Instytut Teleinformatyki

Wydział Inżynierii Elektrycznej i Komputerowej
Politechnika Krakowska

programowanie usług sieciowych

„IPC Systemu V”

laboratorium: 08

Kraków, 2014

Spis treści

1. Wiadomości wstępne

Pierwsza część niniejszej instrukcji zawiera podstawowe wiadomości teoretyczne dotyczące **komunikacji międzyprocesowej IPC** (ang. *Interprocess Communication*). Poznanie tych wiadomości umożliwi prawidłowe zrealizowanie praktycznej części laboratorium.

1.1. Tematyka laboratorium

Tematyką laboratorium są podstawowe mechanizmy komunikacji między procesami – IPC (ang*. Inter-Process Communication*) w Systemie V. Omawiane są zalety i wady poszczególnych technik jak również ich przydatność. Dokument dotyczy również wybranych metod synchronizacji między procesami. Przez pojęcie komunikacji międzyprocesowej Systemu V określamy trzy typy IPC:

- ***kolejki komunikatów*** (ang. *message queues*) ,
- ***semafory*** (ang. *semaphore arrays*) ,
- ***pamięć wspólna*** (ang. *shared memory segmennts*).

Termin ten jest stosowany w odniesieniu do wymienionych metod komunikacji międzyprocesowej, przede wszystkim do zaznaczenia faktu, że ich korzenie tkwią w Systemie V. Funkcje przeznaczone do obsługi obiektów IPC mają wiele elementów wspólnych. Oto niektóre z nich:

- ***identyfikator,***
- ***klucz,***
- ***prawa dostępu,***
- ***ograniczenia konfiguracyjne***.

1.2. Zagadnienia do przygotowania

Przed przystąpieniem do realizacji laboratorium należy zapoznać się z zagadnieniami dotyczącymi **procesów w systemie UNIX**:

- Pobieranie identyfikatora (`getpid, getppid`),
- Tworzenie (`fork`),
- Zakończenie (`exit)`,
- Oczekiwanie (`wait, waitpid`),
- Uruchomienie (`exec`).

Konieczne jest również poznanie mechanizmów **mapowania pamięci:**

- Mapowanie (`mmap`),
- Zwolnienie (`munmap`),
- Synchronizacja (`msync`),

Oraz mechanizmów **alokacji pamięci**:

- Alokacja (`malloc, calloc`),
- Realokacja (`realloc`),
- Zwolnienie (`free`).

Literatura:

[1] W.R. Stevens, *„Programowanie w środowisku systemu UNIX”*, rozdział 8
[2] W.R. Stevens, „UNIX - Programowanie usług sieciowych, tom. 2”, rozdział 12
[3] http://atos.wmid.amu.edu.pl/~maciejf/materials/sop322/unixipc2.html#1
[4] http://rainbow.mimuw.edu.pl/SO/Linux/Temat03/ipc.html

1.3. Cel laboratorium

Celem laboratorium jest zapoznanie się z podstawowymi metodami komunikacji między procesami oraz prostą metodą synchronizacji. Techniki te są bazą w tworzeniu wieloprocesowych/wielowątkowych aplikacji typu klient serwer. Podczas realizacji tego laboratorium zapoznasz się z:

- wymianą informacji między procesami poprzez kolejkę komunikatów;
- wymianą informacji między procesami poprzez pamięć dzieloną;
- synchronizacją procesów przy pomocy semaforów.

Parametr/opcja	Opis

2. Przebieg laboratorium

Druga część instrukcji zawiera zadania do praktycznej realizacji, które demonstrują zastosowanie technik z omawianego zagadnienia.

2.1. Przygotowanie laboratorium

Przed przystąpieniem do ćwiczenia należy skopiować wszystkie pliki z katalogu

```
/home/shared/pus/pus_08_ipc_sysv
```

do katalogu domowego. Są to źródła programów, które będą wykorzystywane do laboratorium, część z nich będzie modyfikowana podczas ćwiczenia, warto więc mieć dodatkową kopię zapasową.

2.2. Zadanie 1. Działanie polecenia ipcs oraz ipcrm

Zadanie będzie polegało na skompilowaniu trzech programów tworzących kolejkę komunikatów, semafor oraz dostęp do pamięci dzielonej oraz obserwowaniu powstałych wszystkich obiektów IPC Systemu V dzięki opcji `ipcs` oraz zapoznanie się z działaniem tego polecenia oraz polecenia `ipcrm`.

