StuDocu.com

Shell - P2 2020

Sistemas Operativos (Universidade da Coruña)

StuDocu no está patrocinado ni avalado por ningún colegio o universidad.
Descargado por Manu Amestoy (manuamest@gmail.com)

```
// Autores: Tomás-David Aguado Domínguez  DNI: 45960153N
//        Xián García Ferreiro            DNI: 35632138D
// Logins:  tomas.david.aguado.dominguez
//        x.gferreiro
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <time.h>
#include <dirent.h>

#include <sys/stat.h>
#include <errno.h>
#include <pwd.h>
#include <grp.h>

#include <fcntl.h>
#include "list.c"

#include <sys/types.h>
#include <sys/resource.h>
#include <sys/wait.h>
#include <sys/mman.h>
#include <sys/shm.h>
#include <ctype.h>

#define LEERCOMPLETO ((ssize_t)-1)
int mem1, mem2, mem3;


int TrocearCadena(char * cadena, char * trozos[]){

int i=1;

      if ((trozos[0]=strtok(cadena," \n\t"))==NULL){
            return 0;}
      while ((trozos[i]=strtok(NULL," \n\t"))!=NULL){
            i++;}
            return i;
}



char TipoFichero (mode_t m) {

      switch (m&S_IFMT) { /*and bit a bit con los bits de formato,0170000 */
            case S_IFSOCK: return 's'; /*socket */
            case S_IFLNK: return 'l'; /*symbolic link*/
            case S_IFREG: return '-'; /* fichero normal*/
            case S_IFBLK: return 'b'; /*block device*/
            case S_IFDIR: return 'd'; /*directorio */
            case S_IFCHR: return 'c'; /*char device*/
            case S_IFIFO: return 'p'; /*pipe*/

            default: return '?'; /*desconocido, no deberia aparecer*/
      }
}
```

This document is available free of charge on StuDocu.com

Descargado por Manu Amestoy (manuamest@gmail.com)

```
char * ConvierteModo (mode_t m, char * permisos) {

      strcpy (permisos,"---------- ");
      permisos[0]=TipoFichero(m);

      if (m&S_IRUSR) permisos[1]='r'; /*propietario*/
      if (m&S_IWUSR) permisos[2]='w';
      if (m&S_IXUSR) permisos[3]='x';
      if (m&S_IRGRP) permisos[4]='r'; /*grupo*/
      if (m&S_IWGRP) permisos[5]='w';
      if (m&S_IXGRP) permisos[6]='x';
      if (m&S_IROTH) permisos[7]='r'; /*resto*/
      if (m&S_IWOTH) permisos[8]='w';
      if (m&S_IXOTH) permisos[9]='x';
      if (m&S_ISUID) permisos[3]='s'; /*setuid, setgid y stickybit*/
      if (m&S_ISGID) permisos[6]='s';
      if (m&S_ISVTX) permisos[9]='t';

      return permisos;
}

off_t TamanoFichero(char * nombre)
{
      struct stat fileStat;
      if(lstat(nombre, &fileStat) == -1) {
            return -1;
      } else {
            return fileStat.st_size;
      }
}
off_t InodoFichero(char * nombre)
{
      struct stat fileStat;
      if(lstat(nombre, &fileStat) == -1) {
            return -1;
      } else {
            return fileStat.st_ino;
      }
}




void infoFichero(int ltrozoso, char * nombre)
{
      struct stat fileStat;
      lstat(nombre,&fileStat);
      char string[50];
      time_t tmodif;
      struct tm *modif;

      if(lstat(nombre,&fileStat)){
            printf("No se pudo acceder a la información de %s: %s\n", nombre,
strerror(errno));
            return;
      }

      if (ltrozoso) {
```

Descargado por Manu Amestoy (manuamest@gmail.com)

```
            tmodif = fileStat.st_mtime;
            modif = localtime(&tmodif);

            strftime(string, sizeof(string), "%b %e %H:%M ", modif);
            printf( "%s ",string);
            printf("%6ld ", InodoFichero(nombre));
            printf("%s ",getpwuid(fileStat.st_uid)->pw_name);
            printf("%s ",getgrgid(fileStat.st_gid)->gr_name);
            printf("%s ", ConvierteModo(fileStat.st_mode,string));
            printf("%5ld ", TamanoFichero(nombre));
            printf("%3d ",(int)fileStat.st_nlink);
            printf("%s\n", nombre);
            if(S_ISLNK(fileStat.st_mode)){
                  readlink(nombre,string, fileStat.st_size);
                  string[fileStat.st_size] = '\0';
                  printf("%s\n",string);
            }

      } else {
            printf("%5ld ", TamanoFichero(nombre));
            printf("%s\n", nombre);
      }
};


void imprimirPrompt() {

      printf("-> ");    //esto si es poco serio podemos cambiarlo.
}



void leerEntrada(char * a){

      fgets(a, 1024, stdin);
      a[strlen(a) - 1] = '\0'; //  salto de linea

}



void Authors(char * trozos[],int nPalabras){

      if (nPalabras >= 2){
            if (strcmp("-n",trozos[1]) == 0) {
                  printf("Xián García Ferreiro \n");
                  printf("Tomás-David Aguado Dominguez \n");}
            if (strcmp("-l",trozos[1]) == 0) {
                    printf("x.gferreiro \n");
                  printf("tomas.david.aguado.dominguez \n");}
            }
      else{
            if(nPalabras==1){
                  printf("Xián García \n");
                  printf("Tomás-David Aguado Dominguez \n");
                  printf("x.gferreiro \n");
                  printf("tomas.david.aguado.dominguez \n");}
            }
}
```

