Fundamentos de los Sistemas Operativos

Departamento de Informática de Sistemas y Computadoras (DISCA)

Universitat Politècnica de València

fSO

Práctica 8
Proyección de Archivos en memoria (II)

1 Objetivos

El objetivo principal de esta práctica es **familiarizarse con el concepto de proyección de archivos en memoria.** Para ello trabajaremos con los servicios disponibles en POSIX para llevar a cabo esta operación y analizaremos el mapa de memoria de procesos con un archivo de texto proyectado visualizando su archivo ***/proc/PID/maps*** para identificar las diferentes regiones que en él aparecen y su correspondencia con el código fuente. También se analizará cómo influye la proyección de un archivo con la creación de **múltiples procesos o hilos por parte de uno dado**. En esta práctica se trabaja la estructura utilizada en los sistemas Linux para la arquitectura PC.

Nota: En esta práctica se hace referencia a todas las descripciones y aspectos del mapa de memoria descritos en el seminario 7 y la práctica 7. Se supone que se trabaja con un sistema Linux de 64 bits; la opción de compilación "–m32" permite definir mapas de memoria de 32 bits para los programas creados.

2 Mapa de Memoria: proceso con archivo mapeado o proyectado

La técnica de memoria virtual ofrece al usuario una forma alternativa de acceder a los archivos. El sistema operativo permite que un archivo o parte de un archivo se corresponda con una zona del mapa de memoria de un proceso, proyectando/mapeando el archivo en el mapa de memoria de procesos. Una zona del mapa de memoria del proceso contendrá una copia idéntica del contenido de los bloques de dicho archivo en disco, como representa la figura-1. Una vez que el fichero está proyectado, acceder a él es más rápido que a disco, ya que no son necesarias llamadas al sistema.

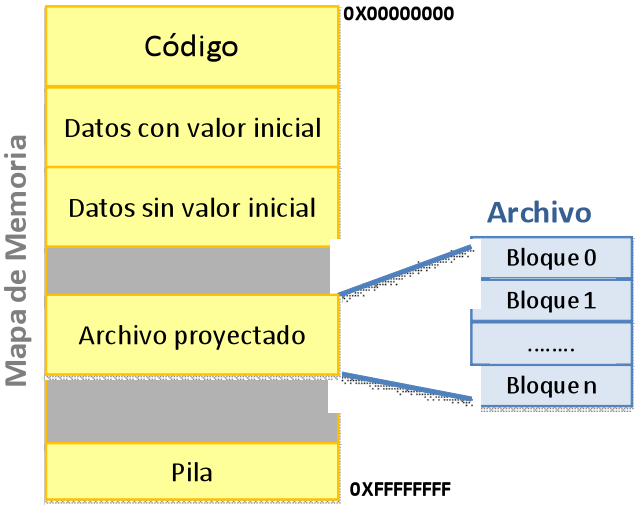

Figura1. El archivo es proyectado sobre el mapa de memoria del proceso

2.1 Herramientas y órdenes: mmap y munmap

POSIX presenta las llamadas *mmap()* y *munmap()* para proyectar y eliminar la proyección de archivos en memoria.

```
#include <unistd.h>
#include <sys/mman.h>
caddr_t mmap(void *start, size_t length, int protec , int flags,
                                                  int fd, off_t offset);

int munmap(void *start, size_t length);
```

La función *mmap()* intenta ubicar un número `length` de bytes en memoria del archivo especificado en `fd` comenzando a partir del desplazamiento `offset` de dicho archivo.

El parámetro *start* indica la dirección del espacio lógico preferente o deseada a partir de la cual ubicar en el mapa de memoria el archivo. Esta última dirección es una sugerencia y normalmente se especifica como 0. El lugar real donde es ubicado el objeto es devuelto por mmap().

El parámetro *lenght* expresa el número de bytes del archivo que se desea proyectar en memoria. Si se desea proyectar archivo completo se ha de proporcionar la longitud en bytes del archivo.

