

Operating Systems

Grado en Informática 2022/2023

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 syntax for the commands. You can use the help command, "command -?" or "command -help" to get help):

allocate	allocates a block of memory and adds it to the list of memory blocs
deallocate	deallocates a block of memory a removes it from the list of memory blocks
i-o	performs input output from disk to memory and viceversa
memdump	dumps the contents of memory to the screen
memfill	fills the memory with one character
memory	shows information on the memory of the process
recurse	executes a recursive function

IMPORTANT:

We have to implement (same list type as we used before) 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 **allocate** and **deallocate**, allocate and deallocate memory blocks and add (or remove) them from the list. We'll use **allocate -malloc** and **deallocate -malloc** to deal with malloc memory, **allocate -shared** and **deallocate -shared** to deal with shared memory and so on. Our list must be of the same type as the one we have used before and we'll store in it information about three different kinds 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 detach it from) the process address space. *shmget* needs the key, the size and the flags. We'll use *flags=IPC_CREAT | IPC_EXCL | permissions* to create a new one (gives error if it already exists) and *flags=permissions* to use an already created one. To delete a key we'll use **deallocate -delkey** command. Status of the shared memory in the system can be checked via

the command line with the *ipc* 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 appear 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 such as these:

- *memdump* or *memfill* try to access an invalid address supplied through the command line
- *memfill* or *io-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. This 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 treated 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 submitted not conforming to these rules will be disregarded.
- **DEADLINE: 23:00, Friday November the 18th, 2022**