Operating Systems

Grado en Informática 2024/2025

Lab Assignment 2: Memory

We continue to code the shell we started in the first lab assignment. Now the shell will have the ability to allocate or deallocate memory blocks via malloc, mmmap or shared memory. The shell will keep a list ot the memory blocks allocated (only the ones allocated with the command allocate, not the ones it needs to allocate for its normal working).

Command	Param	Param	
allocate	**-malloc**	**n**	Allocates a block of malloc memory of size n bytes. Updates the list of memory blocks
allocate	**-mmap**	**file perm**	Maps a fille to memory with permissions perm. Updates the list of memory blocks
allocate	**-create shared**	**cl n**	Creates a block of shared memory with key cl and size n and attaches it to the process address space.Updates the list of memory blocks
allocate	**-shared**	**cl**	Attaches block a shared memory to the process address space (the block must be already created but not necessarily attached to the process space). Updates the list of memory blocks
deallocate	**-malloc**	**n**	Deallocates a block of malloc memory of size n (provided it has been previously allocated with allocate). Updates the list of memory blocks
deallocate	**-mmap**	**file**	Unmaps a file from memory (provided it has been previously mapped).Updates the list of memory blocks
deallocate	**-shared**	**cl**	Detaches a block of shared memory of key cl (provided it has been previously allocated).Updates the list of memory blocks
deallocate	**-delkey**	**cl**	Removes de memory block of key cl from the system. **IT DOES NOT DETACH THE SHARED MEMORY WITH THAT KEY SHOULD IT BE ATTACHED**
deallocate	**addr**		Deallocates the block with addres addr. (if it is a malloc block, it frees it; if it is a sharedcmemory block, it detaches it….). Updates the list of memory blocks
memfill	**addr**	**ch cont**	fills the memory with character ch, starting at address addr, for cont bytes
memdump	**addr**	**cont**	dumps the contents of cont bytes of memory at address addr to the screen. Dumps hex codes, and for printable characters the associated charrcter
memory	**-funcs**		Prints the addresses of 3 program functions and 3 library functions
memory	**-vars**		Prints the addresses of 3 external, 3 external initialized, 3 static, 3 static initialized and 3 automatic variables

memory	**-blocks**		Prints the list of allocated blocks
memory	**-all**		Prints all of the above (-funcs, -vars and -blocks)
memory	**-pmap**		Shows the output of the command pmap for the shell process (vmmap en macos)
readfile	**file**	**addr cont**	Reads cont bytes of a file into memory address addr
writefile	**file**	**addr cont**	writes to a file cont bytes starting at memory address addr
read	**df**	**addr cont**	The same as readfile but we use a (already opened) file descriptor
write	**df**	**addr cont**	The same as writefile but we use a (already opened) file descriptor
recurse	**n**		executes the recursive function n times. This funcion has an automatic array of size 2048. a static array of size 2048 and prints the addresses of both arrays and its parameter (as well as the number o recursion) before calling itself

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 ***allocate*** *and* ***deallocate*** 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 command ***allocate*** 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 **deallocate -delkey** command (this command deallocates 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.txt) 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.txt) 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

LEGITIMATE RUNTIME ERRORS

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 , readfile** 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 *version* command, on newer reference shell, shows which version you are using)
- 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 and placed on the same directory as the sources (no build directories or similar)
- 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 22th, 2023