Shared Memory (2/2)

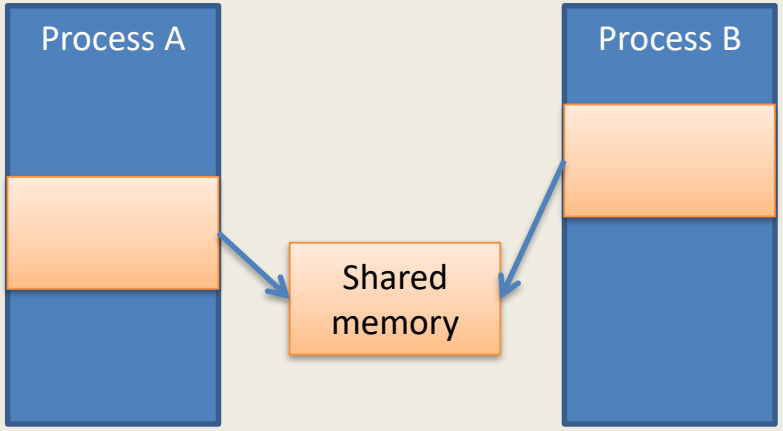
Sistemas de Computadores

2017/2018

Luís Miguel Pinho / Luis Ferreira / Luís Nogueira / Nuno Pereira

Previous class: POSIX shared mem in Linux

- Shared memory
  - Allow two or more processes to access the same memory area



1. Create
   - **`shm_open()`, `ftruncate()`, `mmap()`**
2. Use
   - With a pointer as dynamic memory
3. This class: remove
   - **`munmap()`, `close()`, `shm_unlink()`**

Remove

- **`munmap()`**
  - Disconnects the shared memory area from the process address space (inverse of **`mmap()`**)
- **`close()`**
  - Closes the file descritor returned by **`shm_open()`**
- **`shm_unlink()`**
  - Removes the memory área from the file system. Marks it to be deleted, as soon as all processes using it close it

munmap()

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

int munmap(void *addr, size_t length);
```

- Disconnects the shared memory area from the process address space
- Returns 0 if successful, or -1 in case of error (*errno* is set with the error)

munmap()

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

int munmap(void *addr, size_t length);
```

- **addr**: Pointer returned by mmap()
- **length**: Size of the mapped memory area
  - ≤ size defined in **ftruncate()**
  - Usually rounded to memory page multiple

close()

```
#include <unistd.h>

int close(int fd);
```

- Closes the file descriptor
- **fd**: the file descriptor returned by **shm_open()**
- Returns 0 if successful, or -1 in case of error (*errno* is set with the error)

shm_unlink()

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

int shm_unlink(const char *name);
```

- Removes memory area from file system
  - May not be immediate if any process still has it open
  - The area is marked to be removed as soon as all processes close it
- **name**: shared memory name
- Returns 0 if successful, or -1 in case of error (*errno* is set with the error)

Close shared memory

- Example:
  - Process close its descriptor as soon as not needing it
  - Another process can still open and use it

Close shared memory

```
1.   int fd, r;
2.   void *addr; /* pointer to the shared memory */
3.   fd = shm_open(”/shmtest”, ...);
4.   ftruncate (fd, 100);
5.   addr = mmap(NULL, 100, ...);

6.   ... /* use – read/write */

7.   r = munmap(addr, 100); /* disconnects */
8.   if (r < 0) exit(1); /* Check error */

9.   r = close(fd); /* closes */
10.  if (r < 0) exit(1); /* Check error*/
11.  exit(0); /* all descriptors are closed... */
```

Remove shared memory

- Next example:
  - Process removes the shared memory
  - Other processes will have to create it again

Remove shared memory

```
1.   int fd, r;
2.   void *addr; /* pointer to the shared memory */
3.   fd = shm_open(”/shmtest”, ...);
4.   ftruncate (fd, 100);
5.   addr = mmap(NULL, 100, ...);

6.   ... /* use – read/write */

7.   r = munmap(addr, 100); /* disconnects */
8.   if (r < 0) exit(1); /* Check error */

9.   r = shm_unlink(”/shmtest”); /* removes */
10.  if (r < 0) exit(1); /* Check error */
11.  exit(0);
```

Exercise TP6.1

- Create two unrelated processes: writer and reader.
  - Writer: asks user the name of the place and a set of 10 temperature readings of the last 24 hours, putting all this information in the shared memory
  - Leitor: Calculates and prints average
- The two processes may be executed simultaneously
- Reader removes area from the system

Exercise TP6.2

- Implement a program that creats a shared memory área for two arrays of integer: **`v[1000]`** e **`max[10]`**
  - Initialize **`v[]`** with random numbers
- Create 10 children; Each child should search the max in 1/10 of **`v[]`** and put it in **`max[i]`** (**`i`** is the child index)
- Parents waits all children to terminate and calculates the maximum in **`max[]`**
- All processes should disconnect and close
- Parent should remove shared memory area before terminating

Obtain information about shared memory

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

int fstat(int fd, struct stat *buf);
```

- Args:
  - **`fd:`** file descritor returned **`shm_open()`**
  - **`buf`**: **`stat`** structure where information will be placed
- The structure will have info on the area, such as size, permissions, owner, etc.

Obtain information about shared memory

```
struct stat {
    dev_t     st_dev;     /* ID of device containing file */
    ino_t     st_ino;     /* inode number */
    mode_t    st_mode;    /* protection */
    nlink_t   st_nlink;   /* number of hard links */
    uid_t     st_uid;     /* user ID of owner */
    gid_t     st_gid;     /* group ID of owner */
    dev_t     st_rdev;    /* device ID (if special file) */
    off_t     st_size;    /* total size, in bytes */
    blksize_t st_blksize; /* blocksize for file system I/O */
    blkcnt_t  st_blocks;  /* number of 512B blocks allocated */
    time_t    st_atime;   /* time of last access */
    time_t    st_mtime;   /* time of last modification */
    time_t    st_ctime;   /* time of last status change */
};
```

Obtain information about shared memory

```
int main(void){
    int fd; /* File descriptor */
    int ret;
    struct stat shm_stat;

    fd = shm_open("/shmtest", O_CREAT|O_EXCL|O_RDWR, 0600);
    if (fd < 0) exit(1);

    ftruncate (fd, 100);
    addr = mmap(NULL,100,PROT_READ|PROT_WRITE,MAP_SHARED,fd,0);
    if (addr == NULL) exit(1);

    if(fstat(fd, &shm_stat) < 0) exit (1);

    printf("mode = %d\n", shm_stat.st_mode);
    printf("size = %d\n", shm_stat.st_size);
    ...
}
```

/dev/shm

- In Linux, the temporary file system (**tmpfs**) used for POSIX shared memory is mapped in **/dev/shm**
  - POSIX semaphores (next classes) are also mapped here

- Shared memory objects can be listed as files (with **ls**)

```
% ls /dev/shm
shmtest   shmtp61
```

/dev/shm

- And removed with **rm**

```
% rm /dev/shm/shmtest
```