semaforów określonym wartością *semid*
nsops – liczba elementów tablicy operacji semaforowych

UWAGI:
Wykonanie operacji semaforowej. Operacja semaforowa może być wykonywana jednocześnie na kilku semaforach w tej samej tablicy identyfikowanej przez *semid*. Każdy element tablicy opisuje jedną operację semaforową i ma następującą strukturę:

```
struct sembuf {
short sem_num;        /* numer semafora – od 0 */
short sem_op;         /* operacja semaforowa */
short sem_flg;        /* flagi operacji */
};
```

Pole *sem_op* zawiera wartość, która zostanie dodana do zmiennej semaforowej pod warunkiem, że zmienna semaforowa nie osiągnie w wyniku tej operacji wartości mniejszej od 0. Dodatnia liczba całkowita oznacza zwiększenie wartości semafora (co z reguły oznacza zwolnienie zasobu), ujemna wartość *sem_op* oznacza zmniejszenie wartości semafora (próbę pozyskania zasobu), a 0 – testowanie, czy semafor jest równy 0, czyli czy wszystkie zasoby są zajęte.

Funkcja **semop** podejmuje próbę wykonania wszystkich operacji wskazywanych przez *sops*. Gdy chociaż jedna z operacji nie będzie możliwa do wykonania nastąpi blokada procesu lub błąd wykonania funkcji **semop**, zależnie od ustawienia flagi IPC_NOWAIT i żadna z operacji semaforowych zdefiniowanych w tablicy *sops* nie zostanie wykonana. Flagi określają dodatkowe cechy operacji (np. IPC_NOWAIT – jeżeli operacja na semaforze nie zostać przeprowadzona wywołanie zostaje natychmiast zakończone, SEM_UNDO – jeżeli nie użyto znacznika IPC_NOWAIT, umożliwia cofnięcie operacji blokującej która kończy sie parokrotnie niepowodzeniem).

III. Programowa realizacja semafora

Listing 1 prezentuje implementacje operacji semaforowych na semaforze ogólnym, czyli operacji podnoszenia semafora (zwiększania wartości zmiennej semaforowej o 1) i operacji opuszczania semafora (zmniejszania wartości zmiennej semaforowej o 1).

```
   #include <sys/types.h>
   #include <sys/ipc.h>
3  #include <sys/sem.h>

   static struct sembuf buf;
6
   void podnies(int semid, int semnum){
      buf.sem_num = semnum;
9     buf.sem_op = 1;
      buf.sem_flg = 0;
      if (semop(semid, &buf, 1) == -1){
12          perror("Podnoszenie semafora");
            exit(1);
      }
15 }

   void opusc(int semid, int semnum){
18    buf.sem_num = semnum;
      buf.sem_op = -1;
      buf.sem_flg = 0;
21    if (semop(semid, &buf, 1) == -1){
            perror("Opuszczenie semafora");
            exit(1);
24    }
   }
```

Listing 1:Realizacja semafora ogólnego

Opis programu: W celu wykonania operacji semaforowej konieczne jest przygotowanie zmiennej o odpowiedniej strukturze. Ponieważ opisywane operacje wykonywane są tylko na jednym elemencie tablicy semaforów, zmienna ta jest typu pojedynczej struktury (linia 5), składającej się z trzech pól. Poprzedzenie deklaracji zmiennej słowem **static** oznacza, że zmienna ta będzie widoczna tylko wewnątrz pliku, w którym znajduje się jej deklaracja. Poszczególne pola zmiennej *buf* są wypełniane wartościami stosownymi do przekazanych parametrów i rodzaju operacji na semaforze (linie 8–10 i 18–20). W przypadku operacji podnoszenia semafora w pole *sem_op* podstawiane jest wartość +1 (linia 9), w przypadku operacji opuszczania -1 (linia 19). O taką liczbę ma zmienić się wartość zmiennej semaforowej.