El parámetro *protec* establece la protección de la región. Esta protección debe ser compatible con el modo de apertura del archivo y puede ser:

- PROT_READ región de lectura
- PROT_WRITE región de escritura
- PROT_EXEC región de ejecución
- cualquier combinación de las anteriores

El parámetro *flags* permite establecer ciertas propiedades para la región insertada como son las opciones de asociación y si las modificaciones hechas a la copia insertada en memoria son privadas al proceso o son compartidas por otras referencias. Puede ser:

•MAP_FIXED: El archivo debe proyectarse justo en la dirección especificada. Si la dirección especificada no puede ser utilizada, mmap() debe fallar.

•MAP_SHARED: La región es compartida, las modificaciones sobre esta región afectarán al fichero. Un proceso hijo comparte esta región con el padre.

•MAP_PRIVATE: La región es privada, las modificaciones sobre esta región no afectan al fichero. Un proceso hijo no comparte esta región con el padre, sino que obtiene un duplicado de la misma.

Valor devuelto

Puntero al área de memoria reservada: si mmap() tiene éxito.

–1: En caso de error

La llamada al sistema ***munmap***() libera las ubicaciones para el rango de direcciones especificado.

2.2 Ejercicio 1 : Mapa de memoria con proyección de archivo

El contenido del archivo `mapear_prac.c` proporcionado con la práctica es el mostrado en la figura 2.

```
/** Contenido del fichero mapear_prac.c ****/
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
```

```
void error (char * mensaje)
{
  fprintf(stderr, "%s", mensaje);
    exit(EXIT_FAILURE);
}
void construye_orden(char orden[80])
{
 //Construye orden para mostrar MAPA MEMORIA
  sprintf(orden,"cat /proc/%d/maps",getpid());

}
int main (int argc,char *argv[])
{   int fd;
    char *mapeo;
    struct stat stadbuf;
    char path_maps[80];

    //Abrir el archivo a ser mapeado
    if (argc!=2) {
      fprintf(stderr,"Usar: mapear NombreArchivo\n");
      exit(EXIT_FAILURE);
    }

    if ((fd=open(argv[1],O_RDONLY))<0)
    error("Fallo en la apertura (open)\n");

    //Obtener la longitud del archivo a mapear
    fstat(fd, &stadbuf); //fstat vuelca su informacion en estadobuf

   //MOSTRAR MAPA
    printf("El fichero proyectado es %s de longitud %d\n",argv[1], stadbuf.st_size);
    printf(" MAPA DE MEMORIA DEL PROCESO /proc/%d/maps \n", getpid());
    construye_orden(path_maps);
    system(path_maps);//Llamada al sistema para ejecutar orden

  //Mapear el archivo de entrada
    if ((mapeo=mmap(0,stadbuf.st_size,PROT_READ,MAP_SHARED,fd,0)) == MAP_FAILED)
    error("Fallo al mapear (open)");
    close(fd); //cierro fichero

  //MOSTRAR MAPA
   printf ("\n\n FICHERO MAPEADO EN MEMORIA\n");
   system(path_maps);//Llmada al sistema para ejecutar orden

   munmap(mapeo,stadbuf.st_size); //Elimina mapeo

  printf ("\n\n ELIMINADO EL MAPEO DEL FICHERO EN MEMORIA\n");
  system(path_maps);
  exit(EXIT_SUCCESS);

}
```

Figura2. Código del archivo `mapear_prac.c`

Compile *mapear_prac.c* y ejecútelo. Recuerde que al ejecutarlo debe pasarle como parámetro el nombre del fichero a proyectar. Escriba, por tanto, lo siguiente:

```
$gcc -m32 mapear_prac.c -o mapear1

$./mapear1 mapear_prac.c
```

Al ejecutar el código se imprime el mapa de memoria antes, durante y después de la proyección.

