Temat zajęć: Mechanizmy IPC: kolejki komunikatów.

Czas realizacji zajęć:	*90 min.*
Zakres materiału, jaki zostanie zrealizowany podczas zajęć:	Tworzenie kolejek komunikatów, wysyłanie i odbieranie komunikatów, parametry i usuwanie kolejek komunikatów, obsługa mechanizmów IPC w konsoli systemu, implementacja przykładowych programów obsługi kolejek komunikatów

I. Mechanizmy IPC

Podobnie jak łącza, mechanizmy IPC (Inter Process Communication) jest grupą mechanizmów komunikacji i synchronizacji procesów działających w ramach tego samego systemu operacyjnego. Mechanizmy IPC obejmują:

- kolejki komunikatów — umożliwiają przekazywanie określonych porcji danych,
- pamięć współdzieloną — umożliwiają współdzielenie kilku procesom tego samego fragmentu wirtualnej przestrzeni adresowej,
- semafory — umożliwiają synchronizacje procesów w dostępie do współdzielonych zasobów (np. do pamięci współdzielonej)

Do tworzenia obiektów IPC i manipulowania ich danymi służy zbiór funkcji przedstawiony w poniższej tabelce:

działanie	kolejka komunikatów	pamięć współdzielona	semafor
Rezerwowanie obiektu IPC, uzyskiwanie do niego dostępu	msgget	shmget	semget
Sterowanie obiektem IPC, uzyskiwanie informacji o stanie modyfikowanych obiektów IPC, usuwanie obiektów IPC	msgctl	shmctl	semctl
Operacje na obiektach IPC: wysyłanie i odbieranie komunikatów, operacje na semaforach, rezerwowanie i zwalnianie segmentów pamięci wspólnej	msgsnd, msgrcv	shmat, shmdt	semop

Wywołania systemowe *get* (**msgget**, **shmget**, **semget**) są stosowane do tworzenia nowych obiektów IPC, lub do uzyskania dostępu do obiektów istniejących. Drugim wywołaniem systemowym wspólnym dla mechanizmów IPC jest *clt* (**msgctl**, **shmctl**, **semctl**), używane w celu przeprowadzenia operacji kontrolnych na obiektach IPC.

Funkcje *get* zwracają wartości całkowitoliczbowe, nazywane identyfikatorami IPC, które identyfikują obiekty IPC. Od strony systemu operacyjnego identyfikator IPC jest indeksem w systemowej tablicy zawierającej struktury z danymi dotyczącymi uprawnień do obiektów IPC. Struktura IPC jest zdefiniowana w pliku `sys/ipc.h`.

Każda z funkcji *get* wymaga określenia argumentu typu *key_t* (będącego ekwiwalentem typu *long*), nazywanego kluczem i umożliwiającego generowanie identyfikatorów IPC. Procesy, poprzez podanie tej samej wartości klucza uzyskują dostęp do konkretnego mechanizmu IPC. Wartość klucza można określić podając samodzielnie konkretną wartość. Polu temu można także przypisać stałą **IPC_PRIVATE**, która spowoduje utworzenie obiektu IPC o niepowtarzalnej wartości identyfikatora. Jednak łącze identyfikowane przez klucz wygenerowany w oparciu o stałą IPC_PRIVATE pozwala na komunikację jedynie pomiędzy procesami spokrewnionymi (ponieważ procesy potomne dziedziczą wartość klucza od swoich przodków)

Drugim parametrem wspólnym dla wszystkich wywołań z rodziny get jest znacznik komunikatu, który określa prawa dostępu do tworzonego obiektu IPC. Prawa te mogą być połączone operacją logiczną OR (oznaczaną w języku programowania C przez |) z flagami **IPC_CREAT** lub **IPC_EXCL**. Flaga **IPC_CREAT** nakazuje funkcjom *get* utworzenie nowego obiektu IPC, jeśli on jeszcze nie istnieje. Jeśli natomiast obiekt IPC już istnieje i jego klucz nie został wygenerowany przy użyciu stałej **IPC_PRIVATE**, to funkcje *get* zwrócą identyfikator tego obiektu. Natomiast użycie flag **IPC_CREAT|IPC_EXCL** spowoduje, że gdy obiekt IPC dla danej wartości klucza już istnieje, wywołanie funkcji get zakończy się błędem. Dzięki połączeniu tych dwóch flag użytkownik posiada gwarancję, że jest on twórcą danego obiektu IPC.