This document is available free of charge on StuDocu.com

Descargado por Manu Amestoy (manuamest@gmail.com)

```
void Pid(char * trozos[],int nPalabras){

int npid;
      npid = getpid();
      printf("El pid del proceso es: %u \n",npid);
}



void PaPid(char * trozos[],int nPalabras){

int nppid;
      nppid = getppid();
      printf("El pid del proceso padre es: %u \n",nppid);
}



void Pwd(){

char cwd[PATH_MAX];
      if (getcwd(cwd, sizeof(cwd)) != NULL) {
            printf(" %s\n", cwd);
      }
      else {
            perror("getcwd() error");
      }
}



void Chdir(char * trozos[],int nPalabras){

      if (nPalabras == 2){
            if (chdir(trozos[1])==-1){
                  perror ("");}}
 else
            Pwd();
      }



void Date(char * trozos[]){

char output[128];
      time_t tiempo = time(0);
      struct tm *tlocal = localtime(&tiempo);
      strftime(output,128,"%d/%m/%y",tlocal);
        printf("%s\n",output);
}



void Time(char * trozos[]){

char output[128];
```

Descargado por Manu Amestoy (manuamest@gmail.com)

```
	time_t tiempo = time(0);
	struct tm *tlocal = localtime(&tiempo);
	strftime(output,128,"%H:%M:%S",tlocal);
	  printf("%s\n",output);
}



void listarFichero (int ltrozoso, int recursivo, int ocultos, int directorios, char
*nombre)
{
	struct stat fileStat;
	mode_t mode;
	lstat(nombre,&fileStat);
	mode = fileStat.st_mode;
	char tipo = TipoFichero(mode);
	char direccion[4096];
	struct dirent *entrada;
	DIR *dir;

	if (tipo != 'd') {
			infoFichero(ltrozoso, nombre);
	} else {
		dir = opendir(nombre);
		if (dir == NULL) {
			printf("Error al abrir el directorio %s\n", nombre);
		} else {
			if (directorios){
				infoFichero(ltrozoso, nombre);}
			if (recursivo) {
				while ((entrada = readdir(dir)) != NULL) {
					if(ocultos || ((entrada -> d_name[0]) != '.')) {
						strcpy(direccion,nombre);
						strcat(direccion,"/");
						strcat(direccion,entrada->d_name);
						if (!strcmp(entrada->d_name,".") || !
strcmp(entrada->d_name,"..")) {
							infoFichero(ltrozoso, direccion);
						} else {
							listarFichero(ltrozoso, 1, ocultos,
directorios, direccion);
						}
					}
				}
			} else {
				while ((entrada = readdir(dir)) != NULL) {
					if(ocultos || ((entrada -> d_name[0]) != '.')) {
						strcpy(direccion,nombre);
						strcat(direccion,"/");
						strcat(direccion,entrada->d_name);
						infoFichero(ltrozoso, direccion);
					}
				}
			}
		}
	}
};
```

This document is available free of charge on StuDocu.com

Descargado por Manu Amestoy (manuamest@gmail.com)

```
void Listar(char * trozos[])
{
	int ltrozoso = 0, recursivo = 0, ocultos = 0, primerNombre = 0,
directorios=1;
	char directorio[4096];

	for (int i = 1; trozos[i] != NULL; i++) {
		if (!strcmp("-rec", trozos[i])){
			recursivo = 1;
			continue;
		} else if (!strcmp("-hid", trozos[i])){
			ocultos = 1;
			continue;
		} else if (!strcmp("-long", trozos[i])){
			ltrozoso = 1;
			continue;
		} else if (!strcmp("-dir", trozos[i])){
			directorios = 0;
			continue;
		} else {
			primerNombre = i;
			break;
		}
	}

	if (primerNombre != 0) {
		for (int j = primerNombre; trozos[j] != NULL; j++) {
			listarFichero(ltrozoso, recursivo, ocultos, directorios,
trozos[j]);
		}
	} else {
		listarFichero(ltrozoso, recursivo, ocultos, directorios,
getcwd(directorio, 4096));
	}
};
```

```
void Create(char * trozos[], int nPalabras){

	int check;
	if (nPalabras == 1) {
		printf("Sintaxis de comando   create   incorrecta.\nAyuda: create [-
dir] nombre\n");}
	else {
	if (trozos[1]!=NULL && !strcmp("-dir", trozos[1])) {
		check = mkdir(trozos[2], 0700);
		if (!check)
				printf("Directorio creado\n");
		else
				printf("No se ha podido crear el directorio\n");
```