Cuestión 1: Analice los diferentes mapas mostrado y responda a las siguientes cuestiones:

1. ¿Qué diferencias encuentra entre los mapas de generados antes y después de la proyección? f7784000-f7785000 r--s 00000000 08:01 1324072 /home/virtual/Desktop/PL08+codigo/mapear_prac.c aparece esto
2. ¿Qué permisos tiene la región donde se ha proyectado el archivo? permisos de lectura y compartida r--s
3. ¿Qué tamaño tiene la región donde se ha proyectado el archivo? f7784000-f7785000 1 en hexadecimal

2.3 Ejercicio 2: Accediendo a las posiciones de memoria de la proyección

El archivo `mapear_leer.c`, que se proporciona con el material de prácticas, contiene un código que es básicamente el del ejercicio anterior, figura-2, salvo que hay una nueva función *contar_caracteres* y la llamada a dicha función, como muestra en la figura 3.

```
//Función que cuenta el número de apariciones de un carácter en un archivo
int contar_caracteres(char caracter, char *dirlog, int longitud)
{int i,contador=0;
  for (i=0; i<longitud; i++)
    if (dirlog[i]==caracter) contador++;
  return contador;
}

int main ()

{……
num_car=contar_caracteres(argv[2][0],mapeo,stadbuf.st_size);
fprintf(stderr,"%s contiene %d caracteres %C\n", argv[1],num_car,argv[2][0]);
……
}
```

Figura3. Parte del código que contiene el archivo mapear_leer.c

Cree un archivo pequeño denominado *hola*. Compile *mapear_leer.c*, y ejecútelo. Recuerde que al ejecutarlo debe pasarle como parámetro el nombre del archivo a proyectar y un carácter:

```
$echo hola alumno de FSO que haces esta practica> hola

$gcc -m32 mapear_leer.c –o mapearleer

$cat hola

$./mapearleer hola o
```

Al ejecutar el código de `mapearleer` se imprime el mapa de memoria antes, durante y después de la proyección del archivo.

Cuestión 2: Analice los diferentes mapas mostrado y responda a las siguientes cuestiones:

1. ¿Con qué instrucciones ha accedido a la información del archivo proyectado?
(mapeo=mmap(0,stadbuf.st_size,PROT_READ,MAP_SHARED,fd,0)

2. ¿Qué permisos tiene la región donde se ha proyectado el archivo?
PROT_READ region de lectura MAP_SHARED region compartida

3. ¿Qué tamaño tiene la región donde se ha proyectado el archivo?
f7762000-f7763000 1 en hexadecimal

Modifique en el archivo `mapear_leer.c` el parámetro `flag` de la llamada `mmap` para que sea privado.

```
if ((mapeo=mmap(0,stadbuf.st_size,PROT_READ,MAP_PRIVATE,fd,0)) == MAP_FAILED)
```

Compile y ejecute de nuevo con los mismos argumentos de antes

4. Justifique los permisos y el tipo de acceso que tiene la región donde se ha proyectado el archivo
PROT_READ region de lectura MAP_PRIVATE region privada

2.4 Ejercicio 3: Escribiendo en las posiciones de memoria de la proyección

El archivo `mapear_escribir.c` contiene un código equivalente al del ejercicio anterior, con una nueva función escribir_ficheros y la llamada a dicha función, como se muestra en figura 4.

```
//Función que escribe caracteres en un archivo mapeado
int escribir_fichero(char *frase, char *dirlog, int longitud)
{ int i,long_frase, sucess=-1;

  long_frase=strlen(frase);
  printf("longitud frase %d, longitud del fichero %d\n",long_frase,longitud);
  for(i=0; i<long_frase; i++)
     dirlog[i]=frase[i];

  if (long_frase==i) sucess=1;

  return sucess;
}
int main ()
{
…
    if ((escribir_fichero(argv[3],mapeo,stadbuf.st_size))<0)
     fprintf(stderr,"Error al escribir\n");
….
}
```

Figura 4. Este código es parte del archivo `mapear_escribir.c` proporcionado

Compile *mapear_escribir.c* y ejecútelo. Note que al ejecutarlo debe pasarle como parámetro el nombre del archivo a proyectar, un carácter y una palabra a escribir. Trabaje con el archivo *hola* creado en el ejercicio anterior y haga lo siguiente:

```
$gcc -m32 mapear_escribir.c –o mapearescribir
$cat hola
$./mapearescribir hola F Tomas
$cat hola
```

Al ejecutar el código se imprime el mapa de memoria antes después de la proyección del archivo.

