Systemy Operacyjne 1
Laboratorium 4
„Komunikacja IPC - kolejki komunikatów”
(jeden tydzień)

dr inż. Arkadiusz Chrobot
dr inż. Grzegorz Łukawski

20 października 2020

Wstęp

Ta instrukcja jest pierwszą z trzech dotyczących mechanizmów komunikacji IPC (ang. *InterProcess Communication*), które zostały opracowane dla systemu Unix w wersji System V. Ponieważ system Linux jest z nim kompatybilny, to również on umożliwia programistom aplikacji korzystanie z tych rozwiązań. Niniejsza instrukcja poświęcona jest kolejkom komunikatów, które są jednym z tych mechanizmów. Jej następny rozdział zawiera ogólną charakterystykę komunikacji IPC w wersji dla System V. Kolejny zawiera ogólny opis kolejek komunikatów. Trzeci rozdział poświęcony jest opisom API kolejek komunikatów, czyli plikom nagłówkowym, strukturom danych i funkcjom umożliwiającym procesom użytkownika korzystanie z nich. Ostatni rozdział zawiera zadania do samodzielnego wykonania w ramach zajęć laboratoryjnych.

1. Komunikacja IPC w wydaniu Systemu V

Mechanizmy komunikacji IPC zapożyczone w Linuksie z Uniksa w wersji System V obejmują trzy rodzaje zasobów: *kolejki komunikatów*, *semafory* i *pamięć dzieloną*. Wszystkie one są tworzone i usuwane przez jądro systemu operacyjnego na wniosek procesu użytkownika. Z takich zasobów może korzystać wiele procesów. Stopień zaangażowania systemu operacyjnego w obsługę wspominanych mechanizmów jest różny, w zależności od ich rodzaju. Zawsze jednak umożliwia on poznanie ich liczb i stanu oraz pozwala na ich usunięcie. Służą do tego dwa polecenia powłoki systemu związane z komunikacją IPC: `ipcs` (Szczegóły: `man ipcs`) wyświetlające informacje statystyczne na temat zasobów IPC oraz `ipcrm` usuwające wskazany zasób (Szczegóły `man ipcrm`). Więcej informacji o mechanizmach IPC można uzyskać za pomocą polecenia `man 5 ipc`. Warto zaznaczyć, że system Linux udostępnia także inną implementację opisywanych mechanizmów. Ta inna implementacja jest zgodna ze standardem POSIX i będzie częściowo opisana w instrukcji dotyczącej wątków użytkownika.

2. Kolejki komunikatów

Kolejki komunikatów są tworzone w przestrzeni jądra systemu. Oznacza, to że proces użytkownika nie ma do nich bezpośredniego dostępu, ale może go uzyskać za pośrednictwem wywołań systemowych (dokładniej, poprzez odpowiednie funkcje biblioteczne języka C, które z nich korzystają). Każda kolejka ma swój unikatowy identyfikator. Obsługa kolejek jest mniej skomplikowana niż korzystanie z łącz nazwanych. Ponadto sposób przesyłania nimi informacji jest bardziej elastyczny. Jedna kolejka może służyć kilku procesom. Procesy mogą odbierać wszystkie komunikaty przechodzące przez kolejkę, lub tylko wybrane. Komunikaty te podlegają jednak ograniczeniom zarówno pod względem wielkości pojedynczego komunikatu, jak i sumarycznej wielkości wszystkich komunikatów (całej kolejki). W systemie Linux pierwszy limit wynosi 8 KiB (stała `MSGMAX`), a drugi 16 KiB (stała `MSGMNB`).

3. API kolejek komunikatów

System Linux udostępnia interfejs służący do obsługi kolejek z poziomu języka C. Ten rozdział opisuje jego najważniejsze elementy.

3.1. Pliki nagłówkowe

Każdy program korzystający z kolejki musi mieć włączone do swojego kodu źródłowego następujące plik nagłówkowe: `sys/types.h` - zawiera definicje typów, `sys/ipc.h` - zawiera deklaracje funkcji i definicje struktur wspólnych dla wszystkich mechanizmów IPC, `sys/msg.h` - zawiera funkcje i struktury przeznaczone wyłącznie do obsługi kolejek komunikatów.

