Fonaments dels Sistemes Operatius

Departament d'Informàtica de Sistemes i Computadores (DISCA)

Universitat Politècnica de València

fso

Pràctica 8
Anàlisi del Mapa de Memòria

1 Objectius

L'objectiu principal d'aquesta pràctica és **"analitzar el mapa de memòria d'un procés" i comprendre** l'evolució que pateix durant l'execució del mateix. Per a fer-ho, visualitzarem l'arxiu ***/proc/PID/maps*** de Linux.

2 Mapa de Memòria d'un procés

L'espai d'adreces lògiques d'un programa conté les diferents seccions en les que s'estructura la seua memòria: àrea de codi, àrea de dades, àrees per a les biblioteques de enllaç dinàmic, àrea per a pila, etc. Aquesta col·lecció de seccions marca l'estructura del mapa de memòria d'un procés i cada sistema operatiu utilitza la seua pròpia.

El mapa de memòria d'un procés es gestionat pel sistema operatiu i evoluciona durant l'execució del mateix. Existeix una gran similitud o correspondència entre el contingut de l'arxiu executable que suporta un procés i el seu mapa de memòria inicial.

En Linux, l'espai de adreces lògiques d'un programa defineix la organització interna del codi d'usuari que el sistema operatiu ubicarà en memòria per a generar el procés. Aquesta pràctica treballa amb el model d'adreces lògiques dels arxius *ELF (Executable and Linking Format)* per a versions Linux amb arquitectura PC.

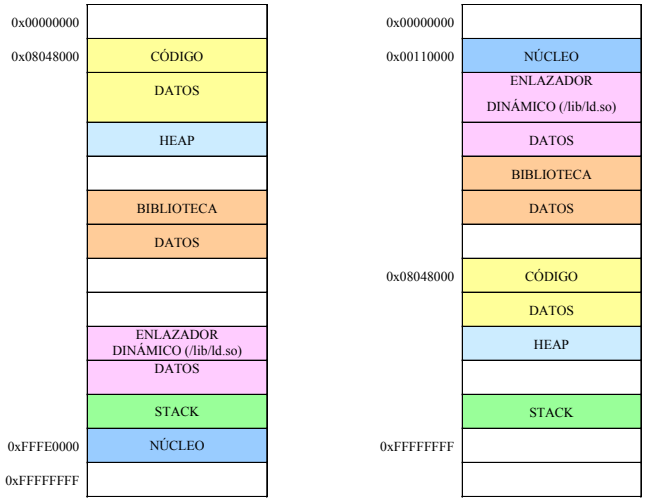

Figura1. Dues possibles organitzacions de la memòria lògica d'un procés per a versions Linux 2.6.x en màquines de 32 bits.

2.1 Mapes de memòria de 32 i 64 bits

Existeixen sistemes operatius per a arquitectures de 32 bits i de 64 bits, es dir, per a processadors de 32 i 64 bits d'adreça. Amb 32 bits s'adrecen espais lògics de hasta 4GB (2^{32}= 4 GB), mentre que amb 64 bits s'adrecen espais lògics de fins que 16 EB (2^{64} = 16 ExaByte).

Dins de la família Intel de processadors, la compatibilitat cap a enrere permet instal·lar un sistema operatiu de 32 bits sobre un processador de 64 bits i executar processos de 32 bits sota un sistema operatiu de 64 bits. Tanmateix, no és possible gestionar processos de 64 bits en un sistema operatiu de 32 bits, tampoc funciona un sistema operatiu de 64 bits sobre un processador de 32 bits. Poden coexistir processos de 64 i de 32 bits en un sistema operatiu de 64 bits, però en un sistema operatiu de 32 bits només hi ha processos de 32 bits.