Cuestión 3: Analice los diferentes mapas mostrado y responda a las siguientes cuestiones:

1. ¿Con qué instrucciones ha accedido a la información del archivo? (mapeo=mmap(0,stadbuf.st_size,PROT_READ|PROT_WRITE,MAP_SHARED,fd,0)
2. ¿Qué permisos tiene la región donde se ha proyectado el archivo? rw-s
3. ¿Qué permisos se han utilizado en la llamada *open* del código proporcionado? PROT_READ|PROT_WRITE,MAP_SHARED
4. ¿Se ha modificado el contenido del fichero *hola*? Justifíquelo si: Tomasalumno de FSO que haces esta practica reemplaza la frase hola por tomas dirlog[i]=frase[i];

3 Mapa de memoria: proceso hijo

En este apartado se trabaja con la llamadas *fork()* y *exec()* observando los cambios que se producen en el mapa de memoria de los procesos. Recuerde que:

- **fork()** crea un proceso hijo, que será una réplica exacta del proceso padre

- **exec()** reemplaza la imagen de memoria del proceso la del archivo ejecutable que se especifique.

3.1 Ejercicio 4: Mapa de memoria de dos procesos con relación padre-hijo

El archivo `maphijo.c` proporcionado en el material de prácticas, contiene el código mostrado en la figura 5. El programa imprime el mapa de memoria del proceso y a continuación realiza una llamada *fork()*. El proceso hijo imprime su mapa de memoria inicial, así como el mapa después de realizar la llamada

```
execlp("cat","cat",path_maps,NULL);
```

que cambiará la imagen de memoria por la del ejecutable `cat /proc/PID/maps`

```
///Archivo maphijo.c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

void construye_orden(char orden[80])
{
 //Construye orden para mostrar MAPA MEMORIA
  sprintf(orden,"cat /proc/%d/maps",getpid());
```

```
}

int main ()
{
    int val_pid;
    char path_maps[80];


    printf(" MAPA DE MEMORIA DEL PROCESO /proc/%d/maps \n", getpid());
    construye_orden(path_maps);
    system(path_maps);  //LLamada al sistema para ejecutar orden

    val_pid=fork();//se crea un proceso

    if(val_pid==0)
      { printf("MAPA DEL PROCESO HIJO %d\n",getpid());
        construye_orden(path_maps);
        system(path_maps); //LLamada al sistema para ejecutar orden

        printf("\n MAPA MEMORIA PROCESO %d DESPUES DE exec \n", getpid());
        sprintf(path_maps,"/proc/%d/maps",getpid());
        execlp("cat","cat",path_maps,NULL);

        exit(0);
      }

    if (val_pid==-1)
     { printf("No se ha podido crear el hijo \n");
       exit(-1);
     }
    if (val_pid>0) wait(); //Padre espera

    printf(" MAPA DE MEMORIA DEL PROCESO /proc/%d/maps \n", getpid());
    construye_orden(path_maps);
    system(path_maps); //LLamada al sistema para ejecutar orden
    exit(0);
}
```

Figura5. Código del archivo *maphijo.c*

Compile *mahijo.c* y ejecútelo:

```
$gcc -m32 maphijo.c -o maphijo

$./maphijo
```

Cuestión 4: Analice los diferentes mapas mostrado y responda a las siguientes cuestiones:

1. Justifique similitudes y diferencias entre el mapa del proceso padre y el del hijo antes del cambio de memoria

son iguales porque el hijo es copia del padre

2. Justifique similitudes y diferencias entre el mapa del proceso hijo antes y después de ejecutar la llamada *exec()*

el hijo ejecuta un codigo distinto al del padre, por lo que ahora si que canvia el mapa: 7fd382bec000-7fd382ff5000 r--p 00000000 08:01 918394 /usr/lib/locale/locale-archive

4 Mapa de memoria: procesos con hilos

Trataremos de analizar el mapa de un proceso que crea hilos de ejecución o threads. Recuerde que en la práctica 5 se trabajó con las llamadas para inicializar, crear y esperar hilos.