Wywołania funkcji *ctl* posiadają dwa argumenty wspólne: identyfikator obiektu IPC (otrzymany w wyniku wywołania odpowiedniej funkcji *get*), oraz następujące stałe: **IPC_STAT**, **IPC_SET** i **IPC_RMID**, zdefiniowane w pliku sys/ipc.h:

- **IPC_STAT** — zwraca informację o stanie danego obiektu IPC
- **IPC_SET** — zmienia właćiciela, grupę i tryb obiektu IPC
- **IPC_RMID** — usuwa obiekt IPC z systemu

II. Obsługa mechanizmów IPC w konsoli systemu

Na poziomie systemowym dane znajdujące się w obiekcie IPC pobiera się za pomocą polecenia **`ipcs`.** Informacje na temat konkretnych obiektów: kolejek komunikatów, pamięci współdzielonej i semaforów otrzymamy stosując odpowiednio przełączniki **`-q, -m, -s`**. Czyli:

`ipcs -q msgid` – informacja nt. kolejki komunikatów o identyfikatorze msgid

`ipcs -m shmid` – informacja nt. segmentu pamięci współdzielonej o identyfikatorze shmid

`ipcs -s semid` – informacja nt. zestawu semaforów o identyfikatorze semid

Dodatkowo przełącznik -b pozwala uzyskać informację nt. maksymalnego rozmiaru obiektów IPC, czyli ilości bajtów w kolejkach, rozmiarów segmentów pamięci współdzielonej i ilości semaforów w zestawach.

Usunięcie obiektu IPC można natomiast wykonując polecenie systemowe `ipcrm`:

`ipcrm -q msgid` – usunięcie kolejki komunikatów o identyfikatorze msgid

`ipcrm -m shmid` – usunięcie segmentu pamięci współdzielonej o identyfikatorze shmid

`ipcrm -s semid` – usunięcie zestawu semaforów o identyfikatorze semid

III. Kolejki komunikatów, funkcje systemowe obsługujące kolejki komunikatów i ich argumenty.

Kolejki komunikatów umożliwiają przesyłanie pakietów danych, nazywanych komunikatami, pomiędzy różnymi procesami. Sam komunikat jest zbudowany jako struktura *msgbuf*, jego definicja znajduje się w pliku `<sys/msg.h>`:

```
struct msgbuf{
long mtype;      //typ komunikatu (>0)
char mtext[1];   //treść komunikatu
}
```

Każdy komunikat ma określony typ i długość. Typ komunikatu, pozwalający określić rodzaj komunikatu, nadaje proces inicjujący komunikat.

Komunikaty są umieszczane w kolejce w kolejności ich ich wysyłania. Nadawca może wysyłać komunikaty, nawet wówczas gdy żaden z potencjalnych odbiorców nie jest gotów do ich odbioru. Komunikaty są w takich przypadkach buforowane w kolejce oczekiwania na odebranie. Przy odbiorze komunikatu, odbiorca może oczekiwać na pierwszy przybyły komunikat lub na pierwszy komunikat określonego typu. Komunikaty w kolejce są przechowywane nawet po

zakończeniu procesu nadawcy, tak długo, aż nie zostaną odebrane lub kolejka nie zostanie zlikwidowana.

Podczas tworzenia kolejki komunikatów tworzona jest systemowa struktura danych o nazwie `msgid_ds`. Definicję tej obsługiwanej przez system struktury można znaleźć w pliku nagłówkowym `<sys/msg.h>`.
Funkcje umożliwiające komunikację za pomocą kolejek komunikatów zdefiniowane są w plikach `<sys/types.h>`, `<sys/ipc.h>` oraz `<sys/msg.h>`.

