7 Kolejki komunikatów

7.1 Wprowadzenie

Kolejki komunikatów (ang. *message queues*) są mechamizmem komunikacji między procesami podobnym do potoków, jednak bardziej wyrafinowanym i dającym większe możliwości. Zostały one wprowadzone w Systemie V w ramach tzw. technik IPC (ang. *interprocess communication*), do których należą jeszcze semafory i pamięć dzielona. Podstawową jednostką informacji używaną w mechanizmie *kolejek komunikatów* jest tzw. *komunikat*, będący pewną stukturą. Struktura ta, oprócz porcji danych, zawiera również identyfikator, zwany *typem komunikatu*, służący do identyfikowania przesyłanych danych. Dzięki temu w ramach *kolejki komunikatów* można tworzyć *podkolejki*, których wyróżnikiem jest właśnie *typ komunikatu*: dana *podkolejka* zawiera *komunikaty określonego typu*. Umożliwia to multipleksowanie zadań z różnych źródeł. *Komunikaty* umieszczane są przez system w *kolejce*, do której *dostęp* odbywa się w trybie *FIFO* (element wpisany jako pierwszy jest odczytany również jako pierwszy). Jeżeli *kolejka* składa się z kilku *podkolejek*, to *dostęp FIFO* dotyczy każdej *podkolejki oddzielnie*.

Z poziomu powłoki informacje o *kolejkach komunikatów* można uzyskiwać przy pomocy komendy: `ipcs -q` (możliwe dalsze opcje, jak w przypadku semaforów i pamięci dzielonej), zaś do usuwania kolejek służy komenda: `ipcrm -msg` *msqid*, gdzie *msqid* jest systemowym identyfikatorem kolejki.

7.2 Tworzenie kolejek komunikatów

Do *tworzenia kolejki komunikatów*, jak również do *odwoływania się* do *istniejącej już kolejki* przez proces, który chce z niej skorzystać, służy funkcja systemowa msgget.

Pliki włączane	`<sys/types.h>, <sys/ipc.h>, <sys/msg.h>`		
Prototyp	`int msgget(key_t key, int msgflg);`		
Zwracana wartość	Sukces	Porażka	Czy zmienia `errno`
	Identyfikator kolejki	-1	Tak

Zakończona sukcesem funkcja msgget tworzy *kolejkę komunikatów* (lub uzyskuje dostęp, jeżeli taka już istnieje) dla danego klucza key (podobnie jak dla semaforów i pamięci dzielonej) i zwraca idendyfikator tej kolejki, będący pewną liczbą całkowią nieujemną. Parametr msgflg określa sposób wykonania funkcji oraz prawa dostępu do kolejki komunikatów.

• **Flagi** msgflg (najważniejsze):

`IPC_CREAT`	utworzenie kolejki komunikatów, jeśli nie istnieje lub uzyskanie dostępu do istniejącej już kolejki;
`IPC_EXCL`	użyta w połączeniu z `IPC_CREAT` zwraca błąd, jeżeli dla danego klucza istnieje już kolejka komunikatów;
Prawa dostępu	podobnie jak dla plików, np. `0666`: możliwość wykonywania operacji czytania i pisania przez wszystkie procesy.

Znaczniki oraz prawa dostępu można łączyć przy pomocy sumy bitowej, np. `IPC_CREAT|0666`.

7.3 Sterowanie kolejkami komunikatów

Do uzyskiwania informacji o kolejce komunikatów lub modyfikowania pewnych jej parametrów służy funkcja systemowa msgctl.

Pliki włączane	`<sys/types.h>, <sys/ipc.h>, <sys/msg.h>`		
Prototyp	`int msgctl(int msqid, int cmd, /* struct msqid_ds *buf */...);`		
Zwracana wartość	Sukces	Porażka	Czy zmienia `errno`
	0	−1	Tak

Funkcja ta wykonuje operacje określone parametrem cmd na kolejce komunikatów o identyfikatorze msqid (zwracanym przez funkcję msgget).

• **Możliwe operacje** cmd:

`IPC_STAT` usyskanie informacji o kolejce komunikatów przechowywanej w strukturze `msqid_ds` zdefiniowanej w pliku nagłówkowym `<sys/msg.h>`;

`IPC_SET` modyfikowanie wybranych pól struktury `msqid_ds` (użytkownik musi najpierw utworzyć strukturę typu `msqid_ds`, zmienić odpowiednie jej pola, a dopiero potem wywołać funkcję `msgctl` ze znacznikiem `IPC_SET` oraz adresem tej struktury);

`IPC_RMID` usunięcie danej kolejki komunikatów (trzeci argument może być tu pominięty).

7.4 Operacje na kolejkach komunikatów

Kolejki komunikatów umożliwiają *wysyłanie* i *odbieranie* niewielkich porcji danych. W pliku nagłówkowym `<sys/msg.h>` zdefiniowany jest *szablon* komunikatu jako następująca struktura:

```
struct msgbuf {
  long mtype;         /* typ komunikatu   */
  char mtext[1];      /* tekst komunikatu */
};
```

W rzeczywistości jest to jedynie sugestia jak definiować komunikat. Użytkownik bowiem może zdefiniować własną strukturę komunikatu. Istotne jest jedynie to, by *pierwsze pole* tej struktury zawierało *typ komunikatu* (liczba typu `long`), natomiast reszta może być dowolna. Wszystko bowiem co następuje po polu typu `long` jest traktowane przez system jako tekst komunikatu. *Typ komunikatu* (najczęściej liczba całkowita dodatnia) umożliwia selektywne wybieranie komunikatów z kolejki. Komunikaty są umieszczane w kolejce w takim porządku w jakim są wysyłane, bez grupowania według typów. Tak więc koncepcja *podkolejek* opisana we podrozdziale 7.1 jest jedynie pewną logiczną abstrakcją, a nie faktyczną organizacją kolejki w systemie. Niemniej jednak, w wielu zastosowaniach wygodnie jest posługiwać się koncepcją podkolejek bez wchodzenia w szczegóły ich fizycznej organizacji (to ostatnie jest raczej sprawą systemu operacyjnego niż użytkownika).