[ilość modułów: 1]

IV. Problem producenta i konsumenta z wykorzystaniem semaforów i pamięci współdzielonej

Listingi 2 i 3 przedstawiają rozwiązanie problemu producenta i konsumenta (odpowiednio program producenta i program konsumenta) przy założeniu, że istnieje jeden proces producenta i jeden proces konsumenta, które przekazują sobie dane (wyprodukowane elementy) przez bufor w pamięci współdzielonej i synchronizują się przez semafory. W prezentowanym rozwiązaniu zakłada się, że producent uruchamiany jest przed konsumentem, w związku z czym w programie producenta (listing 2) tworzone i inicjalizowane są semafory oraz segment pamięci współdzielonej.

```
   #include <sys/types.h>
   #include <sys/ipc.h>
3  #include <sys/shm.h>
   #include <sys/sem.h>

6  #define MAX 10
   main(){
9      int shmid, semid, i;
       int *buf;
12     semid = semget(45281, 2, IPC_CREAT|0600);
       if (semid == -1){
              perror("Utworzenie tablicy semaforow");
15            exit(1);
       }
       if (semctl(semid, 0, SETVAL, (int)MAX) == -1){
18            perror("Nadanie wartosci semaforowi 0");
              exit(1);
       }
21     if (semctl(semid, 1, SETVAL, (int)0) == -1){
              perror("Nadanie wartosci semaforowi 1");
              exit(1);
24     }
       shmid = shmget(45281, MAX*sizeof(int), IPC_CREAT|0600);
27     if (shmid == -1){
              perror("Utworzenie segmentu pamieci wspoldzielonej");
              exit(1);
30     }
       buf = (int*)shmat(shmid, NULL, 0);
33     if (buf == NULL){
              perror("Przylaczenie segmentu pamieci wspoldzielonej");
              exit(1);
36     }
       for (i=0; i<10000; i++){
39            opusc(semid, 0);
              buf[i%MAX] = i;
              podnies(semid, 1);
42     }
   }
```

Listing 2: Synchronizacja producenta w dostępie do bufora cyklicznego

Opis programu: W linii 12 tworzona jest tablica semaforów o rozmiarze 2 (obejmująca 2 semafory). W liniach 17 i 21 semaforom tym nadawane są wartości początkowe. Semafor nr 0 otrzymuje wartość początkową MAX (linia 17), semafor nr 1 otrzymuje 0 (linia 21). W liniach 26–36 tworzony jest obszar pamięci współdzielonej. Zapis bufora cyklicznego (linia 40) poprzedzony jest wykonaniem operacji opuszczenia semafora nr 0 (linia 39). Semafor ten ma początkową wartość MAX, zatem można wykonać MAX operacji zapisu, czyli zapełnić całkowicie bufor, którego rozmiar jest równy MAX. Semafor osiągnie tym samym wartość 0 i przy kolejnym obrocie pętli

nastąpi zablokowanie procesu w operacji opuszczania tego semafora, aż do momentu podniesienia go przez inny proces. Będzie to proces konsumenta (odczytujący), a operacja wykonana będzie po odczytaniu (linia 33 na listingu 3). Wartość semafora nr 0 określa więc liczbę wolnych pozycji w buforze. Po każdym wykonaniu operacji zapisu podnoszony jest semafor nr 1. Jego wartość odzwierciedla zatem poziom zapełnienie bufora i początkowo jest równa 0 (bufor jest pusty). Semafor nr 1 blokuje konsumenta przed dostępem do pustego bufora (linia 31 na listingu 3).

```
   #include <sys/types.h>
   #include <sys/ipc.h>
3  #include <sys/shm.h>
   #include <sys/sem.h>

6  #define MAX 10

   main(){
9     int shmid, semid, i;
      int *buf;

12    semid = semget(45281, 2, 0600);
      if (semid == -1){
            perror("Uzyskanie identyfikatora tablicy semaforow");
15          exit(1);
      }
18    shmid = shmget(45281, MAX*sizeof(int), 0600);
      if (shmid == -1){
            perror("Uzyskanie identyfikatora segmentu pamieci
            wspoldzielonej");
21          exit(1);
      }
24    buf = (int*)shmat(shmid, NULL, 0);
      if (buf == NULL){
            perror("Przylaczenie segmentu pamieci wspoldzielonej");
27          exit(1);
      }
30    for (i=0; i<10000; i++){
            opusc(semid, 1);
            printf("Numer: %5d Wartosc: %5d\n", i, buf[i%MAX]);
33          podnies(semid, 0);
      }
   }
```

Listing 3: Synchronizacja konsumenta w dostępie do bufora cyklicznego

Opis programu: W programie konsumenta nie jest tworzony segment pamięci współdzielonej ani tablica semaforów. Są tylko pobierane ich identyfikatory (linia 12 i 18). Konsument może pobrać jakiś element, jeśli bufor nie jest pusty. Przed dostępem do pustego bufora chroni semafor nr 1. Konsument nie może go opuścić, jeśli ma on wartość 0. Semafor ten zwiększany jest po umieszczeniu kolejnego elementu w buforze przez producenta (linia 41 na listingu 1).

