Operating Systems

Grado en Informática. Course 2021-2022

Lab assignment 2: Memory

CONTINUE to code the shell started in previous lab assignments. The goal of this assignment is to understand how the memory of a proccess is organized. At this stage of developement the shell will be able to map files into memory and to allocate memory to itself (either shared or private). In next assignments we will deal with creating proccesses and executing programs.

The shell will keep track (using a list) of all the memory blocks it allocates and deallocates using the commands *malloc, mmap* and *shared.* For each block of memory allocated with those commands the shell must store its memory address (the address of the block), its size (the size of the block), the time at which it was allocated, the type of allocation (malloc, shared memory, mapped file), and other pieces of information depending on the type of allocation (name of file for mapped files, key for shared memory ...)

Values in that list must be coherent with what pmap shows for the shell proccess

In addition to the commands done in previous lab assignments, the shell has to implement the following commands

malloc [**-free**] [**tam**] The shell allocates *tam* bytes using *malloc* and shows the memory address returned by *malloc*. This address, together with *tam* and the time of the allocation, must be kept in the aforementioned list. If *tam* is not specified the command will show the list of addresses allocated with the `malloc` command. **malloc() requires an argument of size_t** Example:

```
-> malloc 100000000
allocated 100000000 at 0x7f6649f24010
-> malloc
0x7f6649f24010: size:100000000. malloc  Mon Oct 26 20:07:05 2018
-> malloc  500000000
allocated 500000000 at 0x7f662c24d010
-> malloc
0x7f6649f24010: size:100000000. malloc  Mon Oct 26 20:07:05 2018
```

```
0x7f662c24d010: size:500000000. malloc  Mon Oct 26 20:08:17 2018
->
```

If used as **malloc -free** [**tam**] The shell deallocates one of the blocks of size *tam* that has been allocated with the command *malloc*. If no such block exists or if *tam* is not specified, the command will show the list of addresses allocated with the `malloc` command. Should there be more than one block of size *tam* it deallocates ONLY one of them (any). Example:

```
-> malloc -free  100000000
deallocated 100000000 at 0x7f6649f24010
-> malloc
0x7f662c24d010: size:500000000. malloc  Mon Oct 26 20:08:17 2015
->
```

mmap [**-free**] **fich** [**perm**] Maps in memory the file *fich* (all of its length starting at offset 0) and shows the memory address where the file has been mapped. *perm* represents the mapping permissions (*rwx* format, without spaces). The address of the mapping, together with the size, the name of the file, the file descriptor, and the time of the mapping will be stored in the aforementioned list. If *fich* is not specified, the command will show the list of addresses allocated with the `mmap` command. Example

```
-> mmap Shell.c rw
file Shell.c mapped at 0x7f6650436000
-> mmap Shell.c rwx
file Shell.c mapped at 0x7f6650424000
-> mmap
0x7f6650436000: size:71806. mmap Shell.c (fd:3) Mon Oct 26 20:10:07 2018
0x7f6650424000: size:71806. mmap Shell.c (fd:5) Mon Oct 26 20:10:17 2018
```

mmap -free fich Unmaps and closes the file *fich* and removes the address where it was mapped from the list. If *fich* has been mapped several times, only one of the mappings will be undone. If the file *fich* is not mapped by the process or if *fich* is not specified, the command will show the list of addresses (and size, and time ...) allocated with the `mmap` command.

shared [**-free|-create|-delkey**] **cl** [**tam**] Gets shared memory of key *cl*, maps it in the proccess address space and shows the memory address where

