Systemy Operacyjne #7

API pamięci współdzielonej w systemach Linux i Windows

swernikowski@zut.edu.pl

(wersja: 2021-10-1)

Schemat postępowania z pamięcią współdzieloną w systemie Linux

- pamięć współdzielona jest w systemie Linux jednym z obiektów podsystemu **IPC** (ang. *Inter-Process Communication*)
- interfejs zarządzania obiektami IPC jest w dużej mierze ujednolicony i obejmuje również semafory i przesyłanie komunikatów
- w przypadku pamięci współdzielonej korzystanie z niej wymaga następujących kroków:

1) wygenerowanie klucza obiektu IPC (klucz może zostać podany jawnie, ale zalecane jest uzyskanie go z nazwy wybranego pliku)
   funkcja **`ftok()`** (ang. *file to key*)
2) przydzielenie bloku pamięci współdzielonej i pobranie od systemu identyfikatora tego bloku
   funkcja **`shmget()`** (ang. *shared memory get*)
3) dołączenie bloku pamięci współdzielonej do przestrzeni adresowej procesu
   funkcja **`shmat()`** (ang. *shared memory attach*)
4) wykorzystanie dostępu do pamięci
5) odłączenie bloku pamięci od przestrzeni adresowej procesu
   funkcja **`shmdt()`** (ang. *shared memory detach*)
6) zwolnienie bloku pamięci współdzielonej
   funkcja **`shmctl()`** (shared memory control)

Wygenerowanie klucza obiektu – funkcja `ftok()`

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

key_t ftok(const char *pathname, int proj_id);
```

Wejście:

- `pathname` – nazwa pliku, który ma zostać użyty do wygenerowania klucza; plik musi istnieć!
- `proj_id` – dana całkowita, której 8 najmłodszych bitów zostanie użytych do wygenerowania klucza; wartość na tych bitach musi być niezerowa; pozwala uzyskiwać wiele różnych kluczy z tej samej nazwy pliku

Wynik:

- wygenerowany klucz albo -1 w przypadku błędu (`errno` zna szczegóły)

Przydzielenie bloku pamięci współdzielonej – funkcja `shmget()`

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

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

Wejście:
- `key` – klucz identyfikujący przydzielany blok
- `size` – rozmiar przydzielanego bloku pamięci w bajtach
- `shmflag` – opcje przydziału: składają się praw nadawanych blokowi pamięci (takich samych jak w przypadku plików) oraz sumy bitowej z flag:
  - `IPC_CREAT` – utwórz nowy segment; jeśli tej opcji nie podano, funkcja będzie szukać bloku przydzielonego w innym procesie
  - `IPC_EXCL` – używane w połącznie z `IPC_CREAT` dla zapewnienia, że faktycznie zostanie utworzony nowy blok; jeśli blok już istnieje, funkcja zwróci błąd

Wynik:
- identyfikator przydzielonego bloku albo `-1` w przypadku błędu (`errno` zna szczegóły)

Dołączenie bloku pamięci współdzielonej – funkcja `shmat()`

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

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

Wejście:
- `shmid` – identifikator istniejącego bloku pamięci współdzielonej
- `shmaddr` – adres, do którego należy dołączyć blok, albo `NULL`, jeśli system ma sam wybrać adres
- `shmflag` – opcje dołączenia: suma bitowa z flag:
  - `SHM_EXEC` – zezwolenie na wykonanie kodu w dołączonym bloku (trzeba mieć uprawnienie **x** do tego bloku)
  - `SHM_RDONLY` – blok będzie wykorzystywany tylko do odczytu (brak tej opcji oznacza dostęp **rw**)

Wynik:
- lokalny dla procesu adres dołączonego bloku albo `(void *)-1` w przypadku błędu (`errno` zna szczegóły)

Odłączenie bloku pamięci współdzielonej – funkcja `shmdt()`

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

int shmdt(const void *shmaddr);
```