Listingi 4 i 5 prezentuje rozwiązanie problemu producentów i konsumentów, czyli dopuszczają istnienie jednocześnie wielu producentów i wielu konsumentów. W tym programie założono, że proces, który pierwszy utworzy tablicę semaforów, dokona inicjalizacji odpowiednich struktur.

```
   #include <sys/types.h>
   #include <sys/ipc.h>
3  #include <sys/shm.h>
   #include <sys/sem.h>

6  #define MAX 10

   main(){
9     int shmid, semid, i;
      int *buf;

12    shmid = shmget(45281, (MAX+2)*sizeof(int), IPC_CREAT|0600);
      if (shmid == -1){
            perror("Utworzenie segmentu pamieci wspoldzielonej");
15          exit(1);
      }

18    buf = (int*)shmat(shmid, NULL, 0);
      if (buf == NULL){
            perror("Przylaczenie segmentu pamieci wspoldzielonej");
21          exit(1);
      }

24    #define indexZ buf[MAX]
      #define indexO buf[MAX+1]

27    semid = semget(45281, 4, IPC_CREAT|IPC_EXCL|0600);
      if (semid == -1){
            semid = semget(45281, 4, 0600);
30          if (semid == -1){
                  perror("Utworzenie tablicy semaforow");
                  exit(1);
33          }
      }
      else{
36          indexZ = 0;
            indexO = 0;
            if (semctl(semid, 0, SETVAL, (int)MAX) == -1){
39                perror("Nadanie wartosci semaforowi 0");
                  exit(1);
            }
42          if (semctl(semid, 1, SETVAL, (int)0) == -1){
                  perror("Nadanie wartosci semaforowi 1");
                  exit(1);
45          }
            if (semctl(semid, 2, SETVAL, (int)1) == -1){
                  perror("Nadanie wartosci semaforowi 2");
48                exit(1);
            }
            if (semctl(semid, 3, SETVAL, (int)1) == -1){
51                perror("Nadanie wartosci semaforowi 3");
                  exit(1);
            }
54    }
```

[ilość modułów: 1]

```
      for (i=0; i<10000; i++){
57          opusc(semid, 0);
            opusc(semid, 2);
            buf[indexZ] = i;
60          indexZ = (indexZ+1)%MAX;
            podnies(semid, 2);
            podnies(semid, 1);
63    }
   }
```