1. Przejdź do katalogu ze źródłami laboratorium:
   ```
   $ cd zad1/
   ```
2. Skompiluj plik queue.c do postaci binarnej:
   ```
   $ gcc queue.c -o queue
   ```
3. Uruchom skompilowany program:
   ```
   $ ./queue
   ```
4. Wydaj polecenie :
   ```
   $ ipcs -a
   ```


-q	Pokaż tylko kolejki wiadomości
-s	Pokaż tylko semafory
-m	Pokaż tylko pamieć dzieloną
-a	Pokaż status wszystkich obiektów
-h	Dodatkowe argumenty

5. Powtórz ćwiczenie od 1-4 dla plików `semaphore.c` oraz `memory_shared.c`
6. Poleceniem `ipcs` sprawdź obiekty, których jesteś właścicielem i sprawdź IPC ID, następnie usuń utworzone obiekty IPC, które stworzyłeś za pomocą polecenia `ipcrm` z odpowiednimi opcjami. Poniżej podaje przykład :

```
$ ipcrm msg 1736732
```

2.3. Zadanie 2. Kolejki komunikatów

Należy skompilować przykładowe programy przy pomocy poleceń:

1. Przejdź do katalogu ze źródłami laboratorium:

```
$ cd zad2/
```

2. Skompiluj programy:

```
$ gcc send.c -o send
$ gcc remove.c -o remove
$ gcc receive.c -o receive
```

3. Następnie uruchamiamy program, który tworzy i wysyła komunikat do kolejki (treść komunikatu w postaci liczby całkowitej podajemy jako parametr):

```
$./send 45
```

Po tym jak program zakończy działanie sprawdzamy czy kolejka została utworzona, jakie są do niej uprawnienia, identyfikator, klucz, ile bajtów i wiadomości zalega w kolejce. Korzystamy z narzędzia `ipcs`. Program `send` możemy uruchomić kilkukrotnie, za każdym razem sprawdzając jak kolejka się zmienia.

4. Kolejnym krokiem jest uruchomienie programu pobierającego komunikaty z kolejki:

```
$./receive
```

Każdorazowo sprawdzamy narzędziem `ipcs` ile komunikatów jeszcze pozostało w kolejce. Sprawdzamy też sytuację w której brak komunikatów a my żądamy odczytu.

5. Po skończonym testowaniu usuwamy kolejkę:


```
$./remove
```

6. Sprawdzamy narzędziem czy kolejka została usunięta.

2.4. Zadanie 3. Interaktywne tworzenie kolejki komunikatów.

Zadanie polega na stworzeniu kolejki komunikatów, przy czym użytkownik podczas wykonywania programu ma wpływ na to tym jaki będzie klucz IPC kolejki, prawa dostępu do kolejki oraz zestaw flag funkcji *msgget.*

Jeśli wywołanie jest pomyślne i utworzona zostanie nowa kolejka lub udostępniona istniejąca, funkcja *msgget* powinna zwrócić identyfikator kolejki komunikatów (ang. message queue identifier) w postaci nieujemnej liczby całkowitej.

Gdy tworzona jest kolejka komunikatów, do nadania jej uprawnień używane jest dziewięć mniej znaczących bitów permflags. Są one pamiętane w strukturze ipc_perm, tworzonej razem z samą kolejką. Podczas wykonywania programu użytkownik musi podać trzy parametry, ktorych opis znajduje się w poniższej tabeli.

Parametr/opcja	**Opis**
`key`	Liczba całkowita - klucz kanału IPC dla tworzonej kolejki
`perm`	Prawa dostępu do kolejki – w postaci np. 0777
`flags`	Liczba całkowita - flagi, z którymi chcesz utworzyć kolejkę 0 – bez flag 1 - IPC_CREAT 2 - IPC_CREAT and IPC_EXCL

Uruchomienie programu:

1. Przejdź do katalogu ze źródłami zadania 3:
   ```
   $ cd ~/pus/pus_08_ipc_sysv/zad3
   ```
2. Skompiluj źródła do postaci binarnej:
   ```
   $ make
   lub
   $ cc -o p3 p3.c
   ```
3. Uruchom program:
   ```
   $ ./p3
   ```

W rezultacie działania programu otrzymujemy numer ID utworzonej kolejki (lub komunikat o błędzie, jeśli którąś z wartości podamy nieprawidłowo).