the shared memory has been mapped. That address, together with the key, the size of the shared memory block and the time of the mapping, will be stored in the aforementioned list. *cl* **IS THE KEY**, (***ftok*** **should not be used**). Parameter *tam* is only used when creating a new block of shared memory. if *-create* is not given, it is assumed that key *cl* is in use in the system so a new block of shared memory SHOULD NO BE CREATED, and an error must be reported if a block of key *cl* does not exist . If no *cl* is specified, the command will show the list of addresses (and size, and time . . .) allocated with the `shared` command. **(shared memory blocks can be of size_t size)**. **THE MEMORY SHOULD BE OBTAINED WITH shmget, NOT SEARCHED FOR IN THE LIST**
shared -create cl tam Creates a shared memory block of key *cl*, and size *tam*, maps it in the proccess address space and shows the memory address where the shared memory has been mapped. That address, together with the key, the size of the shared memory block and the time of the mapping, will be stored in the aforementioned list. *cl* **IS THE KEY**, (***ftok*** **should not be used**). It is assumed that key *cl* is not in use in the system so a new block of shared memory MUST BE CREATED, and an error must be reported if a block of key *cl* exists already. If *cl* is not specified, the command will show the list of addresses (and size, and time . . .) allocated with the `shared` command.
shared -free cl Detaches the shared memory block with key *cl* from the process' address space ad eliminates its address from the list. If shared memory block with key *cl* has been attached several times, ONLY one of them is detached. *cl* **IS THE KEY**, *ftok* should not be used. If *cl* is not specified, the command will show the list of addresses (and size, and time . . .) allocated with the `shared` command
shared -delkey cl Removes the shared memory region of key *cl*. **nothing gets unmapped**: this is just a call to *shmctl(id, IPC_RMID. . .)*

```
-> shared 15
Cannot allocate: No such file or directory
-> shared -create 15 300000000
Allocated shared memory (key 15) at 0x7f661a432000
-> shared -create 15 200000000
Cannot allocate: File exists
-> shared 15
```

```
Allocated shared memory (key 15) at 0x7f6608617000
-> shared
0x7f661a432000: size:300000000. shared memory (key 15) Mon Oct 26 20:12:47 20
0x7f6608617000: size:300000000. shared memory (key 15) Mon Oct 26 20:12:54 20
-> shared -free 15
Shared memory block at 0x7f661a432000 (key 15) has been dealocated
-> shared
0x7f6608617000: size:300000000. shared memory (key 15) Mon Oct 26 20:12:54 20
-> shared -delkey 15
Key 15 removed from the system
-> shared -delkey 15
Cannot remove key 15: No such file or directory
-> shared 15
Cannot allocate: No such file or directory
-> shared
0x7f6608617000: size:300000000. shared memory (key 15) Mon Oct 26 20:12:54 20
```

dealloc [-malloc|-shared|-mmap] . . . deallocates one of the memory blocks allocated with the command *malloc*, *mmap* or *shared* and removes it from the list. If no arguments are given, it prints a list of the allocated memory blocks (that is, prints the list)

```
-> dealloc
0x7f6649f24010: size:100000000. malloc  Mon Oct 26 20:07:05 2018
0x7f662c24d010: size:500000000. malloc  Mon Oct 26 20:08:17 2018
0x7f6650436000: size:71806. mmap Shell.c (fd:3) Mon Oct 26 20:10:07 2018
0x7f6650424000: size:71806. mmap Shell.c (fd:5) Mon Oct 26 20:10:17 2018
0x7f6608617000: size:300000000. shared memory (key 15) Mon Oct 26 20:12:54 20
```

dealloc -malloc size Does exactly the same as *malloc -free size*
dealloc -shared cl Does exactly the same as *shared -free cl*
dealloc -mmap file Does exactly the same as *mmap -free file*
dealloc addr Deallocates *addr* (it searchs in the list how it was allocated, and proceeds accordingly) and removes it from the list. If *addr* is not in the list or if *addr* is not supplied the command will show all the addresses (and size, and time . . .) allocated with the **malloc**, **mmap**, **shared** commands. This does the same (albeit with a different parameter) as *malloc -free*, *shared -free* or *mmap -free* deppending on *addr*. Example

```
-> dealloc
0x7f6649f24010: size:100000000. malloc  Mon Oct 26 20:07:05 2018
0x7f662c24d010: size:500000000. malloc  Mon Oct 26 20:08:17 2018
0x7f6650436000: size:71806. mmap Shell.c (fd:3) Mon Oct 26 20:10:07 2018
0x7f6650424000: size:71806. mmap Shell.c (fd:5) Mon Oct 26 20:10:17 2018
0x7f6608617000: size:300000000. shared memory (key 15) Mon Oct 26 20:12:54 20
-> dealloc -malloc 100000000
block at address 0x7f6649f24010 deallocated (malloc)
-> dealloc -shared 15
block at address 0x7f6608617000 deallocated (shared)
-> dealloc -mmap Shell.c
block at address 0x7f6650436000 deallocated (mmap)
-> dealloc 0x7f662c24d010
block at address 0x7f662c24d010 deallocated (malloc)
-> dealloc 0x7f662c24d010
0x7f6650424000: size:71806. mmap Shell.c (fd:5) Mon Oct 26 20:10:17 2018
```