3.2. Struktury danych

Struktura komunikatu wysyłanego kolejką może być dowolna, z jednym zastrzeżeniem. Musi ona zawierać obowiązkowe pole określające typ komunikatu. Dalsza część jej definicji jest dowolna. Najczęściej jako przykładową strukturę komunikatu podaje się tę zawartą w listingu 1:

```c
struct msgbuf {
            long mtype;
            char mtext[1];
};
```

Listing 1: Przykładowa struktura komunikatu

Jądro systemu, dla każdej kolejki komunikatów utrzymuje strukturę `msqid_ds`, której zawartość można częściowo modyfikować za pomocą funkcji `msgctl()`. Głównie dotyczy to pól struktury w niej zagnieżdżonej - `msg_perm`:

uid identyfikatora użytkownika dla właściciela,

guid identyfikator grupy dla właściciela,

mode obejmuje 9 najmłodszych bitów określających prawa dostępu do kolejki.

3.3. Funkcje

Do najważniejszych funkcji związanych z obsługą kolejek komunikatów zalicza się:

`ftok()` funkcja ta zwraca unikatowy[1] klucz, który może być użyty do tworzenia zarówno kolejki komunikatów, jak i innych zasobów IPC. Aby dwa niespokrewnione procesy mogły korzystać z tego samego zasobu komunikacyjnego muszą podać te same argumenty dla wywołania tej funkcji (tj. ścieżkę dostępu do dowolnie wybranego pliku i naturalną liczbę ośmiobitową[2]).
Szczegóły: `man ftok`

`msgget()` funkcja, w zależności od ustawień flag z jakim została wywołana, tworzy kolejkę komunikatów i zwraca jej identyfikator lub zwraca identyfikator istniejącej kolejki. Kolejka jest tworzona na podstawie unikatowego klucza zwróconego przez funkcję `ftok()` lub dobranego przez programistę. Jeśli kolejka ma być stworzona tylko i wyłącznie na użytek jednego programu[3], to jako klucz podaje się stałą `IPC_PRIVATE`. Drugi argument wywołania tej funkcji określa prawa dostępu (trzycyfrowa liczba ósemkowa, lub odpowiednia kombinacja stałych `MSG_R` i `MSG_W`) oraz może zawierać flagi określające, czy ma być uzyskany identyfikator nowej kolejki, czy już istniejącej (`IPC_CREAT` i `IPC_EXCL`).
Szczegóły: `man msgget`

`msgsnd()` funkcja ta umożliwia dodanie komunikatu do kolejki (wysłanie komunikatu za jej pośrednictwem). Przyjmuje ona cztery argumenty wywołania. Pierwszym jest identyfikatorem kolejki zwrócony przez `msgget()`, drugim jest adres struktury zawierającej komunikat do nadania, trzecim jest rozmiar treści komunikatu (rozmiar struktury komunikatu, bez rozmiaru pola typu). W zależności od wartości ostatniego argumentu (stała `IPC_NOWAIT` lub zero) funkcja może, w przypadku kiedy kolejka jest zapełniona, oczekiwać na zwolnienie miejsca lub natychmiast kończyć swe działanie, zwracając wyjątek. W przypadku powodzenia `msgsnd()` zwraca zero, w przeciwnym przypadku `-1`.
Szczegóły: `man msgsnd`

`msgrcv()` funkcja umożliwia odbiór komunikatu z kolejki. Przyjmuje ona pięć argumentów wywołania: identyfikator kolejki, adres bufora do którego zostanie zapisany odebrany komunikat, rozmiar treści tego komunikatu (patrz opis poprzedniej funkcji), typ odbieranego komunikatu oraz flagi. Argument typu komunikatu umożliwia selektywny odbiór komunikatów. Jeśli będzie on miał wartość `0`,

[1] Jest to założenie idealne. W praktyce zdarzają się kolizje (powtórki). Oznacza to, ze dla dwóch różnych par argumentów wywołania funkcja `ftok()` może zwrócić takie same wartości.

[2] Chociaż argument, przez który przekazywana jest ta liczba ma typ `int`, to przez funkcję branych jest pod uwagę tylko osiem jego najmłodszych bitów.