- **`int msgget(key_t key, int msgflg)`**

Wartości zwracane:
poprawne wykonanie funkcji: identyfikator kolejki komunikatów
zakończenie błędne: -1

Możliwe kody błędów (*errno*) w przypadku błędnego zakończenie funkcji:
EACCES – Kolejka skojarzona z *key*, istnieje, ale proces wywołujący funkcję nie ma wystarczających praw dostępu do tej kolejki.
EEXIST – Kolejka skojarzona z wartością *key* istnieje a *msgflg* zawiera jednocześnie oba znaczniki IPC_CREAT i IPC_EXCL.
EIDRM – Kolejka została przeznaczona do usunięcia.
ENOENT – Kolejka skojarzona z wartością *key* nie istnieje, zaś *msgflg* nie zawiera flagi IPC_CREAT.
ENOMEM – Kolejka komunikatów powinna zostać utworzona, ale w systemie brak jest pamięci na utworzenie nowej struktury danych.
ENOSPC – Kolejka komunikatów powinna zostać utworzona, ale przekroczone zostałoby systemowe ograniczenie (MSGMNI) na ilość istniejących kolejek komunikatów.

Argumenty funkcji:
key – liczba, która identyfikuje kolejkę
msgflg – sumą bitowa stałej **IPC_CREAT** i dziewięciu bitów określających prawa dostępu do kolejki.

UWAGI:
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.
Istnienie flag **IPC_CREAT** i **IPC_EXCL** w parametrze *msgflg* znaczy tyle samo w przypadku kolejki komunikatów, co istnienie flag O_CREAT i O_EXCL w argumencie mode wywołania **open** , tzn. funkcja **msgget** nie wykona się prawidłowo jeśli *msgflg* będzie zawierać flagi IPC_CREAT i IPC_EXCL , zaś kolejka komunikatów skojarzona z kluczem key już będzie istnieć.

- **`int msgsnd(int msgid, struct msgbuf *msgp, int msgs, int msgflg)`**

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

Możliwe kody błędów (*errno*) w przypadku błędnego zakończenie funkcji:
EACCES – kolejka skojarzona z wartością *key*, istnieje, ale proces wołający funkcję nie ma wystarczających praw dostępu do kolejki
EAGAIN – kolejka jest pełna, a flaga IPC_NOWAIT była ustawiona
EFAULT – niepoprawny wskaźnik *msgq*

EIDRM – kolejka została przeznaczona do usunięcia
EINTR – otrzymano sygnał podczas oczekiwania na operację zapisu
EINVAL – niepoprawny identyfikator kolejki, lub ujemny typ wiadomości, lub nieprawidłowy rozmiar wiadomości

Argumenty funkcji:
msgid – identyfikator kolejki komuniktów
msgp – wskaźnik na komunikat do wysłania
msgs – rozmiar właściwej treści komunikatu (w bajtach)
msgflg – flagi specyfikujące zachowanie się funkcji w warunkach nietypowych. Wartość ta może być ustawiona na 0 lub IPC_NOWAIT

UWAGI:
Jeśli w momencie wysyłania komunikatu system osiągnął już limit długości kolejki, to w zależności of wartości flagi *msgflg* funkcja nie wyśle komunikatu i powróci z funkcji **msgsnd** (dla *msgflg* = IPC_NOWAIT), lub zablokuje proces wywołujący aż do chwili, gdy w kolejce będzie wystarczająco dużo wolnego miejsca, by żądany komunikat mógł być wysłany (przy *msgflg*=0).
Treść wysyłanego komunikatu w rzeczywistości może mieć dowolną strukturę. Struktura ta, zanim zostanie wysłana, musi być wcześniej zarezerwowane miejsce w pamięci.
Pole *mtype* określa typ komunikatu, dzięki czemu możliwe jest przy odbiorze wybieranie z kolejki komunikatów określonego rodzaju. Typ komunikatu musi być wartością większą od 0.
Funkcja MSGRCV

- **`int msgrcv (int msgid, struct msgbuf *msgp, int msgs, long msgtyp, int msgflg)`**

Wartości zwracane:
poprawne wykonanie funkcji: ilość odebranych bajtów
zakończenie błędne: -1

Możliwe kody błędów (*errno*) w przypadku błędnego zakończenie funkcji – analogicznie do funkcji **msgsnd**

