5 Mechanizmy IPC

Mechanizmy IPC (ang. Interprocess Communication) obejmują pamięć współdzieloną, semafory i kolejki komunikatów. Semafory są raczej mechanizmem synchronizacji, niż komunikacji procesów. Ponieważ dostęp współbieżnych procesów do pamięci współdzielonej wymaga najczęściej odpowiedniej synchronizacji, w sekcji 5.3 zaprezentowane jest łączne użycie semaforów razem z pamięcią współdzieloną do rozwiązania problemu producenta i konsumenta z ograniczonym buforem cyklicznym.

Wszystkie funkcje dotyczące mechanizmów IPC mają przedrostek zależny od rodzaju mechanizmu. Dla pamięci współdzielonej jest to *shm*, dla semaforów jest to *sem*, a dla kolejek komunikatów — *msg*. Niezależnie od rodzaju mechniazmu można wyodrębnić pewne ich cechy wspólne. Obiekt reprezentujący mechanizam z grupy IPC tworzony jest funkcją typu *get*, czyli funkcją o nazwie „get" poprzedzonej przedrostkiem właściwym dla danego typu mechanizmu. Podobnie, pewne operacje sterujące i kontrolne można wykonać funkcją typu *ctl*. Właściwe operacje umożliwiające komunikację lub synchronizację pomiędzy procesami zależne są od rodzaju mechanizmu. Funkcje te zebrane zostały w tab. 5.1.

	tworzenie *get*	operacje *op*	kontrola *ctl*
pamięć współdzielona	`shmget`	`shmat` `shmdt`	`shmctl`
semafory	`semget`	`semop`	`semctl`
kolejki komunikatów	`msgget`	`msgsnd` `msgrcv`	`msgctl`

Tablica 5.1: Funkcje systemowe mechanizmów IPC

Pierwszym argumentem funkcji *get* jest *klucz*, który jest wartością całkowitą typu `key_t`. Klucz jest odnośnikiem do konkretnego obiektu w ramach danego rodzaju mechanizmów i jest odwzorowywany na identyfikator, który jest wartością zwrotną funkcji *get* (jest to wartość typu `int`). Za pomocą klucza różne procesy mogą odnosić się do tego samego obiektu (uzyskać jego identyfikator), co jest warunkiem wykorzystania obiektu do komunikacji lub synchronizacji pomiędzy tymi procesami. Każdy rodzaj mechanizmu IPC ma osobną przestrzeń wartości kluczy. Użycie zatem tej samej wartości klucza dla semafora i dla segmentu pamięci współdzielonej nie powoduje konfliktów. Jeśli nie ma potrzeby przekazywania innym procesom klucza, gdyż identyfikator danego obiektu uzyskiwany jest w inny sposób (np. dziedziczony jest przez potomka od procesu macierzystego), zamiast konkretnej wartości klucza do funkcji *get* można przekazać stałą `IPC_PRIVATE`. Ostatni parametr funkcji *get* — *flg* — określa prawa dostępu do nowo tworzonego obiektu reprezentującego mechanizm IPC. W celu utworzenia obiektu do praw dostępu należy dołożyć poprzez sumę bitową flagę `IPC_CREAT` (tzn. `IPC_CREAT|0640`). Jeśli obiekt o podanym konkretnym kluczu (innym niż `IPC_PRIVATE`) już istnieje, zwrócony zostanie jego identyfikator i nowy obiekt nie jest tworzony. Jeśli dodatkowo ustawiona zostanie flaga `IPC_EXCL` (tzn. `IPC_CREAT|IPC_EXCL|0640`), to w przypadku istnienia obiektu o takim kluczu zwrócona zostanie wartość –1.