[3] Należy przez to rozumieć, że będzie z niej korzystało kilka spokrewnionych procesów, choć korzystanie z kolejki komunikatów przez wyłącznie jeden proces też jest możliwe, ale niepraktyczne.

to będzie odebrany pierwszy komunikat znajdujący się w kolejce, jeśli wartość dodatnią, to odebrany będzie pierwszy komunikat o takim samym typie. Jeśli natomiast argument ten będzie miał wartość ujemną, to odebrany będzie komunikat o typie takim samym lub mniejszym co do wartości bezwzględnej od wartości tego argumentu. Ostatni argument funkcji może być zerem lub stałą `IPC_NOWAIT`. W przypadku, kiedy jest on tą stałą, to funkcja `msgrcv()`, jeśli zostanie wywołana dla pustej kolejki, natychmiast zakończy swoje działanie i zasygnalizuje wyjątek. Jeśli jest on jednak zerem, to funkcja ta będzie czekała tak długo, aż pojawi się w kolejce komunikat do odebrania. Funkcja `msgrcv()` zwraca rozmiar odebranej treści komunikatu lub `-1` w przypadku niepowodzenia odbioru.
Szczegóły: `man msgrcv`

`msgctl()` funkcja ta umożliwia sterowanie istniejącą kolejką komunikatów. Przyjmuje ona trzy argumenty wywołania: identyfikator kolejki, dla której ma być przeprowadzona operacja, stałą określającą rodzaj operacji oraz adres struktury typu `struct msqid_ds`. Może ona wykonywać trzy operacje:

`IPC_STAT` pobranie do struktury `msqid_ds` informacji o kolejce,

`IPC_SET` ustawienie danych kolejki na podstawie zawartości struktury `msqid_ds`,

`IPC_RMID` usunięcie kolejki - w jej przypadku ostatni argument funkcji `msgctl()` może mieć wartość `NULL` lub `0`.

Ostatnia operacja powinna być przeprowadzana tylko wtedy, gdy wszystkie procesy korzystające z kolejki zakończą operacje przeprowadzane na niej. Inaczej te operacje zostaną przerwane i zasygnalizują wyjątki. Funkcja zwraca `0` w przypadku powodzenia lub `-1` w przeciwnym przypadku.
Szczegóły: `man msgctl`

Listingi 3 i 2 zawierają przykłady dwóch prostych programów, które przesyłają sobie jednokierunkowo komunikaty z użyciem kolejki.

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

struct msgbuf {
        long type;
        char mtext[50];
};

void receive_message(int mqid)
{
        struct msgbuf buffer;

        if(msgrcv(mqid,&buffer,sizeof(buffer.mtext),1,0)<0)
                perror("msgrcv");
        else
                printf("Odebrany komunikat: %s\n",buffer.mtext);
}

int main(void)
{
        int key = ftok("/tmp",8);
        if(key<0)
                perror("ftok");

        int id = msgget(key,0600|IPC_CREAT|IPC_EXCL);
        if(id<0)
                perror("msgget");

        receive_message(id);

        if(msgctl(id,IPC_RMID,0)<0)
                perror("msgctl");

        return 0;
}
```

Listing 2: Przykładowy program tworzący kolejkę i odbierający z niej komunikaty

Za utworzenie kolejki komunikatów odpowiedzialny jest program odbierający z niej komunikaty. Najpierw pozyskuje on za pomocą `ftok()` klucz, który później jest wykorzystany przy tworzeniu wspomnianego zasobu przez funkcję `msgget()`. Połączenie flag `IPC_CREAT` i `IPC_EXCL` powoduje, że jeśli kolejka nie istnieje, to zostanie ona utworzona, a jeśli istnieje to funkcja zasygnalizuje wyjątek. Po utworzeniu kolejki program próbuje z niej odebrać komunikat o wartości typu również `1`. Jeśli kolejka będzie pusta lub nie będzie w niej komunikatu o takim typie, to program będzie tak długo czekał, aż się on w niej pojawi. Po odebraniu komunikatu program wyświetla na ekranie jego treść, usuwa kolejkę i kończy swe działanie. Powinien on być uruchamiany przed nadawcą, ponieważ to on tworzy kolejkę[4].

[4] Jest to uproszczenie. W rzeczywistości kolejkę tworzy jądro systemu operacyjnego na wniosek tego programu.