Wejście:
- `shmaddr` – adres odłączanego bloku

Wynik:
- `0` w przypadku udanego odłączenia albo `-1` w przypadku błędu (`errno` zna szczegóły)

Zwolnienie bloku pamięci współdzielonej – funkcja `shmctl()`

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

int shmctl(int shmid, int cmd, struct shmid_ds *buf);
```

Wejście:

- `shmid` – identyfikator zwalnianego bloku
- `cmd` – polecenie do wykonania przez funkcję, w przypadku zwalniania ma być równy `IPC_RMID`; faktyczne zwolnienie bloku nastąpi w chwili, gdy żaden proces nie będzie już dołączony do tego bloku
- `buf` – adres na strukturę z parametrami wykonywanego polecenia, dla `IPC_RMID` adres ten ustawia się na `NULL`

Wynik:

- `0` w przypadku udanego zwolnienia albo `-1` w przypadku błędu (`errno` zna szczegóły)

Program demonstracyjny

Założenia:

- uruchamiamy dwa procesy – nadajnik i odbiornik

- nadajnik startuje pierwszy i dostaje 2 argumenty: nazwę pliku do wygenerowania klucza i komunikat (string)
- nadajnik przydziela blok pamięci współdzielonej i kopiuje do niego treść komunikatu
- nadajnik czeka na naciśnięcie klawisza <Enter> przez użytkownika
- nadajnik zwalnia przydzieloną pamięć, usuwa ją z systemu i kończy pracę

- odbiornik dostaje 1 argument: nazwę pliku do wygenerowania klucza
- odbiornik przydziela blok pamięci współdzielonej i wypisuje na stdout odczytany z pamięci komunikat
- odbiornik odłącza przydzieloną pamięć i kończy pracę

Nadajnik: plik `lin_sender.c` (część 1/3)

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

#define MEM_SIZE   1000

int main(int argc, char *argv[]) {
    if (argc != 3) {
        fprintf(stderr, "usage: %s key_name message\n", argv[0]);
        return 1;
    }
:
:
```

Nadajnik: plik `lin_sender.c` (część 2/3)

```
:
:
key_t key = ftok(argv[1], 'a');
if(key < 0) {
   fprintf(stderr, "Error creating key\n");
   return 2;
}
int shmid = shmget(key, MEM_SIZE, 0666 | IPC_CREAT | IPC_EXCL);
if(shmid < 0) {
   fprintf(stderr, "Error getting shared memory\n");
   return 3;
}
:
:
```

Nadajnik: plik `lin_sender.c` (część 3/3)

```
:
void *memseg = shmat(shmid, NULL, 0);
if(memseg == (void *)-1) {
    fprintf(stderr, "Error attaching shared memory segment\n");
    return 4;
}
strcpy(memseg, argv[2]);
puts("Press <Enter> to continue...");
getchar();
if(shmdt(memseg) == -1) {
    fprintf(stderr, "Error detaching shared memory segment\n");
    return 5;
}
if(shmctl (shmid, IPC_RMID, NULL) == -1) {
    fprintf(stderr, "Error removing sharem memory\n");
    return 6;
}
return 0;
```

Nadajnik: plik `lin_receiver.c` (część 1/2)

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

#define MEM_SIZE   1000

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "usage: %s key_name\n", argv[0]);
        return 1;
    }
    key_t key = ftok(argv[1], 'a');
    if(key < 0) {
        fprintf(stderr, "Error creating key\n");
        return 2;
    }
    :
```

Nadajnik: plik `lin_receiver.c` (część 2/2)