El compilador de Linux permet triar quin tipus de mapa d'adreces utilitzarà el codi que genere. Allò habitual és que el compilador només dispose de les biblioteques apropiades per al sistema amb el que opera i que, per omissió, genere el codi d'acord amb això. Però es pot fer una compilació "creuada" si disposa de les biblioteques necessàries, es a dir, podrà generar codi per a un tipus de sistema diferent. Per a processadors Intel, gcc genera codi de 32 bits utilitzant l'opció "-m32". Aquesta és la opció per omissió en les versions de linux 32 bits, però cal explicitar-la en la línia d'ordres per a sistemes de 64 bits.

En aquesta pràctica anem a utilitzar mapes de memòria de 32 bits. En les **ordres de compilació**, ha de ficar l'opció **"-m32"** ja que el sistema operatiu instal·lat en el laboratori és de 64 bits. Si voleu, pot provar a analitzar un mapa de 64 bits, i comprovar la seua complexitat.

2.2 Regions del mapa de memòria i les seues propietats

El mapa de memòria d'un procés està format per varies regions, com es mostra en la figura-1. Cada **regió** és una **zona contigua de l'espai lògic del procés** caracteritzada per l'adreça dins del mapa, la grandària i una sèrie de propietats:

- **Suport de la regió**. Existeixen dos possibilitats
  - *Regió amb suport en arxiu*: El contingut de la regió s'emmagatzema en un arxiu.
  - *Regió sense suport*: La regió no té contingut inicial.

- **Tipus d'us de la regió**. Indica si està permès compartir la regió entre processos.
  - *Privada (p):* El contingut de la regió només és accessible pel procés.
  - *Compartida (s):* El contingut d'aquesta regió pot ser accessible por diversos processos.

- **Protecció:** Tipus d'accés permès a la regió: lectura (r), escriptura (w), execució (x).

- **Grandària fixa o variable**: Existeixen regions la grandària del qual pot variar durant l'execució del procés i altres que la grandària es fixa (no varia).

Les regions en el **mapa de memòria inicial** del procés són:

- **Codi o text.** Regió compartida de lectura (r) /execució (x), grandària fixa i suportada pel fitxer executable. Una regió marcada como "p" sense permisos de escriptura, en la pràctica estaráà compartida. En versions anteriors del nucli de Linux, aquestes es marquen amb la marca "s".
  r-xp Regió de codi

- **Dades amb valor inicial**. Regió privada (p), cada procés necessita una còpia pròpia de les variables, és de lectura(r)/escriptura(w), grandària fixa i suportada pel fitxer executable.
  rw-p. Regió de dades inicialitzades quan està suportada per un fitxer (DADES).

- **Dades sense valor inicial**. Regió privada (p), és de lectura(r)/escriptura(w), grandària fixa i no està suportada ja que el seu contingut inicial és irrellevant.
  rw-p. Regió de dades no inicialitzats quan no està suportada por un fitxer (BSS).

- **Pila**. Regió privada (p) de lectura(r)/escriptura(w) de grandària variable, serveix de suport per a emmagatzemar les variables locals, paràmetres i adreces de retorn de les crides a funcions.

El mapa de memòria d'un procés és dinàmic, amb regions que poden afegir-se o eliminar-se durant la seua execució. Algunes de les noves regions que poden crear-se són:

- **Heap.** Regió privada de r/w sense suport (inicialment s'ompli a zeros) i que creix segons reserva memòria el procés i decreix quan l'allibera. Serveix de suport a la memòria dinàmica que un procés reserva en temps d'execució (en llenguatge C utilitzant la funció "malloc").

•**Arxius projectats**. Quan es projecta un arxiu sobre l'espai lògic del procés se crea una regió per a això, la protecció de la qual l'especifica el procés a l'hora de projectar-la.

•**Memòria Compartida**. Quan es crea una zona de memòria compartida, apareix una regió de caràcter compartit en el mapa.

•**Piles de threads**. Cada *thread* necessita una pila pròpia.