Identyfikator zwrócony przez funkcję *get* jest pierwszym argumentem pozostałych funkcji do obsługi mechanizmów IPC. Pozostałe parametry tych funkcji zależne są od konkretnego mechanizmu. W podobny, niezależny od rodzaju mechanizmu, sposób można jednak usunąć istniejący obiekt IPC oraz pobrać lub ustawić atrybuty takiego obiektu. W tym celu wywołuję się odpowiednią funkcję typu *ctl*, przekazując odpowiednio stałą `IPC_RMID`, `IPC_STAT` lub `IPC_SET` jako wartość parametru *cmd*.

5.1 Pamięć współdzielona

Segment pamięci współdzielonej tworzony jest przez wywołanie funkcji systemowej `shmget`. Specyficznym parametrem tej funkcji jest rozmiar segmentu współdzielonej pamięci.

`int shmget(key_t` *key*`, int` *size*`, int` *shmflg*`)` — utworzenie segmentu pamięci współdzielonej. Funkcja zwraca identyfikator segmentu pamięci współdzielonej w przypadku poprawnego zakończenia lub –1 w przypadku błędu.

Opis parametrów:

key klucz,
size rozmiar obszaru współdzielonej pamięci w bajtach,
shmflg flagi (prawa dostępu, `IPC_CREAT`, `IPC_EXCL`).

W celu udostępnienia segmentu pamięci współdzielonej w procesie, należy go włączyć w przestrzeń adresową danego procesu za pomocą funkcji `shmat`. Segment taki można odłączyć za pomocą funkcji `shmdt`. Po włączeniu segmentu proces uzyskuje adres początku współdzielonego obszaru pamięci w swojej przestrzeni adresowej.

`void *shmat(int` *shmid*`, const void *`*shmaddr*`, int` *shmflg*`)` — włączenie segmentu współdzielonej pamięci w przestrzeń adresową procesu. Funkcja zwraca adres segmentu w przypadku poprawnego zakończenia lub –1 w przypadku błędu.

Opis parametrów:

shmid identyfikator obszaru współdzielonej pamięci, zwrócony przez funkcję `shmget`,
shmaddr adres w przestrzeni adresowej procesu, pod którym ma być dostępny segment współdzielonej pamięci (wartość `NULL` oznacza wybór adresu przez system),
shmflg flagi, specyfikujące sposób przyłączenia (np. `SHM_RDONLY` — przyłączenie tylko do odczytu).

`int shmdt(const void *`*shmaddr*`)` — wyłączenie segmentu z przestrzeni adresowej procesu. Funkcja zwraca 0 w przypadku poprawnego zakończenia lub –1 w przypadku błędu.

Opis parametrów:

shmaddr adres początku obszaru współdzielonej pamięci w przestrzeni adresowej procesu.

Funkcja systemowa `shmctl` umożliwia przeprowadzanie operacji kontrolnych, np. pobranie atrybutów segmentu współdzielonej pamięci lub jego usunięcie.

`int shmctl(int` *shmid*`, int` *cmd*`, struct shmid_ds *`*buf*`)` — wykonanie operacji kontrolnych na segmencie pamięci współdzielonej. Funkcja zwraca 0 w przypadku poprawnego zakończenia lub –1 w przypadku błędu.

Opis parametrów:

shmid identyfikator obszaru współdzielonej pamięci, zwrócony przez funkcję `shmget`,
cmd specyfikacja wykonywanej operacji kontrolnej (np. `IPC_STAT`, `IPC_SET`, `IPC_RMID`),
shmid_ds wskaźnik na strukturę opisującą atrybuty segmentu współdzielonej pamięci (np. właściciel, prawa dostępu).

Listing 1 przedstawia program, w którym następuje cykliczny zapis bufora umieszczonego we współdzielonym obszarze pamięci. Listing 2 przedstawia program, w którym jest analogiczny odczyt bufora cyklicznego.