```
  :
  int shmid = shmget(key, MEM_SIZE, 0666);
  if(shmid < 0) {
      fprintf(stderr, "Error getting shared memory\n");
      return 3;
  }
  void *memseg = shmat(shmid, NULL, 0);
  if(memseg == (void *)-1) {
      fprintf(stderr, "Error attaching shared memory segment\n");
      return 4;
  }
  printf("Message: '%s'\n", (char *)memseg);
  if(shmdt(memseg) == -1) {
      fprintf(stderr, "Error detaching shared memory segment\n");
      return 5;
  }
  return 0;
}
```

Narzędzi konsole

- Linux udostępnia narzędzia konsolowe, przy pomocy których można zarządzać obiektami IPC, w tym pamięcią współdzieloną:
- `ipcmk --shmem <size> --mode <prawa>`
  tworzy blok pamięci współdzielonej i wypisuje jego identyfikator na stdout; pominięcie parametru `--mode` ustawia prawa na `0644`
- `ipcs --shmem`
  wyprowadza na stdout listę wszystkich przydzielonych bloków pamięci współdzielonej
- `ipcrm --shmem-id <id>`
  usuwa blok pamięci o podanym identyfikatorze
- `ipcrm --shmem-key <key>`
  usuwa blok pamięci o podanym kluczu

Schemat postępowania z pamięcią współdzieloną w systemie Windows

- pamięć współdzielona jest w systemie Windows szczególnym przypadkiem usługi mapowania plików tzn. odwzorowywania zawartości pliku dyskowego na obszar pamięci operacyjnej
- wykorzystanie pamięci współdzielonej wymaga następujących kroków:

1) wykonania pozornego mapowania pliku (nieistniejącego) na pamięć operacyjną
   funkcja `CreateFileMapping()` jeśli blok jest tworzony od zera albo
2) dołączenia się do już istniejącego mapowania pliku
   funkcja `OpenFileMapping()`
3) uzyskania adresu zamapowanego pliku
   funkcja `MapViewOfFile()`
4) wykorzystanie dostępu do pamięci
5) odłączenie mapowania
   funkcja `UnmapViewOfFile()`
6) zakończenie mapowania pliku
   funkcja `CloseHandle()` (shared memory control)

Wykonanie mapowania – funkcja `CreateFileMapping()`

```
#include <windows.h>

HANDLE CreateFileMapping(
  HANDLE                hFile,
  LPSECURITY_ATTRIBUTES lpFileMappingAttributes,
  DWORD                 flProtect,
  DWORD                 dwMaximumSizeHigh,
  DWORD                 dwMaximumSizeLow,
  LPCSTR                lpName
);
```

Wynik:

- uchwyt utworzonego mapowania albo `NULL` w przypadku błędu (GetLastError() zna szczegóły)

Wykonanie mapowania – funkcja `CreateFileMapping()`

```
#include <windows.h>

HANDLE CreateFileMapping(

  HANDLE                hFile
  :
);
```

Wejście:

- uchwyt mapowanego pliku albo stała symboliczna `INVALID_HANDLE_VALUE`, jeśli ma być wykonane mapowanie pozorne dla pamięci współdzielonej

Wykonanie mapowania – funkcja `CreateFileMapping()`

```
#include <windows.h>

HANDLE CreateFileMapping(
  :
  LPSECURITY_ATTRIBUTES lpFileMappingAttributes,
  :
);
```

Wejście:

- deskryptor zabezpieczeń dla tworzonego mapowania albo `NULL` dla deskryptora domyślnego

Wykonanie mapowania – funkcja `CreateFileMapping()`

```
#include <windows.h>

HANDLE CreateFileMapping(
  :
  DWORD                 flProtect,
  :
);
```

Wejście:

- ustawienia zabezpieczeń dla tworzonego mapowania; tylko te procesy uzyskają dostęp do mapowania, które wyspecyfikują ustawienia zgodne z użytymi w czasie tworzenia
- wybrane z możliwych wartości:
  `PAGE_READONLY` – tylko odczyt
  `PAGE_READWRITE` – zapis i odczyt

Wykonanie mapowania – funkcja `CreateFileMapping()`

```
#include <windows.h>

HANDLE CreateFileMapping(
  :
  DWORD                 dwMaximumSizeHigh,
  DWORD                 dwMaximumSizeLow,
  :
);
```

Wejście:

- maksymalny (w przypadku pamięci współdzielonej faktyczny) rozmiar mapowanej pamięci, rozbity na dwie dane typu `DWORD`:
- część starszą (`dwMaximumSizeHigh`)
- część młodszą (`dwMaximumSizeLow`)

Wykonanie mapowania – funkcja `CreateFileMapping()`