Descargado por Manu Amestoy (manuamest@gmail.com)

```
	} else {
		FILE *pf;
		pf = fopen(trozos[1], "w");
		fclose(pf);
	}}
}



void procesarDelete(char * nombre, int recursivo)
{
	struct stat fileStat;
	mode_t mode;
	lstat(nombre,&fileStat);
	mode = fileStat.st_mode;
	char tipo = TipoFichero(mode);
	char direccion[4096];
	DIR *dir;
	struct dirent *entrada;

	if (tipo != 'd') {
		if (unlink(nombre)){
			printf("No se ha podido borrar %s: %s\n", nombre,
strerror(errno));
		}
		return;
	} else if(!rmdir(nombre)) {
		return;
	} else if (recursivo) {
		dir = opendir(nombre);
		strcpy(direccion,nombre);
		while ((entrada = readdir(dir)) != NULL) {
			if((entrada -> d_name[0]) == '.')
				continue;
			strcpy(direccion,nombre);
			strcat(direccion,"/");
			strcat(direccion,entrada->d_name);
			procesarDelete(direccion, 1);
		}

		closedir(dir);
		procesarDelete(nombre, 0);
	} else if(rmdir(nombre)) {
		perror ("rmdir");}

};



void Delete(char * trozos[], int nPalabras)
{
	char directorio[4096];
	if (nPalabras==1){
		listarFichero(0, 0, 0, 1, getcwd(directorio, 4096));}
	else{
	for(int p=1;p<nPalabras;p++){
		if (trozos[p]!=NULL && !strcmp("-rec", trozos[1]) && trozos[p+1]!=NULL)
{
			procesarDelete(trozos[p+1], 1);
```

This document is available free of charge on StuDocu.com

Descargado por Manu Amestoy (manuamest@gmail.com)

```
            } else if (trozos[p]!=NULL && strcmp("-rec", trozos[1])) {
                  procesarDelete(trozos[p], 0);
            }
      }}
};

/
*****************************************************************************
********/
void * MmapFichero (char * fichero, int protection, TLISTA * listaMemoria)
{
      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){
            close(df);
            return NULL;
      }
      Elemento elemento;
      elemento.direccion= p;
      elemento.tamano= s.st_size;
      strcpy(elemento.modo,"mmap");
      strcpy(elemento.archivo,fichero);
      elemento.fecha= time(NULL);
      elemento.fd=df;
      InsertarElemento(listaMemoria,elemento);
      return p;
}

void * ObtenerMemoriaShmget (key_t clave, size_t tam)
{
      void * p;
      int aux,id,flags=0777;
      struct shmid_ds s;
      if (tam) /*si tam no es 0 la crea en modo exclusivo */
            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);
}


int buscarAddr(char *dir, TLISTA* listaMemoria)
```

Descargado por Manu Amestoy (manuamest@gmail.com)

```
{
	int pos = 1;
	do {
		if(dir == listaMemoria->elementos[pos].direccion){
			if(!strcmp(listaMemoria->elementos[pos].modo,"malloc")){
				printf ("block at address %p deallocated
(malloc)\n",listaMemoria->elementos[pos].direccion);
				free(listaMemoria->elementos[pos].direccion);
			}else if(!strcmp(listaMemoria->elementos[pos].modo,"mmap")){
				printf ("block at address %p deallocated
(mmap)\n",listaMemoria->elementos[pos].direccion);
				munmap(listaMemoria->elementos[pos].direccion,
listaMemoria->elementos[pos].tamano);
				close(listaMemoria->elementos[pos].fd);
			}else if(!strcmp(listaMemoria->elementos[pos].modo,"shared")){
				if (shmdt(listaMemoria->elementos[pos].direccion)==-1){
					perror ("shmctl: imposible eliminar memoria
compartida\n");
					return -1;
				}
				printf ("block at address %p deallocated
(shared)\n",listaMemoria->elementos[pos].direccion);
			} return pos;
		} else {
			pos++;
		}
	} while (pos<listaMemoria->current);
	return -1;
}


void AllocateMalloc(char * trozos[], TLISTA * listaMemoria)
{
	if (trozos[3] == NULL) {
		MostrarTodo(listaMemoria,"malloc");
	} else {
		char* endPtr;
		int Number = strtoull(trozos[3],&endPtr,10);
		void *p;
		if ((p = malloc(sizeof(Number)))==NULL){
			perror("Cannot allocate: ");
			return;
		}
		Elemento elemento;
		elemento.direccion= p;
		elemento.tamano= Number;
		strcpy(elemento.modo,"malloc");
		elemento.fecha= time(NULL);
		InsertarElemento(listaMemoria,elemento);
		printf ("Allocated %s at %p\n", trozos[3], p);

	}
}

void AllocateMmap (char * trozos[], TLISTA * listaMemoria)
{
	char *perm;
	void *p;
	int protection=0;
```