Listing 1: Zapis bufora cyklicznego

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

#define MAX 10

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

   shmid = shmget(45281, MAX*sizeof(int), IPC_CREAT|0600);
```

```
12    if (shmid == -1){
         perror("Utworzenie segmentu pamieci wspoldzielonej");
         exit(1);
15    }

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

      for (i=0; i<10000; i++)
24       buf[i%MAX] = i;
  }
```

Opis programu: W linii 11 tworzony jest segment współdzielonej pamięci o kluczu `45281`, o rozmiarze `MAX*sizeof(int)` i prawach do zapisu i odczytu przez właściciela. Jeśli obszar o takim kluczu już istnieje, zwracany jest jego identyfikator, czyli nie jest tworzony nowy obszar i tym samym rozmiar podany w drugim parametrze oraz prawa dostępu są ignorowane. W linii 17 utworzony segment włączony zostaje do segmentów danego procesu i zwracany jest adres tego segmentu. Zwrócony adres podstawiany jest pod zmienną `buf`. `buf` jest zatem adresem tablicy o rozmiarze `MAX` i typie składowym `int`. Pętla w liniach 23–24 oznacza cykliczny zapis tego bufora, tzn. indeks pozycji, na której zapisujemy jest równy `i%MAX`, czyli zmienia się cyklicznie od 0 do `MAX-1`.

Listing 2: Odczyt bufora cyklicznego

```
1 #include <sys/types.h>
  #include <sys/ipc.h>
  #include <sys/shm.h>
4
  #define MAX 10