Listing 4: Synchronizacja wielu producentów w dostępie do bufora cyklicznego

Opis programu: W linii 27 następuje próba utworzenia tablicy semaforów. Flaga IPC_EXCL powoduje zwrócenie przez funkcję **semget** wartości -1, gdy tablica o podanym kluczu już istnieje. Zwrócenie przez funkcję wartości -1 oznacza, że tablica już istnieje i pobierany jest tylko jej identyfikator (linia 29). W przeciwnym przypadku (tzn. wówczas, gdy tablica rzeczywiście jest tworzona) proces staje się inicjatorem struktur danych na potrzeby komunikacji, wykonując fragment programu w liniach 36–53.
Do prawidłowej synchronizacji producentów i konsumentów potrzebne są cztery semafory. Dwa z nich (semafory numer 0 i 1 w tablicy) służą do kontroli liczby wolnych i zajętych pozycji w buforze, podobnie jak w przypadku jednego producenta i jednego konsumenta. W przypadku wielu producentów i wielu konsumentów zarówno producenci jak i konsumenci muszą współdzielić indeks pozycji odpowiednio do zapisu i do odczytu. Indeksy te przechowywane są we współdzielonym segmencie pamięci zaraz za buforem. Stąd rozmiar tworzonego segmentu ma wynosi MAX+2 pozycji typu int (linia 12). Indeks kolejnej pozycji do zapisu przechowywany jest pod indeksem MAX we współdzielonej tablicy (linia 24), a indeks kolejnej pozycji do odczytu przechowywany jest pod indeksem MAX+1 w tej tablicy (linia 25). Pozycje 0–MAX-1 stanowią bufor do komunikacji. Pozostałe dwa (semafory numer 2 i 3) służ, a zatem do zapewnienia wzajemnego wykluczania w dostępie do współdzielonych indeksów pozycji do zapisu i do odczytu. Poszczególne semafory są inicjalizowane odpowiednimi wartościami w liniach 38, 42, 46 i 50. W liniach 36 i 37 inicjalizowane są wartościami zerowymi współdzielone indeksy pozycji do zapisu i od odczytu.
Działanie producentów polega na uzyskaniu wolnej pozycji w buforze dzięki opuszczeniu semafora nr 0, podobnie jak w poprzednim przykładzie, oraz uzyskaniu wyłączności dostępu do indeksu pozycji do zapisu. Wyłączność dostępu do indeksu jest konieczna, gdyż w przeciwnym razie dwaj producenci mogliby jednocześnie modyfikować ten indeks, w efekcie czego uzyskaliby tę samą wartość i próbowaliby umieścić ”wyprodukowane” elementy na tej samej pozycji w buforze. Wyłączność dostępu do indeksu uzyskiwana jest przez opuszczenie semafora nr 2 (linia 58), którego wartość początkowa jest 1. Po opuszczeniu przyjmuje on wartość 0, co uniemożliwia opuszczenie go przez inny proces do momentu podniesienia w linii 61. Przed dopuszczeniem innego procesu do możliwości aktualizacji indeksu następuje umieszczenie ”wyprodukowanego” elementu w buforze (linia 59) oraz aktualizacja indeksu do zapisu tak, żeby wskazywał on na następną pozycję w buforze (linia 60). W linii 62 następuje podniesienie semafora nr 1, wskazującego na liczbę elementów do ”skonsumowania” w buforze.

```
   #include <sys/types.h>
   #include <sys/ipc.h>
3  #include <sys/shm.h>
   #include <sys/sem.h>
6  #define MAX 10

   main(){
9      int shmid, semid, i;
       int *buf;
12     shmid = shmget(45281, (MAX+2)*sizeof(int), IPC_CREAT|0600);
       if (shmid == -1){
             perror("Utworzenie segmentu pamieci wspoldzielonej");
15           exit(1);
       }
18     buf = (int*)shmat(shmid, NULL, 0);
       if (buf == NULL){
             perror("Przylaczenie segmentu pamieci wspoldzielonej");
21           exit(1);
       }

24     #define indexZ buf[MAX]
       #define indexO buf[MAX+1]

27     semid = semget(45281, 4, IPC_CREAT|IPC_EXCL|0600);
       if (semid == -1){
             semid = semget(45281, 4, 0600);
30           if (semid == -1){
                   perror("Utworzenie tablicy semaforow");
                   exit(1);
33           }
       }
       else{
36           indexZ = 0;
             indexO = 0;
             if (semctl(semid, 0, SETVAL, (int)MAX) == -1){
39                 perror("Nadanie wartosci semaforowi 0");
                   exit(1);
             }
42           if (semctl(semid, 1, SETVAL, (int)0) == -1){
                   perror("Nadanie wartosci semaforowi 1");
                   exit(1);
45           }
             if (semctl(semid, 2, SETVAL, (int)1) == -1){
                   perror("Nadanie wartosci semaforowi 2");
48                 exit(1);
             }
             if (semctl(semid, 3, SETVAL, (int)1) == -1){
51                 perror("Nadanie wartosci semaforowi 3");
                   exit(1);
             }
54     }
```

[ilość modułów: 1]

```
      for (i=0; i<10000; i++){
57          opusc(semid, 1);
            opusc(semid, 3);
            printf("Numer: %5d Wartosc: %5d\n", i, buf[indexO]);
60          indexO = (indexO+1)%MAX;
            podnies(semid, 3);
            podnies(semid, 0);
63    }
   }
```

Listing 5: Synchronizacja wielu konsumentów w dostępie do bufora cyklicznego

Opis programu: Program konsumenta w liniach 1–56 jest identyczny, jak program producenta. W pozostałych liniach jest on ”symetryczny”w tym sensie, że opuszczany jest semafor nr 1, kontrolujący liczbę zajętych pozycji (elementów do ”skonsumowania”,linia 57), a po pobraniu elementu podnoszony jest semafor nr 0, kontrolujący liczbę wolnych pozycji (linia 62). Wzajemne wykluczanie w dostępie do współdzielonego indeksu do zapisu zapewnia semafor nr 3, który jest opuszczany w linii 58, a podnoszony w linii 61.

V. Zadania do samodzielnego wykonania.

Napisz programy realizujące poniższe zadania. Do implementacji programów można wykorzystać uogólnione operacje semaforowe *P* i *V* operujące na semaforach wielowartościowych (atomowe zmniejszanie i zwiększanie semafora o dowolną wartość).