This document is available free of charge on StuDocu.com

Descargado por Manu Amestoy (manuamest@gmail.com)

```
	if (trozos[3]==NULL) {
			MostrarTodo(listaMemoria,"mmap");
			return;
		}
	if ((perm=trozos[4])!=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(trozos[3],protection,listaMemoria))==NULL)
		perror ("Cannot map file: ");
	else{
		printf ("file %s mapped at %p\n", trozos[3], p);
	}
}

void AllocateCreateShared (char *trozos[], TLISTA * listaMemoria)
{
	key_t key;
	size_t Number=0;
	void *p;
	if (trozos[3]==NULL || trozos[4]==NULL) {
		MostrarTodo(listaMemoria,"shared");
		return;
	}
	key=(key_t) atoi(trozos[3]);
	char* endPtr;
	if (trozos[4]!=NULL)
		Number=(size_t) strtoull(trozos[4],&endPtr,10);
	if ((p=ObtenerMemoriaShmget(key,Number))==NULL)
		perror ("Cannot allocate: ");
	else{
		Elemento elemento;
		elemento.direccion= p;
		elemento.tamano= Number;
		elemento.key=key;
		strcpy(elemento.modo,"shared");
		elemento.fecha= time(NULL);
		InsertarElemento(listaMemoria,elemento);
		Tamano_key tamano_key;
		tamano_key.key=key;
		tamano_key.tamano=Number;
		InsertarTamanoKey(listaMemoria,tamano_key);
		printf ("Allocated shared memory (key %d) at %p\n",key,p);
	}
}

int buscar_tamano(int key, TLISTA * listaMemoria){
	for(int i=1;i<listaMemoria->currentTamano_key;i++){
		if(key==listaMemoria->tamano_keys[i].key){
			return listaMemoria->tamano_keys[i].tamano;
			break;
		}
	}
	return 0;
}

void AllocateShared(char *trozos[], TLISTA * listaMemoria)
{
```

Descargado por Manu Amestoy (manuamest@gmail.com)

```
	key_t key;
	void *p;
	if (trozos[3] == NULL) {
		MostrarTodo(listaMemoria,"shared");
		return;
	} else {
		char* endPtr;
		key=(key_t) strtoull(trozos[3],&endPtr,10);
		if ((p=ObtenerMemoriaShmget(key,0))==NULL)
			perror ("Cannot allocate: ");
		else{
			Elemento elemento;
			elemento.direccion= p;
			elemento.tamano= buscar_tamano(key,listaMemoria);
			elemento.key=key;
			strcpy(elemento.modo,"shared");
			elemento.fecha= time(NULL);
			InsertarElemento(listaMemoria,elemento);

			printf ("Allocated shared memory (key %d) at %p\n",key,p);
		 }
	}
	return;
}

void ProcesarAllocate (char * trozos[], TLISTA * listaMemoria)
{
	if (trozos[1] == NULL) {
		MostrarTodo(listaMemoria,"todo");
	} else if (!strcmp("-malloc", trozos[2])){
		AllocateMalloc(trozos,listaMemoria);
	} else if (!strcmp("-mmap", trozos[2])){
		AllocateMmap(trozos, listaMemoria);
	} else if (!strcmp("-createshared", trozos[2])){
		AllocateCreateShared(trozos, listaMemoria);
	} else if (!strcmp("-shared", trozos[2])){
		AllocateShared(trozos, listaMemoria);
	}
}



void DeallocMalloc (char *trozos[], TLISTA* listaMemoria)
{
	char* endPtr;
		void* p;
		int Number;
	if (trozos[3] == NULL) {
		MostrarTodo(listaMemoria,"todo");
	} else {
		Number = strtoull(trozos[3],&endPtr,10);
		for (int i=1;i<listaMemoria->current;i++){
			if(!strcmp(listaMemoria->elementos[i].modo, "malloc") &&
listaMemoria->elementos[i].tamano==Number){
				p = listaMemoria->elementos[i].direccion;
				free(listaMemoria->elementos[i].direccion);
				EliminarElemento(listaMemoria, listaMemoria-
>elementos[i]);
				break;
```