7 main(){
     int shmid, i;
     int *buf;
10
     shmid = shmget(45281, MAX*sizeof(int), IPC_CREAT|0600);
     if (shmid == -1){
13       perror("Utworzenie segmentu pamieci wspoldzielonej");
         exit(1);
     }
16
     buf = (int*)shmat(shmid, NULL, 0);
     if (buf == NULL){
19       perror("Przylaczenie segmentu pamieci wspoldzielonej");
         exit(1);
     }
22
     for (i=0; i<10000; i++)
        printf("Numer: %5d   Wartosc: %5d\n", i, buf[i%MAX]);
25 }
```

Opis programu: Powyższy program jest analogiczny, jak program na listingu 1, przy czym w pętli w liniach 23–24 następuje cykliczny odczyt, czyli odczyt z pozycji w buforze, zmieniającej się cyklicznie od 0 do `MAX-1`.

5.2 Semafory

Funkcja `semget` umożliwia utworzenie zbioru semaforów, przy czym operacje semaforowe można wykonywać niezależnie na każdym semaforze z utworzonego zbioru lub w sposób atomowy na kilku (w szczególności na wszystkich) semaforach w zbiorze. Specyficznym parametrem funkcji `semget` jest liczba semaforów w tworzonym zbiorze.

`int semget(key_t` *key*`, int` *nsems*`, int` *semflg*`)` — utworzenie zbioru (tablicy) semaforów. Funkcja zwraca identyfikator zbioru semaforów w przypadku poprawnego zakończenia lub –1 w przypadku błędu.

Opis parametrów:

key klucz,
nsems liczba semaforów w tworzonym zbiorze,
semflg flagi (prawa dostępu, `IPC_CREAT`, `IPC_EXCL`).

Wykonanie operacji semaforowej polega na wywołaniu funkcji systemowej `semop`. Same operacje wyspecyfikowane są w tablicy odpowiednich struktur. Każdy element tej tablicy definiuje operację na jednym semaforze z danego zbioru. Operacja rozumiana jest jako próba zmodyfikowania zmiennej semaforowej przez dodanie do niej podanej w odpowiednim polu struktury wartości dodatniej lub ujemnej. Wartość dodatnia oznacza podnoszenie semafora, a wartość ujemna oznacza jego opuszczanie. Próba opuszczenia semafora może oznaczać zablokowanie procesu, jeśli wartość zmiennej semaforowej po operacji miałaby być ujemna. Specjalne znaczenie ma podanie 0, jako wartości modyfikującej. Oznacza ona zablokowanie procesu w oczekiwaniu na osiągnięcie przez zmienną semaforową wartości 0.

`int semop(int` *semid*`, struct sembuf *`*sops*`, unsigned` *nsops*`)` — wykonanie operacji semaforowej. Funkcja zwraca 0 w przypadku poprawnego zakończenia lub –1 w przypadku błędu.

Opis parametrów:

semid identyfikator zbioru semaforów, zwrócony przez funkcję `semget`,
sops adres tablicy struktur, w której każdy element opisuje operację na jednym semaforze w zbiorze, według następującej definicji:

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

Znaczenie poszczególnych pól struktury `sembuf` jest następujące:

sem_num numer semafora, na którym ma być wykonana operacja, w zbiorze
sem_op wartość, która ma zostać dodana do zmiennej semaforowej,
sem_flg flagi operacji (`IPC_NOWAIT` — wykonanie bez blokowania, `SEM_UNDO` — wycofanie operacji w przypadku zakończenia procesu).

nsops rozmiar tablicy adresowanej przez *sops* (liczba elementów o strukturze `sembuf`).

Funkcja `semctl` umożliwia wykonanie podobnych operacji na obiekcie, reprezentującym zbiór semaforów, jak w przypadku segmentu pamięci współdzielonej. Dodatkowo, funkcja ta umożliwia ustawienie konkretnej wartości określonej zmiennej ze zbioru semaforów lub wszystkich zmiennych z tego zbioru oraz odczytanie wartości zmiennej semaforowej. Możliwość ustawienie wartości zmiennej semaforowej jest szczególnie istotna w stanie początkowym przetwarzania w celu odpowiedniej inicjalizacji mechanizmów synchronizacji.

`int semctl(int` *semid*`, int` *semnum*`, int` *cmd*`, ...)` — wykonanie operacji kontrolnych na semaforze lub zbiorze semaforów. Funkcja zwraca 0 w przypadku poprawnego zakończenia lub –1 w przypadku błędu.

Opis parametrów:

semid	identyfikator zbioru semaforów, zwrócony przez funkcję `semget`,
semnum	numer semafora, na którym ma być wykonana operacja, w identyfikowanym zbiorze,
cmd	specyfikacja wykonywanej operacji kontrolnej (np. `IPC_STAT`, `IPC_SET`, `IPC_RMID`, `SETVAL`, `GETVAL`, `SETALL` itp.).

Listing 3 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).

Listing 3: Realizacja semafora ogólnego

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

static struct sembuf buf;

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

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

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.

5.3 Problem producenta i konsumenta z wykorzystaniem semaforów i pamięci współdzielonej

Listingi 4 i 5 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 4) tworzone i inicjalizowane są semafory oraz segment pamięci współdzielonej.

Listing 4: Synchronizacja producenta w dostępie do bufora cyklicznego

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

#define MAX 10

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

    semid = semget(45281, 2, IPC_CREAT|0600);
    if (semid == -1){
       perror("Utworzenie tablicy semaforow");
       exit(1);
    }
    if (semctl(semid, 0, SETVAL, (int)MAX) == -1){
       perror("Nadanie wartosci semaforowi 0");
       exit(1);
    }
    if (semctl(semid, 1, SETVAL, (int)0) == -1){
       perror("Nadanie wartosci semaforowi 1");
       exit(1);
    }

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

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

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

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, podobnie jak w przykładach w sekcji 5.1. 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 5). 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 5).

Listing 5: Synchronizacja konsumenta w dostępie do bufora cyklicznego

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

  #define MAX 10
7
  main(){
     int shmid, semid, i;
10   int *buf;

     semid = semget(45281, 2, 0600);
13   if (semid == -1){
        perror("Uzyskanie identyfikatora tablicy semaforow");
        exit(1);
16   }

     shmid = shmget(45281, MAX*sizeof(int), 0600);
19   if (shmid == -1){
        perror("Uzyskanie identyfikatora segmentu pamieci
           wspoldzielonej");
        exit(1);
22   }

     buf = (int*)shmat(shmid, NULL, 0);
25   if (buf == NULL){
        perror("Przylaczenie segmentu pamieci wspoldzielonej");
        exit(1);
28   }

     for (i=0; i<10000; i++){
31      opusc(semid, 1);
        printf("Numer: %5d   Wartosc: %5d\n", i, buf[i%MAX]);
        podnies(semid, 0);
34   }
  }
```

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 4).