Argumenty funkcji:
msgid – identyfikator kolejki komunikatów
msgp – wskaźnik do obszaru pamięci w którym ma zostać umieszczony pobrany komunikat
msgs – rozmiar właściwej treści komunikatu
msgtyp – określa typ komunikatu który ma być odebrany z kolejki. Możliwe są nastepujące wartości zmiennej *msgtyp*:

- *msgtyp* > 0 – pobierany jest pierwszy komunikat typu *msgtyp*
- *msgtyp* < 0 – pobierany jest pierwszy komunikat którgo wartość typu jest mniejsza lub równa wartości *msgtyp*
- *msgtyp* = 0 – typ komunikatu nie jest brany, tzn. funkcja pobiera pierwszy komunikat dowolnego typu

msgflg – flaga specyfikujące zachowanie się funkcji w warunkach nietypowych. Wartość ta może być ustawiona na 0, IPC_NOWAIT lub MSG_NOERROR

UWAGI:
Odebranie komunikatu oznacza pobranie go z kolejki, czyli raz odebrany komunikat nie może zostać odebrany ponownie. Argument *msgflg* określa czynność, która jest wykonywana, gdy żądanego komunikatu nie ma w kolejce, lub miejsce przygotowane do odebrania komunikatu jest

niewystarczające. Gdy wartością *msgflg* jest IPC_NOWAIT, funkcja przy żądaniu odbioru komunikatu, którego nie ma w kolejce nie będzie blokowała wywołującego ją procesu. Z kolei flaga MSG_NOERROR powoduje odpowiednie obcinanie rozmiaru komunikatu za dużego, by go odebrać. W przypadku, gdy flaga MSG_NOERROR nie jest ustawiona i otrzymany jest za długi komunikat, to funkcja zakończy się błędem. Gdy nie ma znaczenia fakt, czy komunikaty mają byc obcinane czy nie, flagę *msgflg* należy ustawić na 0.

- **`int msgctl(int msgid, int cmd, struct msgid_ds. *buf)`**

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

Możliwe kody błędów (*errno*) w przypadku błędnego zakończenie funkcji:
EACCES – nie ma praw do odczytu oraz *cmd* jest ustawiona na IPC_STAT
EFAULT – adres wskazywany przez *buf* jest nieprawidłowy
EIDRM – kolejka została usunięta
EINVAL – *msgqid* nieprawidłowe lub *msgsz* mniejsze od 0
EPERM – komendy IPC_SET lub IPC_RMID zostały wydane podczas gdy proces nie ma praw dostępu do zapisu
Argumenty funkcji:
msgid – identyfikator kolejki
cmd – stała specyfikująca rodzaj operacji
- *cmd* = IPC_STAT – pozwala uzyskać informację o stanie kolejki komunikatów
- *cmd* = IPC_SET – pozwala zmienić związane z kolejką ograniczenia
- *cmd* = IPC_RMID – pozwala usunąć kolejkę z systemu

buf – wskaźnik na zmienną strukturalną przez którą przekazywane są parametry operacji

UWAGI:
Funkcja służy do zarządzania kolejką (np. usuwania kolejki, zmiany praw dostępu itp.)

IV. Przykładowe programy obsługujące kolejki komunikatów

Listingi 4.4 i 4.5 przedstawiają rozwiązanie problemu producenta i konsumenta (odpowiednio program producenta i program konsumenta) na kolejce komunikatów. W rozwiązaniu zakłada si ę ograniczone buforowanie, tzn. nie może być więcej nieskonsumowanych elementów, niż pewna założona ilość (pojemność bufora). Rozwiązanie dopuszcza możliwość istnienia wielu producent ów i wielu konsumentów.