This document is available free of charge on StuDocu.com

Descargado por Manu Amestoy (manuamest@gmail.com)

```
                }
            }
            printf ("block at address %p deallocated (malloc)\n",p);
      }
      return;
}

void DeallocMmap (char *trozos[], TLISTA* listaMemoria)
{
      void* p;
      int i;
      if (trozos[3] == NULL) {
            MostrarTodo(listaMemoria,"todo");
      } else {
            for (i=1;i<listaMemoria->current;i++){
                  if(!strcmp(listaMemoria->elementos[i].modo, "mmap") && !
strcmp(trozos[3], listaMemoria->elementos[i].archivo)){
                              p = listaMemoria->elementos[i].direccion;
                              munmap(listaMemoria->elementos[i].direccion,
listaMemoria->elementos[i].tamano);
                              close(listaMemoria->elementos[i].fd);
                              EliminarElemento(listaMemoria, listaMemoria-
>elementos[i]);
                              break;
                }
            }
            printf ("block at address %p deallocated (mmap)\n",p);
      }
      return;
}

void DeallocShared(char *trozos[], TLISTA* listaMemoria)
{
      int key, i;
      char* endPtr;
      void* p;
      if (trozos[3] == NULL) {
            MostrarTodo(listaMemoria,"todo");
      } else {
            key = strtoull(trozos[3],&endPtr,10);
            for (i=1;i<listaMemoria->current;i++){
                  if(!strcmp(listaMemoria->elementos[i].modo, "shared") &&
key==listaMemoria->elementos[i].key){

                        if (shmdt(listaMemoria->elementos[i].direccion)==-1){
                              perror ("shmctl: Cannot delete shared memory\n");
                              return;
                        }

                        p = listaMemoria->elementos[i].direccion;
                        EliminarElemento(listaMemoria, listaMemoria->elementos[i]);
                        break;
                }
            }
            printf ("block at address %p deallocated (shared)\n",p);
      }
      return;
}
```

Descargado por Manu Amestoy (manuamest@gmail.com)

```
void DeallocAddr (char *trozos[], TLISTA* listaMemoria)
{
      int  x;
      char *dir;
      dir = (char*) strtoul(trozos[3],NULL,16);
      if(dir == NULL){
            printf("Error: wrong direction\n");
            return;
      }
      x = buscarAddr(dir, listaMemoria);
      if (x < 0){
            MostrarTodo(listaMemoria,"todo");
      } else {
            EliminarElemento(listaMemoria, listaMemoria->elementos[x]);
      }
      return;
}

void ProcesarDealloc (char * trozos[], TLISTA * listaMemoria)
{
      if (trozos[2] == NULL) {
            MostrarTodo(listaMemoria,"todo");
      } else if (!strcmp("-malloc", trozos[2])){
            DeallocMalloc(trozos, listaMemoria);
      } else if (!strcmp("-mmap", trozos[2])){
            DeallocMmap(trozos,listaMemoria);
      } else if (!strcmp("-shared", trozos[2])){
            DeallocShared(trozos,listaMemoria);
      } else if (trozos[2] != NULL){
            DeallocAddr(trozos,listaMemoria);
      }
}


void ProcesarDeleteKey (char *trozos[],TLISTA* listaMemoria)
{
      key_t clave;
      int id;
      char *key;
      if (!strcmp("-shared", trozos[1])){
            key=trozos[3];
      } else {
            key=trozos[2];
      }
      if (key==NULL || (clave=(key_t) strtoul(key,NULL,10))==IPC_PRIVATE){
            printf (" rmkey clave_valida\n");
            return;
      }
      if ((id=shmget(clave,0,0666))==-1){
            perror ("shmget: impossible to get shared memory");
            return;
      }
      if (shmctl(id,IPC_RMID,NULL)==-1)
            perror ("shmctl: cannot delete shared memory\n");
      Tamano_key tamano_key;
      tamano_key.key=clave;
      EliminarTamano_Key(listaMemoria,tamano_key);
}
```