memoria [**-blocks**] [**-vars**] [**-funcs**] [**-all**] [**-pmap**] Shows addresses inside the process memory space. If no arguments are given, is equivalent to *-all*

memoria -blocks shows the list of addresses (and size, and time ...) allocated with the *malloc*, *shared* and *mmap*

memoria -vars Prints the memory addresses of nine variables of the shel: three extern (global) initializad variables, three static initialized and three automatic (local) initialized variables

memoria -funcs Prints the memory addresses of three program functions of the shell and three C library functions used in the shell program.

memoria -all does the equivalent to *-blocks*, *-vars* and *-funcs* together..

memoria -pmap Calls the program *pmap* for the shell process (sample ode is given in the clues section)

volcarmem ***addr [cont]*** Shows the contents of *cont* bytes starting at memory address *addr*. If *cont* is not specified, it shows 25 bytes. For each byte it prints (at different lines) its hex value and its associated char (a blank if it is a non-printable character). It prints 25 bytes per line. **addr SHOULD NOT BE CHECKED FOR VALIDITY**, so, this command could produce segmentation fault should *addr* were not valid.

```
-> volcarmem 0xb8019000 300
```

```
 #  i  n  c  l  u  d  e     <  u  n  i  s  t  d  .  h  >     #  i  n  c  l
23 69 6E 63 6C 75 64 65 20 3C 75 6E 69 73 74 64 2E 68 3E 0A 23 69 6E 63 6C
 u  d  e     <  s  t  d  i  o  .  h  >     #  i  n  c  l  u  d  e     <  s
75 64 65 20 3C 73 74 64 69 6F 2E 68 3E 0A 23 69 6E 63 6C 75 64 65 20 3C 73
 t  r  i  n  g  .  h  >     #  i  n  c  l  u  d  e     <  s  t  d  l  i  b
74 72 69 6E 67 2E 68 3E 0A 23 69 6E 63 6C 75 64 65 20 3C 73 74 64 6C 69 62
 .  h  >     #  i  n  c  l  u  d  e     <  s  y  s  /  t  y  p  e  s  .  h
2E 68 3E 0A 23 69 6E 63 6C 75 64 65 20 3C 73 79 73 2F 74 79 70 65 73 2E 68
 >     #  i  n  c  l  u  d  e     <  s  y  s  /  s  t  a  t  .  h  >     #
3E 0A 23 69 6E 63 6C 75 64 65 20 3C 73 79 73 2F 73 74 61 74 2E 68 3E 0A 23
```

llenarmem ***addr [cont] [byte]*** Fills *cont* bytes of memory starting at address *addr* with the value *'byte'*. If *'byte'* is not specified, the value of 65 (0x42 or capital A) is assumed, and if *cont* is not specified, we'll use a default value of 128. **addr SHOULD NOT BE CHECKED FOR VALIDITY**, so, this command could produce segmentation fault should *addr* were not valid.
Example:

```
-> llenarmem 0xb8019000 1500 0x42
```

recursiva **n**. Calls a recursive function passing the integer n *n* as its parameter. This recursive function receives the number of times it has to call itself. This function has two variables: an automatic array of 4096 bytes and a static array of the same size. It does the following

- prints the value of the received parameter as well as its memory address.
- prints the address of the static array.
- prints the address of the automatic array.
- decrements n (its parameter) and if n is greater than 0 it calls itself.

A possible coding for the function that does the acual recursion:

```
void doRecursiva (int n)
{
  char automatico[TAMANO];
  static char estatico[TAMANO];

  printf ("parametro n:%d en %p\n",n,&n);
  printf ("array estatico en:%p \n",estatico);
  printf ("array automatico en %p\n",automatico);

  n--;
  if (n>0)
    recursiva(n);
}
```

e-s read fich addr cont Reads (using ONE *read* system call) *cont* bytes from file *fich* into memory address *addr*. If *cont* is not specified ALL of fich is read onto memory address *addr*. Depending on the value of *addr* a segmentation fault could be produced.

e-s write [-o] fich addr cont Writes (using ONE *write* system call) *cont* bytes from memory address *addr* into file *fich*. If file *fich* does not exist it gets created; if it already exists it is not overwritten unless "-o" (overwrite) is specified.

