# Operating Systems

# Grado en Informática 2023/2024

## Lab Assignment 2: Memory

We continue to code the shell we started in the first lab assignment. We'll add the following commands. Check the supplied shell to check the workings and sintax for the commands. You can use the help command, "command -?" or "command -help" to get help):

| | |
|---|---|
| **malloc** | allocates (or deallocates) a block malloc memory. Updates the list of memory blocks |
| **shared** | allocates (or deallocates) a block shared memory. Updates the list of memory blocks |
| **mmap** | maps (or unmaps) a file in memory. Updates the list of memory blocks |
| **read** | reads a file into memory |
| **write** | writes to a file the contents at a memory address |
| **memdump** | dumps the contents of memory to the screen |
| **memfill** | fills the memory with one character |
| **mem** | shows information on the memory of the process |
| **recurse** | executes a recursive function |

IMPORTANT:

We have to implement (list implementation free) a list of memory blocks. For each block we must store

- Its memory address
- Its size
- Time of allocation
- Type of allocation (malloc memory, shared memory, mapped file)
- Other info: key for shared memory blocks, name of file and file descriptor for mapped files.

The shell commands ***malloc, shared*** and ***mmap*** allocate and deallocate memory blocks and add (or remove) them from the list. Each element on the list has info of a memory block we created with the shell commands ***malloc, shared*** and ***mmap.*** The information about that block is the one previously stated. We'll deal with three types of memory blocks.

- **malloc memory.** this is the most common memory we use, we allocate it with the library function *malloc* and deallocate it with the library function *free*

- **shared memory.** this is memory that can be shared among several processes. The memory is identified by a number (called key) so that two processes using the same key get to the same block of memory. We use the system call shmget to obtain the memory and shmat and shmdt to place it in (or detatch it from) the process address space. shmget needs the key, the size and the flags. We'll use *flags=IPC_CREAT | IPC_EXCL| permisions* to create a new one (gives error if it already exists) and *flags=permisions* to use an already created one. To delete a key we'll use

the shared **-delkey** command (this commands dealocates nothing, just deletes the key). Status of the shared memory in the system can be checked via the command line with the *ipcs* command. An additional C file (ayudaP2.c) is provided with some useful functions

- **mapped files.** We can also map files in memory so that they apperar in the address space of a process. System calls mmap and munmap do the trick. Again, the additional C file (ayudaP2.c) is provided with some useful functions

The contents of our list must be compatible with what the system shows with the *pmap* command (*procstat vm*, *vmmap* ...depending on the platform)

The recursive function has a static and a dynamic array for the same size (2048 bytes) and prints their addresses together with the parameter address and value

**Although NO RUNTIME ERROR WILL BE ALLOWED** (segmentation, bus error . . . ) and programs with runtime errors will yield no score, this program can legitimately produce segmentation fault errors in scenarios shuch as these:

- **memdump** or **memfill** try to access an invalid address supplied through the command line
- **memfill** or **read** corrupt the user stack or the heap

REMEMBER:

- Information on the system calls and library functions needed to code these programs is available through man: (shmget, shmat, malloc, free, mmap, munmap, shmctl, open, read, write, close . . .)
- A reference shell is provided (for various platforms) for students to check how the shell should perform. Ths program should be checked to find out the syntax and behaviour of the various commands. **PLEASE DOWNLOAD THE LATEST VERSION**
- The program should compile cleanly (produce no warnings even when compiling with gcc -Wall)
- These programs can have no memory leaks (you can use valgrind to check)
- When the program cannot perform its task (for whatever reason, for example, lack of privileges) it should inform the user (See errors section)
- All input and output is done through the standard input and output
- Errors should be trated as in the previous lab assignment

WORK SUBMISSION

- Work must be done in pairs.
- Moodle will be used to submit the source code: a zipfile containing a directory named P2 where all the source files of the lab assignment reside
- The name of the main program will be p2.c, Program must be able to be compiled with gcc p2.c, Optionally a Makefile can be supplied so that all of the source code can be compiled with just make. Should that be the case, the compiled program should be called p2
- ONLY ONE OF THE MEMBERS OF THE GROUP will submit the source code. The names and logins of all the members of the group should appear in the source code of the main programs (at the top of the file)
- Works submited not conforming to these rules will be disregarded.
- DEADLINE: 23:00, Friday November the 24th, 2023