This document is available free of charge on StuDocu.com

Descargado por Manu Amestoy (manuamest@gmail.com)

```
void showMem (int malloc, int mmap, int shared, TLISTA* listaMemoria)
{
      if (malloc)
            MostrarTodo(listaMemoria,"malloc");
      if (mmap)
            MostrarTodo(listaMemoria,"mmap");
      if (shared)
            MostrarTodo(listaMemoria,"shared");
}

void ProcesarShow (char *trozos[], TLISTA * listaMemoria)
{
      int malloc = 0, mmap = 0, shared = 0;
      for (int i = 0; trozos[i+2] != NULL; i++) {
            if (!strcmp("-malloc", trozos[i+2])){
                  malloc = 1;
            } else if (!strcmp("-mmap", trozos[i+2])){
                  mmap = 1;
            } else if (!strcmp("-shared", trozos[i+2])){
                  shared = 1;
            } else if (!strcmp("-all", trozos[i+2])){
                  MostrarTodo(listaMemoria,"todo");
                  return;
            }
      }
      showMem(malloc, mmap, shared, listaMemoria);
}


void ShowFuncs(){
      printf("Memory address of ProcesarAllocate (program function): %p\n",
ProcesarAllocate);
      printf("Memory address of ProcesarDealloc (program function): %p\n",
ProcesarDealloc);
      printf("Memory address of ProcesarDeleteKey (program function): %p\n",
ProcesarDeleteKey);

      printf("Memory address of strcmp (library function): %p\n", strcmp);
      printf("Memory address of perror (library function): %p\n", perror);
      printf("Memory address of strtoul (library function): %p\n", strtoul);
}

void ShowVars(){
      int mem4, mem5, mem6;
      printf("Memory address of mem1 (extern variable): %p\n", &mem1);
      printf("Memory address of mem2 (extern variable): %p\n", &mem2);
      printf("Memory address of mem3 (extern variable): %p\n", &mem3);

      printf("Memory address of mem4 (authomatic variable): %p\n", &mem4);
      printf("Memory address of mem5 (authomatic varibale): %p\n", &mem5);
      printf("Memory address of mem6 (authomatic variable): %p\n", &mem6);
}

void Dopmap ()
{
      pid_t pid;
      char elpid[32];
```

Descargado por Manu Amestoy (manuamest@gmail.com)

```
      char *argv[3]={"pmap",elpid,NULL};
      sprintf (elpid,"%d", (int) getpid());
      if ((pid=fork())==-1){
            perror ("Cannot create process");
            return;
            }
      if (pid==0){
            if (execvp(argv[0],argv)==-1)
                  perror("cannot execute pmap");
            exit(1);
      }
      waitpid (pid,NULL,0);
}


void ProcesarMemory (char* trozos[], TLISTA* listaMemoria)
{
      if (!strcmp("-allocate",trozos[1])) {
            ProcesarAllocate(trozos,listaMemoria);
            return;
      }
      if (!strcmp("-dealloc",trozos[1])) {
            ProcesarDealloc(trozos,listaMemoria);
            return;
      }
      if (!strcmp("-show",trozos[1])) {
            ProcesarShow(trozos,listaMemoria);
            return;
      }
      if (!strcmp("-deletekey",trozos[1])) {
            ProcesarDeleteKey(trozos, listaMemoria);
            return;
      }
      if (!strcmp("-show-vars",trozos[1])) {
            ShowVars();
            return;
      }
      if (!strcmp("-show-funcs",trozos[1])) {
            ShowFuncs();
            return;
      }
      if (!strcmp("-dopmap",trozos[1])) {
            Dopmap();
            return;
      }
}


void Memdump (char *trozos[]){
      char *address;
      int size;

            address = (char*) strtoul(trozos[1],NULL,16);

    if (trozos[2] == NULL){
               size = 25;
            } else size = atoi(trozos[2]);

      int c = size / 25;
```

This document is available free of charge on StuDocu.com

Descargado por Manu Amestoy (manuamest@gmail.com)

```
	int m = size % 25;
	int desp = 0;
	int cont = 0;
	if (size < 25)
		desp = size;
	else desp = 25;

	for (int i = 0; i < size ; i= i + 25) {
		for(int j = 0; j< desp; j++){
			if(isprint(address[j+i])){
				printf("%3c",(address[j+i]));
			}else {printf("   ");}
		}

		putchar('\n');
		for( int j = 0; j < desp; j++){
			if(isprint(address[j+i])){
				printf("%3X",(address[j+i]));
			} else printf("  0");
			}
			putchar('\n');
		++cont;
		if (cont >= c) desp = m;
	}
		putchar('\n');
}


void Memfill (char* trozos[])
{
	int cont=128,i;
	char* dir;
	char a = 'A';
	if (trozos[1] == NULL) {
		return;
	}
	dir = (void*)strtoul(trozos[1],NULL,16);
	if (trozos[2] != NULL) {
		cont = atoi(trozos[2]);
	}
	if (trozos[3] != NULL) {
		a = strtoul(trozos[3],NULL,16);
	}
	for (i=0; i < cont; i++) {
		dir[i] = a;
	}
	return;
};

void doRecurse (int n)
{
	char automatico[4096];
	static char estatico[4096];
	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)
		doRecurse(n);
```