NOTES ON LIST IMPLEMETATION

- the implementations of list should consist of the data types and the access funtions. All access to the list should be done used the aforementioned access functions.
- students can choose from one of these three list implementations
  1) **linked list:** The list is composed of dynamically allocated nodes. Each node has some item of information and a pointer to the following node. The list itself is a pointer to the first node, when the list is empty this pointer is NULL, so creating the list is asigning NULL to the list pointer, thus the functions `CreateList,` `InsertElement` and `RemoveElement` must receive the list by reference as they may have (case of inserting or removing the first element) to modify the list. There can also be used a double linked version of this list (each node has two pointers)
  2) **linked list with head node:** Similar to the linked list except that the list itself is a pointer to an *empty* (with no information) first node. Creating the list is allocating this first element (head node). `CreateList` must receive the list by reference whereas `InsertElement` and `RemoveElement` can receive the list by value.

There can also be used a double linked version of this list (each node has two pointers)

3) **array of pointers;** The list is an array (statically allocated) of pointers. Each pointer points to one element in the list which is allocated dynamically. For the purpose of this lab assigment we can assume this statically allocated array dimension to be 4096, which should be declared a named constant, and thus easily modifiable. To implement the list with this array we can use either a NULL terminated array or we can use aditional integers. We could also make the list completely dynamic by using a dynamically allocated pointer instead the fixed size array of pointers

- This program should compile cleanly (produce no warnings even when compiling with `gcc -Wall`)
- **NO RUNTIME ERROR WILL BE ALLOWED (segmentation, bus error . . .)**, unless where explicitly spcified. Programs with runtime errors will yield no score.
- This program can have no memory leaks (memory blocks allocated with the **malloc** command are not taken into acount)
- When the program cannot perform its task (for whatever reason, for example, lack of privileges) it should inform the user
- All input and output is done through the standard input and output

Information on the system calls and library functions needed to code this program is available through `man`: (*open, read, write, close, shmget, shmat, shamdt, shmctl, mmap, munmap, malloc, free. . .)*.

WORK SUBMISSION

- Work must be done in pairs.
- Moodle will be used to submit the source code (you'll be informed of the exact procedure at a later time)
- 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`
- **ONLY ONE OF THE MEMBERS OF THE GROUP** will sub-

mit the source code. The names and logins of all the members of the group should be 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 19nd, 2021

ASSESSMENT: For each pair, it will be done in its corresponding group, during the lab classes

CLUES

The following functions could help with the implementation of *shared*, *mmap*, *e-s read* and *e-s write*. It is asumed that