```
#include <windows.h>

HANDLE CreateFileMapping(
  :
  LPCSTR                lpName

);
```

Wejście:

- unikalna nazwa nadana mapowaniu (składnia identyczna jak w przypadku nazw plików w systemie Windows)

Dołączenie się do istniejącego mapowania – funkcja `OpenFileMapping()`

```
#include <windows.h>

HANDLE OpenFileMapping(
  DWORD  dwDesiredAccess,
  BOOL   bInheritHandle,
  LPCSTR lpName
);
```

Wynik:

- uchwyt dołączonego mapowania albo `NULL` w przypadku błędu (GetLastError() zna szczegóły)

Dołączenie się do istniejącego mapowania – funkcja `OpenFileMapping()`

```
#include <windows.h>

HANDLE OpenFileMapping(

  DWORD  dwDesiredAccess,
  :
);
```

Wejście:

- żądany dostęp do mapowania – możliwe wartości:
- `FILE_MAP_ALL_ACCESS` – zapis i odczyt
- `FILE_MAP_READ` – tylko odczyt
- `FILE_MAP_WRITE` – tylko zapis

Dołączenie się do istniejącego mapowania – funkcja `OpenFileMapping()`

```
#include <windows.h>

HANDLE OpenFileMapping(
  :
  BOOL   bInheritHandle,
  :
);
```

Wejście:

- jeśli ustawione na **TRUE**, to zezwala się na odziedziczenie tego uchwytu przez proces potomny

Dołączenie się do istniejącego mapowania – funkcja `OpenFileMapping()`

```
#include <windows.h>

HANDLE OpenFileMapping(
  :
  LPCSTR lpName
);
```

Wejście:

- unikalna nazwa mapowania (C-string)

Udostępnienie mapowania w przestrzeni adresowej procesu – funkcja `CreateFileMapping()`

```
#include <windows.h>

LPVOID MapViewOfFile(
  HANDLE hFileMappingObject,
  DWORD  dwDesiredAccess,
  DWORD  dwFileOffsetHigh,
  DWORD  dwFileOffsetLow,
  SIZE_T dwNumberOfBytesToMap
);
```

Wynik:

- lokalny adres przydzielonego bloku pamięci albo `NULL` w przypadku błędu (GetLastError() zna szczegóły)

Udostępnienie mapowania w przestrzeni adresowej procesu – funkcja `CreateFileMapping()`

```
#include <windows.h>

LPVOID MapViewOfFile(
  HANDLE hFileMappingObject,
  :
);
```

Wejście:

- uchwyt zwrócony z `CreateFileMapping()` albo `OpenFileMapping()`

Udostępnienie mapowania w przestrzeni adresowej procesu – funkcja `CreateFileMapping()`

```
#include <windows.h>

LPVOID MapViewOfFile(
  :
  DWORD  dwDesiredAccess,
  :
);
```

Wejście:

- żądany tryb dostępu do bloku pamięci – wybrane z możliwych wartości:
- `FILE_MAP_ALL_ACCESS` – zapis i odczyt
- `FILE_MAP_READ` – tylko odczyt
- `FILE_MAP_WRITE` – tylko zapis

Udostępnienie mapowania w przestrzeni adresowej procesu – funkcja `CreateFileMapping()`

```
#include <windows.h>

LPVOID MapViewOfFile(
  :
  DWORD  dwFileOffsetHigh,
  DWORD  dwFileOffsetLow,
  :
);
```

Wejście:

- rozbity na część młodszą (`dwFileOffsetLow`) i starszą (`dwFileOffsetHigh`) numer pierwszego dostępnego bajtu wewnątrz bloku

Udostępnienie mapowania w przestrzeni adresowej procesu – funkcja `CreateFileMapping()`