Listingi 6 i 7 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.

Listing 6: Synchronizacja wielu producentów w dostępie do bufora cyklicznego

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

#define MAX 10

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

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

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

   #define indexZ buf[MAX]
   #define index0 buf[MAX+1]

   semid = semget(45281, 4, IPC_CREAT|IPC_EXCL|0600);
   if (semid == -1){
      semid = semget(45281, 4, 0600);
      if (semid == -1){
         perror("Utworzenie tablicy semaforow");
         exit(1);
      }
   }
   else{
      indexZ = 0;
      index0 = 0;
      if (semctl(semid, 0, SETVAL, (int)MAX) == -1){
         perror("Nadanie wartosci semaforowi 0");
         exit(1);
      }
      if (semctl(semid, 1, SETVAL, (int)0) == -1){
         perror("Nadanie wartosci semaforowi 1");
         exit(1);
      }
      if (semctl(semid, 2, SETVAL, (int)1) == -1){
         perror("Nadanie wartosci semaforowi 2");
         exit(1);
      }
      if (semctl(semid, 3, SETVAL, (int)1) == -1){
         perror("Nadanie wartosci semaforowi 3");
         exit(1);
      }
   }

   for (i=0; i<10000; i++){
      opusc(semid, 0);
      opusc(semid, 2);
      buf[indexZ] = i;
```

```
60          indexZ = (indexZ+1)%MAX;
            podnies(semid, 2);
            podnies(semid, 1);
63      }
    }
```

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) służą 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.

Listing 7: Synchronizacja wielu konsumentów w dostępie do bufora cyklicznego

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

   #define MAX 10
7
   main(){
      int shmid, semid, i;
10    int *buf;

      shmid = shmget(45281, (MAX+2)*sizeof(int), IPC_CREAT|0600);
13    if (shmid == -1){
```

```
        perror("Utworzenie segmentu pamieci wspoldzielonej");
        exit(1);
16    }

      buf = (int*)shmat(shmid, NULL, 0);
19    if (buf == NULL){
        perror("Przylaczenie segmentu pamieci wspoldzielonej");
        exit(1);
22    }

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

      semid = semget(45281, 4, IPC_CREAT|IPC_EXCL|0600);
28    if (semid == -1){
        semid = semget(45281, 4, 0600);
        if (semid == -1){
31        perror("Utworzenie tablicy semaforow");
          exit(1);
        }
34    }
      else{
        indexZ = 0;
37      indexO = 0;
        if (semctl(semid, 0, SETVAL, (int)MAX) == -1){
          perror("Nadanie wartosci semaforowi 0");
40        exit(1);
        }
        if (semctl(semid, 1, SETVAL, (int)0) == -1){
43        perror("Nadanie wartosci semaforowi 1");
          exit(1);
        }
46      if (semctl(semid, 2, SETVAL, (int)1) == -1){
          perror("Nadanie wartosci semaforowi 2");
          exit(1);
49      }
        if (semctl(semid, 3, SETVAL, (int)1) == -1){
          perror("Nadanie wartosci semaforowi 3");
52        exit(1);
        }
      }
55
      for (i=0; i<10000; i++){
        opusc(semid, 1);
58      opusc(semid, 3);
        printf("Numer: %5d   Wartosc: %5d\n", i, buf[indexO]);
        indexO = (indexO+1)%MAX;
61      podnies(semid, 3);
        podnies(semid, 0);
      }
64  }
```

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.