```c
#include<stdio.h>
#include<string.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#include<sys/types.h>

#define TEXT_LENGTH 50

struct msgbuf {
        long type;
        char mtext[TEXT_LENGTH];
};

void send_message(int mqid, char message[])
{
        struct msgbuf buffer;

        buffer.type = 1;
        memset(buffer.mtext,0,sizeof(buffer.mtext));
        strncpy(buffer.mtext,message,TEXT_LENGTH-1);

        if(msgsnd(mqid,&buffer,sizeof(buffer.mtext),0)<0)
                perror("msgsnd");
}

int main(void)
{
        int key = ftok("/tmp",8);
        if(key<0)
                perror("ftok");

        int id = msgget(key,0600);
        if(id<0)
                perror("msgget");

        send_message(id,"Systemy Operacyjne");

        return 0;
}
```

Listing 3: Przykładowy program nadający komunikaty do kolejki

Program z listingu 3 uzyskuje identyfikator istniejącej już kolejki nadaję za jej pomocą komunikat o wartości typu 1. Treść komunikatu kopiowana jest w bezpieczny sposób, tzn. najpierw pole bufora na tę treść jest zerowane przy pomocy funkcji `memset()`, a następnie do tego pola jest kopiowanych maksymalnie o jeden mniej znaków niż może ono pomieścić. Proszę zwrócić uwagę na argumenty wywołania funkcji `ftok()` są one identyczne, jak w poprzednim programie, dzięki temu nadawca uzyska identyfikator tej samej kolejki co odbiorca. Proszę także zauważyć, że do wywołania funkcji `msgget()` nie są przekazywane żadne flagi.

Zadania

Uwaga: We wszystkich programach tuż przed zakończeniem ich działania wszystkie kolejki komunikatów z jakich one korzystają powinny zostać usunięte. Programy muszą być napisane z podziałem na funkcje z parametrami oraz muszą sprawdzać, czy wywoływane przez nie funkcje z api systemu operacyjnego nie sygnalizują wyjątków.

1. Napisz program, który utworzy prywatną kolejkę i będzie przesyłał nią komunikaty w obrębie pojedynczego procesu.

2. Zmodyfikuj zadanie pierwsze, tak aby proces tworzył potomka i komunikował się z nim przy użyciu kolejki komunikatów.

3. Napisz dwa programy, które będą komunikowały się przy pomocy kolejki komunikatów. Do wygenerowania klucza użyj funkcji `ftok()`. Zbadaj jak się będzie zachowywała się funkcja `msgrcv()`, w zależności od tego, czy będzie wywołana z flagą `IPC_NOWAIT`, czy nie.

4. Napisz dwa programy: pierwszy wyśle kilka komunikatów o losowo wybranym typie (przyjmijmy, że typy komunikatów należą do przedziału [1,5]), a drugi odbierze je w porządku malejącym lub rosnącym, ze względu na wartość typu i wybór użytkownika.

5. Napisz dwa programy: pierwszy wyśle dziesięć komunikatów, a drugi w zależności od argumentu wywołania odbierze je w takiej samej lub odwrotnej kolejności do tej, z jaką wysłał je pierwszy program.

6. Napisz trzy programy, które będą się komunikowały przez wspólną kolejkę. Komunikacja musi być dwukierunkowa. Każdy program musi odbierać informacje przeznaczone tylko dla niego, oraz wysyłać komunikaty do obu pozostałych programów.

7. Za pomocą kolejki komunikatów programy mogą przesyłać wiadomości o różnej wielkości. Napisz dwa programy komunikujące się przez jedną kolejkę, ale przesyłające między sobą komunikaty o zmiennej wielkości. Załóż, że każdy „właściwy” komunikat będzie poprzedzany komunikatem o ustalonym formacie i wielkości, który będzie informował o wielkości następującego po nim komunikatu.

8. Napisz trzy programy. Pierwszy niech zapisuje do kolejki liczby parzyste, drugi nieparzyste, a trzeci niech odczytuje kolejne pary liczb z kolejki i niech je sumuje.