La función *pthread_attr_init* se encarga de asignar unos valores por defecto a los elementos de la estructura de atributos de un hilo. Si no se inicializan los atributos, el hilo no se puede crear. La función *pthread_create* es la encargada de crear el hilo

```
int pthread_attr_init(pthread_attr_t *attr);

int  pthread_create(pthread_t  *thread, pthread_attr_t *attr,
                                        void *(*start_routine)(void *), void *arg);
```

4.1 Ejercicio 5: Mapa de memoria de un proceso que crea hilos

El archivo *maphilo.c*, proporcionado con el material de prácticas, contiene el código mostrado en la figura-6. El programa imprime el mapa de memoria de un proceso, y a continuación realiza una llamadas *pthread_create* para crear hilos de ejecución. El proceso imprime su mapa de memoria inicial, así como el mapa después de realizar la creación de hilos

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <pthread.h>

void construye_orden(char orden[80])
{//Construye orden para mostrar MAPA MEMORIA
  sprintf(orden,"cat /proc/%d/maps",getpid());
}

void  *Imprime( void *ptr)
{char *men;
  men=(char *) ptr;
  fprintf(stderr,"El hilo dice:%s\n", men);
  pthread_exit(NULL);
}

int main ()
{   int val_pid;
    char path_maps[80];
    pthread_attr_t atrib;
    pthread_t hilo1,hilo2, hilo3;

    printf(" MAPA DE MEMORIA DEL PROCESO /proc/%d/maps \n", getpid());
    construye_orden(path_maps);
    system(path_maps);  //LLamada al sistema para ejecutar orden

    //creo los hilos con atributos
     pthread_attr_init(&atrib);
    pthread_create(&hilo1, &atrib,Imprime, "hola de hilo1");
    pthread_create(&hilo2, &atrib,Imprime, "hola de hilo2");
    pthread_create(&hilo3, &atrib,Imprime, "hola de hilo3");

    printf("MAPA DE MEMORIA PROCESO con hilos /proc/%d/maps \n", getpid());
    construye_orden(path_maps);
    system(path_maps); //LLamada al sistema para ejecutar orden
```

```
    pthread_join(hilo1,NULL);
    pthread_join(hilo2,NULL);
    pthread_join(hilo3,NULL);

    exit(0);
}
```

Figura 6. Este código se encuentra en *maphilo.c*

Compile *maphilo.c* y ejecútelo:

```
$gcc -m32 maphilo.c -lpthread -o maphilo1

$./maphilo1
```

Cuestión 5: Analice los diferentes mapas mostrado y responda a las siguientes cuestiones:

1. Justifique similitudes y diferencias entre los mapas antes y después de crear los hilos
   la memoria dinamica, espacio heap, aumenta.

2. ¿Cómo podría justificarse el que aparezca una región *heap* si no se ha invocado *malloc()*?
   los hilos deben tener su heap para poder funcionar, en fork se hacen copias pero en hilos se usa memoria dinamica

Edite maphilo.c y escriba en él el código necesario para crear un total de 6 hilos de ejecución. Compile *mahilo.c* y ejecútelo:

```
$gcc -m32 maphilo.c -lpthread -o maphilo2

$./maphilo2
```

3. Compare los mapas de maphilo1 y maphilo2 y justifique las diferencias entre ellos
   maphilo2 tiene un espacio más amplio que maphilo1, ya que tiene mas hilos, por lo que reserva mas memoria dinamica

4. ¿Qué permisos poseen las regiones creadas para soportar los nuevos hilos?
   rw-p ---p

5. En la función que ejecutan los hilos hay una variable local. Muestre por pantalla la dirección de esa variable e identifique en qué región se encuentra
   las variables locales se encuentran en el heap ya que las ejecutan los hijos