5.4 Kolejki komunikatów

Listingi 4 i 5 przedstawiają rozwiązanie problemu producenta i konsumenta (odpowiednio program producenta i program konsumenta) na kolejce komunikatów. W rozwiązaniu zakłada się ograniczone buforowanie, tzn. nie może być więcej nie skonsumowanych elementów, niż pewna założona ilość (pojemność bufora). Rozwiązanie dopuszcza możliwość istnienia wielu producentów i wielu konsumentów.

Listing 8: Impelmentacja zapisu ograniczonego bufora za pomocą kolejki komunikatów

```
1  #include <sys/types.h>
   #include <sys/ipc.h>
   #include <sys/msg.h>
4
   #define MAX 10

7  struct buf_elem {
       long mtype;
       int mvalue;
10 };
   #define PUSTY 1
   #define PELNY 2
13
   main(){
       int msgid, i;
16     struct buf_elem elem;

       msgid = msgget(45281, IPC_CREAT|IPC_EXCL|0600);
19     if (msgid == -1){
          msgid = msgget(45281, IPC_CREAT|0600);
          if (msgid == -1){
22           perror("Utworzenie kolejki komunikatów");
             exit(1);
          }
25     }
       else{
          elem.mtype = PUSTY;
28        for (i=0; i<MAX; i++)
             if (msgsnd(msgid, &elem, sizeof(elem.mvalue), 0) == -1){
                perror("Wyslanie pustego komunikatu");
31              exit(1);
             }
       }
34
       for (i=0; i<10000; i++){
          if (msgrcv(msgid, &elem, sizeof(elem.mvalue), PUSTY, 0) == -1){
37           perror("Odebranie pustego komunikatu");
             exit(1);
          }
40        elem.mvalue = i;
          elem.mtype = PELNY;
          if (msgsnd(msgid, &elem, sizeof(elem.mvalue), 0) == -1){
43           perror("Wyslanie elementu");
             exit(1);
          }
46     }
   }
```

Opis programu: Podobnie jak w programach na lisingach 6 i 7 w linii 18 jest próba utworzenia kolejki komunikatów. Jeśli kolejka już istnieje, funkcja `msgget` zwróci wartość –1 i nastąpi

pobranie identyfikatora już istniejącej kolejki (linia 20). Jeśli kolejka nie istnieje, zostanie ona utworzona w linii 18 i nastąpi wykonanie fragmentu programu w liniach 27–32, w wyniku czego w kolejce zostanie umieszczonych `MAX` komunikatów typu `PUSTY`. Umieszczenie elementu w buforze, reprezentowanym przez kolejkę komunikatów, polega na zastąpieniu komunikatu typu `PUSTY` komunikatem typu `PELNY`. Pusty komunikat jest pobierany w linii 36. Brak pustych komunikatów oznacza całkowite zapełnienie bufora i powoduje zablokowanie procesu w funkcji `msgrcv`. Po odebraniu pustego komunikatu przygotowywany jest komunikat pełny (linie 40–41) i umieszczany jest w buforze, czyli wysyłany do kolejki (linia 42). Podobnie jak suma wartości zmiennych semaforowych do kontroli zapełnienia bufora w przykładach 4–5 i 6–7 jest równa `MAX` liczba komunikatów jest również równa `MAX`, z wyjątkiem momentu, gdy jakiś proces wykonuje operację przekazania lub pobrania elementu.