Descargado por Manu Amestoy (manuamest@gmail.com)

```
};

void Recurse (char *trozos[])
{
      int n;
      if (trozos[1] == NULL)
            return;
      n = atoi(trozos[1]);
      doRecurse (n);
}


ssize_t LeerFichero (char *fich, void *p, ssize_t n) /*n=-1 indica que se lea
todo*/
{
      ssize_t nleidos,tam=n;
      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 ReadFile (char *trozos[])
{
      ssize_t cont, x;
      char *dir;
      dir = (char*)strtoul(trozos[2],NULL,16);
      if (trozos[3] != NULL)
            cont = atoi(trozos[3]);
      else
            cont = -1;
      x = LeerFichero(trozos[1], dir, cont);
      if (x ==-1) {
            perror("Error al abrir el archivo");
            return;
      } else {
            printf("read %ld bytes of %s in %s\n", x, trozos[1], trozos[2]);
      }
}

void WriteFile (char *trozos[])
{
      int cont, flags, df;
      char *dir, *nombre;
      if(!strcmp(trozos[1],"-o") && trozos[2]!=NULL && trozos[3]!=NULL &&
trozos[4]!=NULL){
            nombre = trozos[2];
            cont = atoi(trozos[4]);
            dir = (char*)strtoul(trozos[3],NULL,16);
```

This document is available free of charge on StuDocu.com

Descargado por Manu Amestoy (manuamest@gmail.com)

```
            flags = O_WRONLY | O_CREAT | O_TRUNC;
      } else if (trozos[1]==NULL || trozos[2]==NULL || trozos[3]==NULL){
            printf("Faltan parametros\n");
            return;
      } else {
            nombre = trozos[1];
            cont = atoi(trozos[3]);
            dir = (char*)strtoul(trozos[2],NULL,16);
            flags = O_WRONLY | O_CREAT | O_EXCL;
      }
      if((df=open(nombre, flags, S_IRWXU | S_IRWXG | S_IRWXO))==-1){
            perror("Error al abrir el archivo");
            return;
      }
      write(df,dir,cont);
      close(df);
}

void Historico (char* trozos[], char *hist[], int nPalabras, int i, TLISTA*
listaMemoria){

int r = 0;
int tamanho=i;
      if (nPalabras >= 2) {
            if (strcmp("-c",trozos[1]) == 0){        //historic -c
                  for (r=0;r<=tamanho;r++){
                        hist[r]=NULL;
                        free(hist[r]);
                        }}

            else if (trozos[1][0]==45 && trozos[1][1]==114){ //historic -rN
                  int ncomand;
                  char *repeat;
                  char* trocos[1024];
                  if (1 == sscanf(trozos[1], "%*[^0123456789]%d", &ncomand)){
                        if (ncomand>0 && ncomand<=4096 && ncomand <=i){
                              repeat = hist[ncomand-1];}
                              int valor=0;
                              int nPalobras=TrocearCadena(repeat,trocos);


                                    if (!strcmp("getppid",trocos[0])) {valor = 1;}
                                    if (!strcmp("getpid",trocos[0])) {valor = 2;}
                                    if (!strcmp("authors",trocos[0])) {valor = 3;}
                                    if (!strcmp("date",trocos[0])) {valor = 4;}
                                    if (!strcmp("time",trocos[0])) {valor = 5;}
                                    if (!strcmp("chdir",trocos[0])) {valor = 6;}
                                    if (!strcmp("pwd",trocos[0])) {valor = 7;}
                                    if (!strcmp("historic",trocos[0])) {valor = 8;}
                                    if (!strcmp("create",trocos[0])) {valor = 9;}
                                    if (!strcmp("delete",trocos[0])) {valor = 10;}
                                    if (!strcmp("list",trocos[0])) {valor = 11;}

                                    if (!strcmp("memory",trocos[0])) {valor = 12;}
                                    if (!strcmp("memdump",trocos[0])) {valor = 13;}
                                    if (!strcmp("memfill",trocos[0])) {valor = 14;}
                                    if (!strcmp("recurse",trocos[0])) {valor = 15;}
                                    if (!strcmp("readfile",trocos[0])) {valor =
16;}
```

