

# Práce s pamětí

Tématicky zaměřený vývoj aplikací v jazyce C  
skupina Systémové programování – Linux

Petr Velan

Fakulta informatiky  
Masarykova univerzita  
velan@ics.muni.cz

Brno, 26. listopadu 2014

## Hrátky s pamětí

Sdílená a mapovaná paměť

# Sdílená paměť

- jeden ze základních konceptů meziprocesové komunikace
- dva a více procesů sdílejí tutéž oblast v paměti
- velikost sdílené paměti je vždy násobkem velikosti stránky v systému (viz `man 2 getpagesize`)
- nejrychlejší způsob komunikace (nevstupuje se do kernel-space, nekopírují se žádná data)
- je ale třeba **důsledně** řídit přístup k paměti!

# Sdílená paměť

- jeden ze základních konceptů meziprocesové komunikace
- dva a více procesů sdílejí tutéž oblast v paměti
- velikost sdílené paměti je vždy násobkem velikosti stránky v systému (viz `man 2 getpagesize`)
- nejrychlejší způsob komunikace (nevstupuje se do kernel-space, nekopírují se žádná data)
- je ale třeba **důsledně** řídit přístup k paměti!

## Princip fungování

- 1 jeden z procesů alokuje segment paměti požadované velikosti pro konkrétní klíč
- 2 ostatní procesy se k alokovanému segmentu připojují – musí znát identifikátor segmentu, který získají buď předáním ID nebo pomocí klíče a `shmget()`
- 3 po skončení práce se všechny procesy odpojí od alokovaného segmentu
- 4 jeden z procesů paměť nakonec dealokuje

# Sdílená paměť – funkce

```
#include <sys/shm.h>
int shmget(key_t key, size_t size, int shmflg);
void *shmat(int shmid, const void *shmaddr, int shmflg);
int shmdt(const void *shmaddr);
int shmctl(int shmid, int cmd, struct shmids *buf);
```

**key** – identifikátor segmentu (`ftok()`), `IPC_PRIVATE` vytvoří nový segment

**shmflg**<sup>1</sup> – `IPC_CREAT`, `IPC_EXCL`, přístupová práva.

**shmflg**<sup>2</sup> – `SHM_RND`, `SHM_RDONLY`

**cmd** – `IPC_STAT`, `IPC_RMID` (`exit` nechává paměť alokovanou!), `IPC_SET`, ...

Ladění pomocí `ipcs -m`

---

<sup>1</sup>shmget

<sup>2</sup>shmat

# Sdílená paměť – ukázka

```
/* prepare key */
key_t shmkey = ftok("somefile", 1);

/* get a piece of memory */
int shmid = shmget(shmkey, getpagesize(), IPC_CREAT | S_IRUSR | S_IWUSR);

/* attache to the memory */
void *shared_memory = shmat (shmid, 0, 0);

/* do the job */

/* declare memory to be destroyed */
if (shmctl(shmid, IPC_STAT, &shm) == -1 ||
    shmctl(shmid, IPC_RMID, &shm) == -1) {
    perror("shmctl FAILURE");
}

/* detach the memory */
shmdt((void*)shared_memory);
```

### úkol

- Program uloží do sdílené paměti řetězec (parametr programu), který následně v cyklu vypisuje
- Při spuštění další instance programu, začnou všechny předchozí instance vypisovat nový řetězec

# Mapovaná paměť

- opět jeden z nástrojů meziprocesové komunikace
- dva a více procesů komunikují pomocí sdíleného souboru (paměťový segment se jménem)
- druhým (častějším) použitím je urychlení práce se souborem (**se souborem budeme pracovat jako s pamětí**)
- Linux vytvoří asociaci mezi souborem a pamětí a obsah souboru namapuje do stránek v paměti
- Rozpadá se konzistence mezi namapovaným souborem a skutečným souborem na disku
- Často se používá zařízení `/dev/zero` – paměť je pak inicializovaná na nuly



# Mapovaná paměť

```
#include <sys/mman.h>
void *mmap(void *addr, size_t length, int prot, int flags,
           int fd, off_t offset);
int msync(void *addr, size_t length, int flags);
int munmap(void *addr, size_t length);
```

**prot** – PROT\_READ, PROT\_WRITE, PROT\_EXEC, PROT\_NONE

**flags**<sup>3</sup> – MAP\_LOCKED, MAP\_PRIVATE, MAP\_SHARED, MAP\_ANONYMOUS

**flags**<sup>4</sup> – MS\_ASYNC, MS\_SYNC, MS\_INVALIDATE

---

<sup>3</sup>mmap

<sup>4</sup>msync

# Mapovaná paměť – ukázka

- opening file

```
if ((fd = open("./mapped_file", O_CREAT | O_RDWR,  
              S_IRUSR | S_IWUSR)) == -1) {  
    perror("open failed");  
}
```

- map the file into the memory

```
mm_addr = (char *)mmap(NULL, length, PROT_WRITE, MAP_SHARED, fd, 0);  
if (mm_addr == MAP_FAILED) {  
    perror("mmap failed");  
}
```

# Mapovaná paměť – ukázka

- opening file

```
if ((fd = open("./mapped_file", O_CREAT | O_RDWR,  
              S_IRUSR | S_IWUSR)) == -1) {  
    perror("open failed");  
}
```

- getting space in the file

```
lseek(fd, length+1, SEEK_SET);  
write(fd, "", 1);  
lseek(fd, 0, SEEK_SET);
```

- map the file into the memory

```
mm_addr = (char *)mmap(NULL, length, PROT_WRITE, MAP_SHARED, fd, 0);  
if (mm_addr == MAP_FAILED) {  
    perror("mmap failed");  
}
```

## úkol

- reimplementujte předchozí úkol pomocí mapované paměti

# Přístupová práva k paměti

- Změna práv nastavených pomocí parametru `prot` funkce `mmap`

```
<sys/mman.h>
```

```
int mprotect(const void *addr, size_t len, int prot);
```

# Uzamykání stránek ve fyzické paměti

```
#include <sys/mman.h>
int mlock(const void *addr, size_t len);
int munlock(const void *addr, size_t len);
int mlockall(int flags);
int munlockall(void);
```

- zabránění přesunu dat z RAM do SWAPu
- uzamykají se pouze celé stránky
- zamykat může pouze proces s právy superuživatele
- požadavek na rychlost
- požadavek na bezpečnost

## Vytváření dočasných souborů

# Vytváření dočasných souborů

- **bezpečné** vytváření dočasných souborů – řešení nebezpečí souběhu
- `template` je modifikován, takže nesmí jít o konstantní řetězec
- formát `template` je `'jmenoXXXXXX'`

```
#include <stdlib.h>
int mkstemp(char *template);
int mkostemp (char *template, int flags);
int mkstemps(char *template, int suffixlen);
int mkostemps(char *template, int suffixlen, int flags);
```



## Závěr

shrnutí, domácí úkoly a zdroje

# Domácí úkol

**Dokončete úlohy ze cvičení**

# Zdroje

- [beej.us/guide/bgipc/output/html/multipage/mmap.html](http://beej.us/guide/bgipc/output/html/multipage/mmap.html)
- [www.cs.cf.ac.uk/Dave/C/node27.html](http://www.cs.cf.ac.uk/Dave/C/node27.html)