Listing 9: Impelmentacja odczytu ograniczonego bufora za pomocą kolejki komunikatów

```
1  #include <sys/types.h>
   #include <sys/ipc.h>
   #include <sys/msg.h>
4
   #define MAX 10

7  struct buf_elem {
       long mtype;
       int mvalue;
10 };
   #define PUSTY 1
   #define PELNY 2
13
   main(){
       int msgid, i;
16     struct buf_elem elem;

       msgid = msgget(45281, IPC_CREAT|IPC_EXCL|0600);
19     if (msgid == -1){
           msgid = msgget(45281, IPC_CREAT|0600);
           if (msgid == -1){
22             perror("Utworzenie kolejki komunikatów");
               exit(1);
           }
25     }
       else{
           elem.mtype = PUSTY;
28         for (i=0; i<MAX; i++)
               if (msgsnd(msgid, &elem, sizeof(elem.mvalue), 0) == -1){
                   perror("Wyslanie pustego komunikatu");
31                 exit(1);
               }
       }
34
       for (i=0; i<10000; i++){
           if (msgrcv(msgid, &elem, sizeof(elem.mvalue), PELNY, 0) == -1){
37             perror("Odebranie elementu");
               exit(1);
           }
40         printf("Numer: %5d   Wartosc: %5d\n", i, elem.mvalue);
           elem.mtype = PUSTY;
           if (msgsnd(msgid, &elem, sizeof(elem.mvalue), 0) == -1){
43             perror("Wyslanie pustego komunikatu");
               exit(1);
```

```
        }
    }
}
```

5.5 Zadania

5.1. Listingi 10 i 11 przedstawiają zmodyfikowane fragmenty (zgodnie z numeracją linii) programów odpowiednio z listingów 6 (str. 44) i 7 (str 45). Wykaż, że w zmodyfikowanej wersji może dojść do zakleszczenia procesów. Uwaga: w niniejszej wersji semafor nr 3 nie jest używany.

Listing 10: Modyfikacja programu 6

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

Listing 11: Modyfikacja programu 7

```
for (i=0; i<10000; i++){
    opusc(semid, 2);
    opusc(semid, 1);
    printf("Numer: %5d   Wartosc: %5d\n", i, buf[indexO]);
    indexO = (indexO+1)%MAX;
    podnies(semid, 0);
    podnies(semid, 2);
}
```

5.2. Listingi 12 i 13 przedstawiają zmodyfikowane fragmenty (zgodnie z numeracją linii) programów odpowiednio z listingów 6 i 7. Wskaż, na czym polega błąd w zmodyfikowanej wersji.

Listing 12: Modyfikacja programu 6

```
for (i=0; i<10000; i++){
    int index;

    opusc(semid, 0);
    opusc(semid, 2);
    index = indexZ;
    indexZ = (indexZ+1)%MAX;
    podnies(semid, 2);
    buf[index] = i;
    podnies(semid, 1);
}
```

Listing 13: Modyfikacja programu 7

```
for (i=0; i<10000; i++){
    int index;

    opusc(semid, 1);
    opusc(semid, 3);
    index = indexZ;
    indexO = (indexO+1)%MAX;
```

```
        podnies(semid, 3);
9       printf("Numer: %5d   Wartosc: %5d\n", i, buf[index]);
        podnies(semid, 0);
    }
```

5.6 Pytania kontrolne

1. Co jest wartością zwrotną funkcji typu *get*?
2. Jakie są podobieństwa i różnice w semantyce parametrów funkcji `shmget` i `semget`?
3. Do czego służy i co zwraca funkcja systemowa `shmat`?
4. Jaka byłaby prawidłowa kolejność wykonania następujących operacji w celu skorzystania z pamięci współdzielonej:
   (a) wywołanie funkcji `shmat`,
   (b) wywołanie funkcji `shmdt`,
   (c) wywołanie funkcji `shmget`
   (d) zapis lub odczyt danych w segmencie pamięci współdzielonej?
5. W jaki sposób z punktu widzenia programisty wykonuję się operację semaforową?
6. Kiedy może nastąpić zablokowanie procesu w wykonaniu operacji semaforowej?
7. W jaki sposób może nastąpić odblokowanie procesu zablokowanego w trakcie wykonywania operacji semaforowej?
8. W jaki sposób można ustawić określoną wartość zmiennej semaforowej?
9. Na czym polega synchronizacja producenta i konsumenta w dostępie do wspólnego bufora?
10. Które funkcje systemowe i w jakich warunkach mogą spowodować zablokowanie procesu?