Przed wykonaniem ćwiczenia należy sprawdzić przy pomocy polecenia *ipcs* informacje o wszystkich obiektach IPC istniejących w systemie.

```
$ ipcs -a
```

Informację o samych kolejkach komunikatów wyświetli polecenie:

```
$ ipcs -q
```

Po wykonaniu ćwiczenia należy zaobserwować czy zaszły jakieś zmiany przy pomocy tych samych poleceń.

2.5. Zadanie 4. Komunikacja przy użyciu pamięci dzielonej

W zadaniu tym, zostanie pokazane, jak stworzyć oraz wykorzystywać kanał IPC pamięci dzielonej oraz podstawowe operacje na segmencie pamięci, jak dodawanie danych oraz pobieranie danych z segmentu pamięci dzielonej.

Segmenty pamięci dzielonej są tworzone przez funkcję pierwotną ***shmget*** zdefiniowaną następująco:

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

int shmget(key_t key, int size, int shmflg);
```

- ***key*** - określa klucz IPC identyfikujący segment pamięci dzielonej,
- ***size*** - określa rozmiar żądanego segmentu. Jeśli funkcja shmget odwołuje się do istniejącego segmentu pamięci, a żądany rozmiar jest większy niż rzeczywisty rozmiar segmentu, to wywołanie funkcji systemowej zakończy się błędem,
- ***shmflg*** - podaje znaczniki sterujące działaniem funkcji shmget i zawiera prawa dostępu do segmentu i bity znaczników, takie jak IPC_PRIVATE czy IPC_CREAT. Prawa dostępu określają, które procesy mają prawo przyłączyć dany segment. Jeśli proces nie ma ani prawa odczytu, ani zapisu do segmentu, to nie może tego segmentu przyłączyć. Jeśli program spróbuje zapisu do segmentu, który jest tylko do odczytu, to dostanie stygnał SIGSEGV.

Funkcja ***shmat*** - dołącza segment pamięci dzielonej do przestrzeni adresowej procesu

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

void *shmat(int shmid, const void *shmaddr, int shmflg);
```

Funkcja zwraca adres, pod którym segment został przyłączony do procesu lub błąd: errno = EINVAL (nieprawidłowy ID IPC lub podano adress podłączenia)

ENOMEM (brak pamięci aby przyłączyć segment)
EACCES (brak pozwolenia)

Najważniejsze fragmenty programu:

```
#define SHM_SIZE 1024  /*tworzymy 1 KB segment pamięci */
int main(int argc, char *argv[]) {
    key_t key;
    int shmid;  //identyfikator segmentu
    char *data;
    int mode;

    /* Tworzenie klucza */
    if ((key = ftok("dzielona.c", 'R')) == -1) {
        perror("Błąd podczas wykonywania funkcji ftok");
        exit(1);
    }
    /* Przyłącz się i stwórz segment */
    if ((shmid = shmget(key, SHM_SIZE, 0644 | IPC_CREAT)) == -1) {
        perror("Błąd podczas wykonywania funkcji  shmget");
        exit(1);
    }
    /* Dołącz się do segmentu, żeby uzykać wskaźnik do niego  */
    data = shmat(shmid, (void *)0, 0);
    if (data == (char *)(-1)) {
        perror("Błąd podczas wykonywania funkcji shmat");
        exit(1);
    }

    /* Czytaj i modyfikuj segment*/
    if (argc == 2) {
        printf("Zapisywanie do segmentu : \"%s\"\n", argv[1]);
        strncpy(data, argv[1], SHM_SIZE);
    }
    else {
        printf("Segment zawiera: \"%s\"\n", data);
    }

    /* Odłącz się z segmentu */
    if (shmdt(data) == -1) {
        perror("shmdt");
        exit(1);
    }
    return 0; }
```

Program pobiera z wiersza poleceń jeden argument. Określa on string zapisywany do pamięci dzielonej.