- The functioms to perform the tasks *'Guardar En Direcciones de Memoria Shared'. 'Listar Direcciones de Memoria Shared'*... must be implemented as part of the assignment
- *SharedCreate* and *Mmap* receive as parameter a null terminated array of pointers, more info is given in the source code

```
 /*********************************************************/
 /*********************************************************/
void * ObtenerMemoriaShmget (key_t clave, size_t tam)
{               /*Obtienen un puntero a una zaona de memoria compartida*/
                /*si tam >0 intenta crearla y si tam==0 asume que existe*/
 void * p;
 int aux,id,flags=0777;
 struct shmid_ds s;

 if (tam)  /*si tam no es 0 la crea en modo exclusivo
           esta funcion vale para shared y shared -create*/
   flags=flags | IPC_CREAT | IPC_EXCL;
           /*si tam es 0 intenta acceder a una ya creada*/
 if (clave==IPC_PRIVATE)   /*no nos vale*/
       {errno=EINVAL; return NULL;}

 if ((id=shmget(clave, tam, flags))==-1)
```

```
        return (NULL);

 if ((p=shmat(id,NULL,0))==(void*) -1){
        aux=errno;   /*si se ha creado y no se puede mapear*/
        if (tam)     /*se borra */
                shmctl(id,IPC_RMID,NULL);
        errno=aux;
        return (NULL);
        }
 shmctl (id,IPC_STAT,&s);

/* Guardar En Direcciones de Memoria Shared (p, s.shm_segsz, clave.....);*/
 return (p);
}

void SharedCreate (char *arg[]) /*arg[0] is the key
                                      and arg[1] is the size*/
{
   key_t k;
   size_t tam=0;
   void *p;

   if (arg[0]==NULL || arg[1]==NULL)
       {/*Listar Direcciones de Memoria Shared */;*/ return;}

   k=(key_t) atoi(arg[0]);

   if (arg[1]!=NULL)
        tam=(size_t) atoll(arg[1]);

   if ((p=ObtenerMemoriaShmget(k,tam))==NULL)
        perror ("Imposible obtener memoria shmget");
   else
        printf ("Memoria de shmget de clave %d asignada en %p\n",k,p);
}
```

```
/**********************************************************************/
/**********************************************************************/

void * MmapFichero (char * fichero, int protection)
{
  int df, map=MAP_PRIVATE,modo=O_RDONLY;
  struct stat s;
  void *p;

  if (protection&PROT_WRITE)  modo=O_RDWR;

  if (stat(fichero,&s)==-1 || (df=open(fichero, modo))==-1)
      return NULL;

  if ((p=mmap (NULL,s.st_size, protection,map,df,0))==MAP_FAILED)
      return NULL;

  /*Guardar Direccion de Mmap (p, s.st_size,fichero,df......);*/
  return p;
}

void Mmap (char *arg[]) /*arg[0] is the file name
                                  and arg[1] is the permissions*/
{
   char *perm;
   void *p;
   int protection=0;

   if (arg[0]==NULL)
       {/*Listar Direcciones de Memoria mmap;*/ return;}


  if ((perm=arg[1])!=NULL && strlen(perm)<4) {
       if (strchr(perm,'r')!=NULL) protection|=PROT_READ;
       if (strchr(perm,'w')!=NULL) protection|=PROT_WRITE;
       if (strchr(perm,'x')!=NULL) protection|=PROT_EXEC;
   }
   if ((p=MmapFichero(arg[0],protection))==NULL)
```

```
        perror ("Imposible mapear fichero");
   else
        printf ("fichero %s mapeado en %p\n", arg[0], p);

}

#define LEERCOMPLETO ((ssize_t)-1)
ssize_t LeerFichero (char *fich, void *p, ssize_t n)
{                         /* le n bytes del fichero fich en p */
 ssize_t  nleidos,tam=n;  /*si n==-1 lee el fichero completo*/
 int df, aux;
 struct stat s;

 if (stat (fich,&s)==-1 || (df=open(fich,O_RDONLY))==-1)
        return ((ssize_t)-1);

 if (n==LEERCOMPLETO)
        tam=(ssize_t) s.st_size;

 if ((nleidos=read(df,p, tam))==-1){
        aux=errno;
        close(df);
        errno=aux;
        return ((ssize_t)-1);
        }
 close (df);

 return (nleidos);
}

/************************************************************************/
/************************************************************************/
void SharedDelkey (char *args[]) /*arg[0] points to a str containing the key*/
{
   key_t clave;
   int id;
   char *key=args[0];
   if (key==NULL || (clave=(key_t) strtoul(key,NULL,10))==IPC_PRIVATE){
       printf ("   shared -delkey clave_valida\n");
       return;
   }
   if ((id=shmget(clave,0,0666))==-1){
      perror ("shmget: imposible obtener memoria compartida");
      return;
   }
   if (shmctl(id,IPC_RMID,NULL)==-1)
      perror ("shmctl: imposible eliminar memoria compartida\n");
}

void dopmap (void) /*no arguments necessary*/
 { pid_t pid;         /*ejecuta el comando externo pmap para */
   char elpid[32];    /*pasandole el pid del proceso actual */
   char *argv[3]={"pmap",elpid,NULL};

   sprintf (elpid,"%d", (int) getpid());
   if ((pid=fork())==-1){
      perror ("Imposible crear proceso");
      return;
      }
   if (pid==0){
      if (execvp(argv[0],argv)==-1)
        perror("cannot execute pmap");
      exit(1);
  }
  waitpid (pid,NULL,0);
}
```