Descargado por Manu Amestoy (manuamest@gmail.com)

```
                                    if (!strcmp("writefile",trocos[0])) {valor =
17;}

                                    switch (valor) {
                                          case 1: PaPid(trocos, nPalobras); break;
                                          case 2: Pid(trocos, nPalobras); break;
                                          case 3: Authors(trocos,nPalobras); break;
                                          case 4: Date(trocos); break;
                                          case 5: Time(trocos); break;
                                          case 6: Chdir(trocos,nPalobras); break;
                                          case 7: Pwd(); break;
                                          case 8: Historico(trocos, hist,
nPalobras, i, listaMemoria); break;
                                          case  9: Create(trocos,nPalobras); break;
                                          case 10: Delete(trocos, nPalobras);
break;
                                          case 11: Listar(trocos); break;
                                          case 12: ProcesarMemory (trocos,
listaMemoria); break;
                                          case 13: Memdump (trocos); break;
                                          case 14: Memfill (trocos); break;
                                          case 15: Recurse (trocos); break;
                                          case 16: ReadFile (trocos); break;
                                          case 17: WriteFile (trocos); break;

                                          default: printf("No se reconoce el
comando.\n");}}}

            else if ((atoi(trozos[1])) <=0 && (atoi(trozos[1])) >=-4096 &&
(atoi(trozos[1]))>-i ){ //historic -N
                  int limite = abs(atoi(trozos[1]));
                  for (r=0;r<=limite-1;r++){
                  printf("%s \n", hist[r]);}
                  }}
      else {
      if (nPalabras == 1){
            for (r=0;r<=tamanho-1;r++){
                  printf("%s \n", hist[r]);}
            }
      }
}



void procesarEntrada(char* trozos[],char * a, char *hist[], int i, TLISTA*
listaMemoria){

int valor=0;
int nPalabras=TrocearCadena(a,trozos);

      /*Práctica 0*/
      if (!strcmp("getppid",trozos[0])) {valor = 1;}
      if (!strcmp("getpid",trozos[0])) {valor = 2;}
      if (!strcmp("authors",trozos[0])) {valor = 3;}
      if (!strcmp("date",trozos[0])) {valor = 4;}
      if (!strcmp("time",trozos[0])) {valor = 5;}
      if (!strcmp("chdir",trozos[0])) {valor = 6;}
      if (!strcmp("pwd",trozos[0])) {valor = 7;}
      if (!strcmp("historic",trozos[0])) {valor = 8;}
```

This document is available free of charge on StuDocu.com

Descargado por Manu Amestoy (manuamest@gmail.com)

```
      if (!strcmp("end",trozos[0]) || !strcmp("quit",trozos[0]) || !
strcmp("exit",trozos[0])) {exit(0);}
      /*Práctica 1*/
      if (!strcmp("create",trozos[0])) {valor = 9;}
      if (!strcmp("delete",trozos[0])) {valor = 10;}
      if (!strcmp("list",trozos[0])) {valor = 11;}
      /*Práctica 2*/
      if (!strcmp("memory",trozos[0])) {valor = 12;}
      if (!strcmp("memdump",trozos[0])) {valor = 13;}
      if (!strcmp("memfill",trozos[0])) {valor = 14;}
      if (!strcmp("recurse",trozos[0])) {valor = 15;}
      if (!strcmp("readfile",trozos[0])) {valor = 16;}
      if (!strcmp("writefile",trozos[0])) {valor = 17;}
            switch (valor) {
                  /*Práctica 0*/
                  case 1: PaPid(trozos, nPalabras); break;
                  case 2: Pid(trozos, nPalabras); break;
                  case 3: Authors(trozos,nPalabras); break;
                  case 4: Date(trozos); break;
                  case 5: Time(trozos); break;
                  case 6: Chdir(trozos,nPalabras); break;
                  case 7: Pwd(); break;
                  case 8: Historico(trozos, hist, nPalabras, i, listaMemoria);
break;
                  /*Práctica 1*/
                  case  9: Create(trozos, nPalabras); break;
                  case 10: Delete(trozos, nPalabras); break;
                  case 11: Listar(trozos); break;
                  /*Práctica 2*/
                  case 12: ProcesarMemory (trozos, listaMemoria); break;
                  case 13: Memdump (trozos); break;
                  case 14: Memfill (trozos); break;
                  case 15: Recurse (trozos); break;
                  case 16: ReadFile (trozos); break;
                  case 17: WriteFile (trozos); break;
                  default: printf("No se reconoce el comando.\n");
            }

}



int main() {  // Devuelve un  int que es el 0 de la intruccion return o exit
      int terminado = 0;
      char* trozos[1024];
      char a[1024];
      char *hist[4096];
      char *auxi;
      int i=0;
      TLISTA* listaMemoria = malloc(sizeof listaMemoria * (MAXLISTAMEMORIA + 1));
      CrearLista(listaMemoria);

    while (!terminado){
      imprimirPrompt();
      leerEntrada(a);
      auxi=strdup(a);
      hist[i]=auxi;
      i=i+1;
      if ((!strcmp("historic -c",auxi))) {i=0;}
```

Descargado por Manu Amestoy (manuamest@gmail.com)

```
        procesarEntrada(trozos,a, hist, i, listaMemoria);


    }
      free (auxi);
}
```

This document is available free of charge on StuDocu.com

Descargado por Manu Amestoy (manuamest@gmail.com)