2.3 El arxiu /proc/PID/maps

El directori */proc* conté arxius amb informació sobre el sistema i els processos. La informació pròpia de cada procés es troba en un directori identificat pel PID del procés. Per a facilitar l'accés d'un procés a la seua pròpia informació existeix un directori ***self***, per a cadascun, que en realitat s'implementa mitjançant un enllaç simbòlic al directori corresponent a aquest procés. Així quan un procés amb PID 2975 accedeix a */proc/self/* en realitat està accedint al directori */proc/2975/.*

Aquesta pràctica treballa amb la informació continguda en l'arxiu *maps* d'un procés.

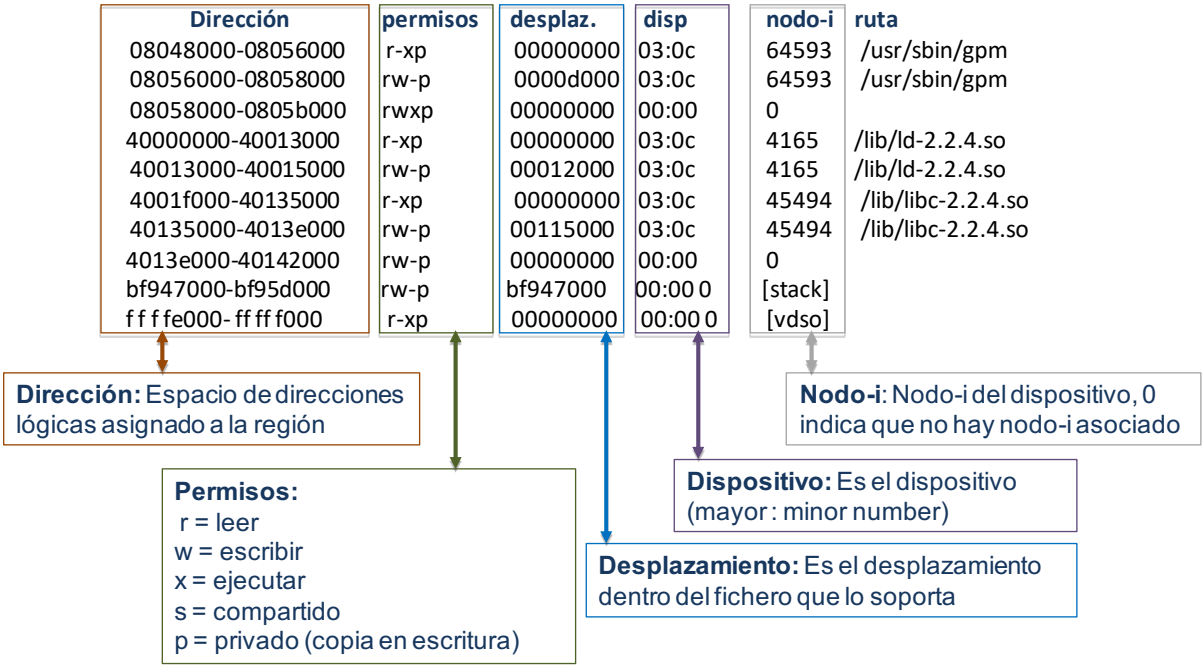

Figura 2. Contingut de les diferents columnes que apareixen al llistar l'arxiu *maps* d'un procés.

En la següent taula se resumen els diferents tipus de regions observats en la figura 2.

Regions de codi	Codi propi del programa Codi de biblioteca dinàmica (.so) Regió VDSO
Regions de dades	Regió de dades recolzada pel fitxer executable Regió de dades no recolzada Pila ([stack])

Tabla-1 Algunes regions que s'observen en un mapa de memòria

La línia del [vdso] de la figura-2, correspon a codi del sistema operatiu. En concreto es tracta del *Virtual Dynamically-linked Shared Object*, una biblioteca compartida que permet al procés realitzar unes poques accions del kernel sense la sobrecàrrega d'una crida a sistema.