1) Przy stole siedzi pięciu filozofów, którzy na przemian jedzą spaghetti ze wspólnej miski oraz myślą. Żeby się najeść filozof potrzebuje dwa widelce, przy czym każdy z widelców jest współdzielony z sąsiadem. Każdy z filozofów wykonuje zatem cyklicznie następujące czynności: myśli, bierze widelce, je, odkłada widelce i znowu myśli. Widelec jest używany w trybie wyłącznym, czyli tylko jeden z dwóch siedzących obok siebie filozofów, może z niego korzystać. Napisać program dla procesu filozofa, zapewniający, że nie nastąpi zakleszczenie ani zagłodzenie któregoś z procesów. Widelce należy traktować jako zasoby.
2) Do czytelni przychodzą dwojakiego rodzaju użytkownicy: czytelnicy i pisarze. Czytelnicy mogą korzystać z czytelni w trybie współdzielonym,(kilku czytelników może korzystać z niej równocześnie), a pisarze korzystają z niej w trybie wyłącznym(pisarz nie może współdzielić czytelni z czytelnikiem lub innym pisarzem) Napisać programy dla procesu użytkownika i procesu-pisarza, umożliwiające każdemu użytkownikowi skorzystanie ostatecznie z czytelni, przy założeniu, że czytelnia ma ograniczony rozmiar. (może jednocześnie pomieścić nie więcej niż m czytelników)
3) Fabryka posiada 2 stanowiska produkcyjne A i B. Na każdym stanowisku składane są wyroby z podzespołów X, Y i Z. Podzespoły przechowywane są w magazynie o pojemności M jednostek. Podzespół X zajmuje jedną jednostkę magazynową, podzespół Y dwie, a podzespół Z trzy jednostki. Podzespoły pobierane są z magazynu, przenoszone na stanowisko produkcyjne i montowane. Z podzespołów X, Y i Z po ich połączeniu powstaje jeden produkt, po czym pobierane są następne podzespoły z magazynu. Jednocześnie trwają dostawy podzespołów do fabryki. Podzespoły pochodzą z 3 niezależnych źródeł i dostarczane są w nieokreślonych momentach czasowych. Fabryka przyjmuje do magazynu maksymalnie dużo podzespołów dla zachowania płynności produkcji.
   Napisz program dla procesów dostawca i monter reprezentujących odpowiednio: dostawców produktów X, Y i Z oraz pracowników na stanowiskach A i B z
4) Przy taśmie transportowej pracuje trzech pracowników oznaczonych przez P_1, P_2 i P_3. Pracownicy wrzucają na taśmę cegły o masach odpowiednio 1, 2 i 3 jednostki. Na końcu taśmy

stoi ciężarówka o ładowności C jednostek, którą należy zawsze załadować do pełna. Wszyscy pracownicy starają się układać cegły na taśmie najszybciej jak to możliwe. Taśma może przetransportować w danej chwili maksymalnie K sztuk cegieł. Jednocześnie jednak taśma ma ograniczony udźwig: maksymalnie M jednostek masy, tak, że niedopuszczalne jest położenie np. samych tylko cegieł najcięższych (3K>M). Po zapełnieniu ciężarówki na jej miejsce pojawia się natychmiast nowa o takich samych parametrach. Cegły „zjeżdżające" z taśmy muszą od razu trafić na samochód dokładnie w takiej kolejności jak zostały położone na taśmie.
Napisz programy symulujące działanie pracowników i ciężarówki.

5) Przy nabrzeżu stoi statek o pojemności N. Statek z lądem jest połączony mostkiem o pojemności K (K<N). Na statek próbują dostać się pasażerowie, z tym, że na statek nie może ich wejść więcej niż N, a wchodząc na statek na mostku nie może być ich równocześnie więcej niż K. Statek co jedną godzinę wypływa w rejs. W momencie odpływania kapitan statku musi dopilnować aby na mostku nie było żadnego wchodzącego pasażera. Jednocześnie musi dopilnować by liczba pasażerów na statku nie przekroczyła N. Napisać odpowiednio procedury Pasażer i Kapitan.

VI. Literatura.

[HGS99] Havilland K., Gray D., Salama B., *Unix - programowanie systemowe*, ReadMe, 1999

[Roch97] Rochkind M.J., *Programowanie w systemie UNIX dla zaawansowanych*, WNT, 1997

[NS99] Neil M., Stones R,*Linux. Programowanie*, ReadMe, 1999

[MOS02] Mitchell M., Oldham J., Samuel A.,*Linux. Programowanie dla zaawansowanych*, ReadMe,2002

[St02] Stevens R.W., *Programowanie w środowisku systemu UNIX*, WNT, 2002