Do *wysyłania* komunikatów do kolejki służy funkcja systemowa `msgsnd`.

Pliki włączane	`<sys/types.h>, <sys/ipc.h>, <sys/msg.h>`		
Prototyp	`int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);`		
Zwracana wartość	Sukces	Porażka	Czy zmienia `errno`
	0	−1	Tak

Funkcja msgsnd *wysyła* do *kolejki* o identyfikatorze `msqid` *komunikat* wskazywany przez msgp o rozmiarze msgsz bajtów (maksymalny rozmiar komunikatu zależy od ustawień systemowych). Parametr msgflg określa zachowanie funkcji w przypadku zapełnienia kolejki komunikatów (osiągnięcia limitu systemowego długości kolejki[5]). Może on przyjmować następujące wartości:

$$\texttt{msgflg} = \begin{cases} \texttt{0}: & \text{operacja blokująca,} \\ \texttt{IPC\_NOWAIT}: & \text{operacja nieblokująca.} \end{cases}$$

Operacja blokująca oznacza wstrzymanie procesu wysyłającego komunikat do zapełnionej kolejki do momentu zwolnienia miejsca w kolejce. W przypadku *operacji nieblokującej* proces nie zostanie wstrzymany, ale funkcja msgsnd zakończy się porażką (komunikat *nie* zostanie wysłany) i ustawi zmienną `errno` na kod `EAGAIN`.

Komunikaty można *pobierać* z kolejki przy pomocy funkcji systemowej `msgrcv`.

Pliki włączane	`<sys/types.h>, <sys/ipc.h>, <sys/msg.h>`		
Prototyp	`int msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp, int msgflg);`		
Zwracana wartość	Sukces	Porażka	Czy zmienia `errno`
	Liczba odebranych bajtów	−1	Tak

Funkcja msgrcv *pobiera* komunikat z kolejki o identyfikatorze msqid i umieszcza go w miejscu pamięci wskazywanym przez msgp (powinien to być wskaźnik do struktury zgodnej z formatem komunikatu) zwracając liczbę faktycznie odebranych bajtów (w przypadku sukcesu). Argument msgsz określa maksymalny rozmiar komunikatu w bajtach – wartość ta powinna być równa najdłuższemu otrzymywanemu komunikatowi. Jeżeli wartośc msgsz będzie mniejsza od rozmiaru komunikatu, a argument msgflg będzie ustawiony na `MSG_NOERROR`, to dojdzie do obcięcia komunikatu (obcięta część będzie stracona), w przeciwnym razie komunikat nie zostanie usunięty z kolejki, a fukcja msgrcv zakończy się porażką generując kod błędu `E2BIG`. Poza tym argument msgflg określa czy pobieranie komunikatu ma być operacją *blokującą* (`0`), czy *nieblokującą* (`IPC_NOWAIT`) – w przypadku braku danego komunikatu w kolejce (niewykluczające się znaczniki można łączyć sumą bitową). Typ komunikatu, który ma być pobrany z kolejki określa argument msgtyp.

[5]Limity systemowe dotyczące mechamizmów IPC można w niektórych wersjach Uniksa sprawdzić przy pomocy komendy: `ipcs -l`.

• **Możliwe wartości** msgtyp **i związane z nimi czynności:**

$= 0$ pobierz pierwszy komunikat *dowolnego* typu;

> 0 pobierz pierwszy komunikat, którego typ będzie *równy* wartości msgtyp;

< 0 pobierz pierwszy komunikat, którego typ będzie *mniejszy* lub *równy* wartości bezwzględnej msgtyp.

Sterując odpowiednio wartościami argumentu msgtyp można łatwo zaimplementować wymianę komunikatów z priorytetami.

ĆWICZENIE 8: Klient–Serwer: Kolejki Komunikatów

Proces *klient* wysyła do procesu *serwera* ciąg znaków. *Serwer* odbiera ten ciąg znaków i *przetwarza* go zamieniając w nim wszystkie litery na wielkie, a natępnie wysyła tak przetworzony ciąg znaków z powrotem do *klienta*. *Klient* odbiera przetworzony ciąg znaków i wypisuje go na ekranie.

Posługując się mechanizmem **kolejek komunikatów** systemu UNIX, zaimplementować powyższe zadanie typu **klient–serwer**, z możliwością obsługi *wielu klientów* naraz. W rozwiązaniu użyć *tylko jednej* kolejki komunikatów. Zastosować odpowiednie etykietowanie komunikatów w celu rozróżniania w kolejce danych *dla serwera* oraz danych *dla poszczególnych klientów*. Najlepiej z każdym komunikatem związać dwa identyfikatory: jeden odpowiadający procesowi adresata, a drugi procesowi wysyłającemu (oczywiście, jeden z nich powinien być umieszczony w obowiązkowym polu *typu komunikatu*).
Spróbować uruchamiać *każdy* proces *klienta* z *innego* X-terminala (np. użyć komendy `xterm -e` *`nazwa_programu`* `&` w pliku `Makefile` do uruchomienia *serwera* i kilku *klientów*; więcej informacji można znaleźć w man xterm).

Podobnie jak dla semaforów, stworzyć własną bibliotekę funkcji do obsługi kolejek.