Parametr/opcja	Opis
`char **argv`	Tablica z argumentami przekazanymi do programu
`input_string`	String – określa tekst zapisywany do pamięci dzielonej

Uruchomienie programu:

1. Przejdź do katalogu ze źródłami zadania 4.
   ```
   $ cd ~/pus/pus_08_ipc_sysv/zad4
   ```
2. Skompiluj źródła do postaci binarnej:
   ```
   $ make
   lub
   $ cc -o shared_memory shared_memory.c
   ```
3. Uruchom program:
   ```
   $ ./shared_memory  - w celu wyświetlenia zawartości pamięci dzielonej
   lub
   $ ./shared_memory input_string – w celu zapisania jakiegoś stringu do pamięci dzielonej
   ```

W rezultacie działania programu otrzymujemy numer utworzonego segmentu (lub komunikat o błędzie, jeśli segmentu pamięci nie utworzono).

Przed wykonaniem ćwiczenia należy sprawdzić przy pomocy polecenia *ipcs* informacje o wszystkich obiektach IPC istniejących w systemie.

```
$ ipcs -a
```

Informację o samych kolejkach komunikatów wyświetli polecenie:

```
$ ipcs –m
```

Testujemy działanie pamięci dzielonej podając na wejście programu różne ciągi znaków. Następnie przy wywołaniu programu bez parametrów sprawdzamy poprawność zapisanej wiadomości. Informacje podane do zapisu powinny zgadzać się z informacjami z odczytu.

Po wykonaniu ćwiczenia należy zaobserwować czy zaszły jakieś zmiany przy pomocy tych samych poleceń.

Po zakończeniu pracy z segmentem pamięci należy usunąć segment przy pomocy polecenia podanego poniżej:

```
$ ipcrm –m numer_segmentu
```

gdzie numer jest numerem uzyskanym z wykonania polecenia ipcs –m.

2.6. Zadanie 5. Semafory

Napisz program, który tworzy jednego potomka. Program (rodzic jak i potomek) winien ciągle wypisywać ten sam komunikat – dwa razy swój identyfikator procesu (w osobnych liniach). Wypisywanie ma być realizowane przy pomocy dwóch oddzielnych instrukcji tzn.

```
printf(”%d\n”, getpid());
printf(”%d\n”, getpid());
```

Nie jest dopuszczalna sytuacje, w której procesy będą wypisywały swoje numery naprzemiennie tzn.

```
100    //rodzic
101    //potomek
100    //rodzic
101    //potomek
```

Użyj semafora systemu V, by ochronić blok wypisywania. Poprawne wyjście powinno wyglądać następująco (przykład):

```
100    //rodzic
100    //rodzic
101    //potomek
101    //potomek
```

2.7. Zadanie 6. Pamięć wspólna

Napisz program klienta oraz program serwera. Serwer powinien umieścić w segmencie pamięci wspólnej imię autora, zaś klient powinien z tego samego segmentu, odczytać je, a następnie „powitać autora”.

Wskazówka:
Przy tworzeniu segmentu użyj tego samego klucza tak, aby segment był dostępny zarówno dla serwera jak i klienta.

3. Opracowanie i sprawozdanie

Realizacja laboratorium pt. *„IPC Systemu V”* polega na wykonaniu wszystkich zadań programistycznych podanych w drugiej części tej instrukcji. Wynikiem wykonania powinno być sprawozdanie w formie wydruku papierowego dostarczonego na kolejne zajęcia licząc od daty laboratorium, kiedy zadania zostały zadane.
Sprawozdanie powinno zawierać:

- opis metodyki realizacji zadań (system operacyjny, język programowania, biblioteki, itp.),
- algorytmy wykorzystane w zadaniach (zwłaszcza, jeśli zastosowane zostały rozwiązania nietypowe),
- opisy napisanych programów wraz z opcjami,
- trudniejsze kawałki kodu, opisane tak, jak w niniejszej instrukcji,
- uwagi oceniające ćwiczenie: trudne/łatwe, nie/realizowalne podczas zajęć, nie/wymagające wcześniejszej znajomości zagadnień (wymienić jakich),
- wskazówki dotyczące ewentualnej poprawy instrukcji celem lepszego zrozumienia sensu oraz treści zadań.