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

   #define MAX 10
6
   struct buf_elem {
      long mtype;
9     int mvalue;
   };
  #define PUSTY 1
12 #define PELNY 2
   main(){
15    int msgid, i;
      struct buf_elem elem;
18    msgid = msgget(45281, IPC_CREAT|IPC_EXCL|0600);
      if (msgid == -1){
            msgid = msgget(45281, IPC_CREAT|0600);
21          if (msgid == -1){
                  perror("Utworzenie kolejki ókomunikatw");
                  exit(1);
24          }
      }
      else{
27          elem.mtype = PUSTY;
            for (i=0; i<MAX; i++)
            if (msgsnd(msgid, &elem, sizeof(elem.mvalue), 0) == -1){
30                perror("Wyslanie pustego komunikatu");
                  exit(1);
            }
33    }
      for (i=0; i<10000; i++){
36          if (msgrcv(msgid,&elem,sizeof(elem.mvalue),PUSTY, 0)== -1){
                  perror("Odebranie pustego komunikatu");
                  exit(1);
39          }
            elem.mvalue = i;
            elem.mtype = PELNY;
42          if (msgsnd(msgid, &elem, sizeof(elem.mvalue), 0) == -1){
                  perror("Wyslanie elementu");
                  exit(1);
45           }
      }
   }
```

Listing 1: Impelmentacja zapisu ograniczonego bufora za pomocą kolejki komunikatów

Opis programu: 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).

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

   #define MAX 10
6
   struct buf_elem {
      long mtype;
9     int mvalue;
   };
   #define PUSTY 1
12 #define PELNY 2

   main(){
15    int msgid, i;
      struct buf_elem elem;

18    msgid = msgget(45281, IPC_CREAT|IPC_EXCL|0600);
      if (msgid == -1){
            msgid = msgget(45281, IPC_CREAT|0600);
21          if (msgid == -1){
                  perror("Utworzenie kolejki komunikatów");
                  exit(1);
24          }
      }
      else{
27          elem.mtype = PUSTY;
            for (i=0; i<MAX; i++)
            if (msgsnd(msgid, &elem, sizeof(elem.mvalue), 0) == -1){
30                perror("Wyslanie pustego komunikatu");
                  exit(1);
            }
33    }

      for (i=0; i<10000; i++){
36          if (msgrcv(msgid,&elem,sizeof(elem.mvalue),PELNY,0) == -1){
                  perror("Odebranie elementu");
                  exit(1);
39          }
            printf("Numer: %5d Wartosc: %5d\n", elem.mvalue);
            elem.mtype = PUSTY;
42          if (msgsnd(msgid, &elem, sizeof(elem.mvalue), 0) == -1){
                  perror("Wyslanie pustego komunikatu");
                  exit(1);
45          }
      }
   }
```

Listing 2: Impelmentacja zapisu ograniczonego bufora za pomocą kolejki komunikatów

Opis programu: Analogicznie do poprzedniego przykładu, z tym, że umieszczenie elementu w buforze, reprezentowanym przez kolejkę komunikatów, polega na zastąpieniu komunikatu typu PELNY komunikatem typu PUSTY. Pełny komunikat jest pobierany w linii 36.

V. Zadania do samodzielnego wykonania.

1) Napisz program tworzący dwa procesy komunikujące się poprzez kolejkę komunikatów. Komunikacja polega na przesłaniu komunikatów różnego typu. Proces odbierający powinien odbierać tylko komunikaty typu podanego jako argument wejściowy. (wskazówka: zakładamy że typ komunikatu jest liczbą z przedziału <1,10>.)
2) Napisz programy klienta i serwera realizujące następujący scenariusz: Klient przesyła dowolny ciąg znaków do serwera, serwer w odpowiedzi odsyła ten sam ciąg po zamianie wszystkich małych liter na duże.
3) Napisz programy klienta i serwera realizujące następujący scenariusz: Serwer sumuje liczby wysyłane przez klienta. Liczby są przesyłane jako komunikat typu M_DANE, a ostatnia z liczb jest wysyłana jako komunikat M_END. Po otrzymaniu tego komunikatu serwer odsyła wynik sumowania komunikatem typu M_WYNIK.
4) Napisz program (programy) komunikujące się za pomocą kolejki komunikatów. Każdy z programów wysyła wiadomości wpisywane z klawiatury, a po odbiorze komunikatu z kolejki wypisuje jego treść na ekran. Operacje odczytu i zapisu powinny być wykonywane w sposób asynchroniczny. Program kończy się po otrzymaniu wiadomości o treści ”exit”.

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