```
#include <windows.h>

LPVOID MapViewOfFile(
   :
   SIZE_T dwNumberOfBytesToMap
);
```

Wejście:

- liczba bajtów wewnątrz bloku, które proces chce użyć (w połączeniu z `FileOffset` nie może wyjść poza zakres określony w `CreateFileMapping()`)

Zakończenie mapowania w przestrzeni adresowej procesu – funkcja `UnmapViewOfFile()`

```
#include <windows.h>

BOOL UnmapViewOfFile(
  LPCVOID lpBaseAddress
)
```

Wynik:
- wartość różna od zera w przypadku powodzenia albo `0` w przypadku błędu (GetLastError() zna szczegóły)

Wejście:
- `lpBaseAddress` – adres bloku pamięci uzyskany z `MapViewOfFile()`

Program demonstracyjny

Założenia takie same, jak w przypadku Linuksa, z wyjątkiem:

- nadajnik startuje pierwszy i dostaje 2 argumenty: **nazwę** bloku pamięci współdzielonej i komunikat (string)

Nadajnik: plik `win_sender.c` (część 1/3)

```
#include <windows.h>
#include <stdio.h>

#define MEM_SIZE 1000

int main(int argc, char *argv[])
{
    if (argc != 3) {
        fprintf(stderr, "usage: %s name message\n", argv[0]);
        return 1;
    }

    HANDLE MapFile = CreateFileMapping(
            INVALID_HANDLE_VALUE,    // use paging file
            NULL,                    // default security
            PAGE_READWRITE,          // read/write access
            0,                       // maximum object size (high-order DWORD)
            MEM_SIZE,                // maximum object size (low-order DWORD)
            argv[1]);                // name of mapping object
```

Nadajnik: plik `win_sender.c` (część 2/3)

```
    :
    :
    if (MapFile == NULL)
    {
        fprintf(stderr, "Could not create file mapping object\n");
        return 2;
    }
    void *Buf = MapViewOfFile(
            MapFile,   // handle to map object
            FILE_MAP_ALL_ACCESS, // read/write permission
            0,
            0,
            MEM_SIZE);

    if (Buf == NULL)
    {
        fprintf(stderr, "Could not map view of file\n");
        CloseHandle(MapFile);
        return 3;
    }
```

Nadajnik: plik `win_sender.c` (część 3/3)

```
    :
    :
    strcpy(Buf, argv[2]);
    puts("Press <Enter> to continue...");
    getch();
    if(!UnmapViewOfFile(Buf)) {
        fprintf(stderr, "Could not unmap view of file\n");
        CloseHandle(MapFile);
        return 4;
    }
    CloseHandle(MapFile);
    return 0;
}
```

Nadajnik: plik `win_receiver.c` (część 1/2)

```
#include <windows.h>
#include <stdio.h>

#define MEM_SIZE 1000

int main(int argc, char *argv[])
{
    if (argc != 2) {
        fprintf(stderr, "usage: %s name\n", argv[0]);
        return 1;
    }

    HANDLE MapFile = OpenFileMapping(
            FILE_MAP_ALL_ACCESS,   // read/write access
            FALSE,                 // do not inherit the name
            argv[1]);              // name of mapping object

    if(MapFile == NULL) {
        fprintf(stderr, "Could not open file mapping object\n");
        return 1;
    }
```

Nadajnik: plik `win_receiver.c` (część 2/2)

```
    :
    void *Buf = MapViewOfFile(MapFile, // handle to map object
            FILE_MAP_ALL_ACCESS,  // read/write permission
            0,
            0,
            MEM_SIZE);
    if (Buf == NULL)
    {
        fprintf(stderr, "Could not map view of file\n");
        CloseHandle(MapFile);
        return 2;
    }
    printf("Message: '%s'\n", (char *)Buf);
    if(!UnmapViewOfFile(Buf)) {
        fprintf(stderr, "Could not unmap view of file\n");
        CloseHandle(MapFile);
        return 4;
    }
    CloseHandle(MapFile);
    return 0;
}
```