Exercici 1: Experimenteu amb /proc/self

Executeu dues vegades seguides l'ordre: $ ls -l /proc/self Expliqueu perquè ixen dos resultats distints.

L'ordre *head -1* mostra la primera línia d'un arxiu de text o de la entrada estàndard. Proveu les següents ordres:
$ head -1 /proc/self/maps
$ cat /proc/self/maps | head -1
Justifiqueu els resultats obtinguts

2.4 Ferramentes i ordres: system() i size

Analitzeu el mapa de memòria d'un procés LINUX visualitzant l'arxiu */proc/PID/maps*, on PID representa l'identificador del procés. Durant l'execució d'un procés utilitzarem l'ordre ***cat*** i l'executarem amb *system()* per a mostrar el contingut del fitxer *maps*. Per a això s'ha de construir un string amb la cadena *"cat /proc/PID/maps"* e invocar la crida *system(cadena)*. Les següents línies de codi s'han d'incloure en aquells punts del programa on es desitge mostrar el mapa de memòria del procés per a analitzar-lo.

```
sprintf(path_maps,"cat /proc/%d/maps",getpid()); //Crea orden per a mostrar
MAPA
flush(stdout);                                    /*vacia buffer*/
system(path_maps);                  /*Llamada al sistema per a executar orden**/
```

Figura 3. Codi per a visualitzar el mapa de memòria d'un procés

L'ordre del ***shell size*** mostra la grandària de cadascuna de les seccions d'un arxiu executable, imprimint en pantalla la grandària de cadascuna de les diferents regions que componen el procés, per a més informació sobre aquesta ordre podeu fer *$man size*.

```
$ size /bin/ls
   text    data     bss     dec     hex filename
 101164    1896    3424  106484   19ff4 /bin/ls
```

3 Mapa amb variables locals, globals i paràmetres de funcions

Com ja sabeu, quan es parla de variables es poden distingir entre:

-**Variables locals**: es troben definides dins d'una, i només poden ser accedides dins de la funció.

-**Variables globals:** es defineixen fora del cos de qualsevol funció, i poden ser accedides per qualsevol funció del fitxer font.

-**Paràmetres d'una funció**: són els valores que reb una funció pel codi que la invoca.

Comprovarem que aquests tres tipus de variables no tenen per què trobar-se ubicades en la mateixa regió del mapa de memòria d'un procés, encara que podrien estar-ho. Treballeu amb l'arxiu "map1.c" proporcionat com a material de pràctiques que contenen el codi mostrat en la figura-4.

```
/* map1.c code*/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

char path_maps[80]; /* Global variable */
char ni_glob[4095];
long i_glob=20;      /* Initialized global variable */

void f(int param)
{
    printf("Address of funcion f parameter: %p\n",&param);
}

int main() {
    long i_loc=20; /* Initialized local variable */
    long ni_loc; /* Non initialized local variable */
    printf("Process PID: %d\n\n", getpid()); /* Shows process PID */

     /**** ADDRESS VISUALIZATION ***/
    printf("main function address: %p \n", main);
    printf("f function address: %p \n", f);
    printf("Initialiazed global variable i_glob address: %p\n", &i_glob);
    printf("Non initialized global variable ni_glob address \n");
    printf(" 1st ni_glob element address: %p\n", &ni_glob[0]);
    printf(" Last ni_glob element address: %p\n", &ni_glob[4095]);
    printf("Initialized local variable i_loc address: %p\n", &i_loc);
    printf("Non initialized local variable ni_loc address: %p\n", &ni_loc);
    f(40);
    printf("\n PROCESS MEMORY MAP /proc/%d/maps \n", getpid());
    // Create command "path_maps" to show memory map
    sprintf(path_maps, "cat /proc/%d/maps",getpid());
    // Empty the buffer
    fflush(stdout);
    // Execute command "path_maps"
    system(path_maps);
    printf ("            ---------------\n\n");
    return 0;
}
```

Figura 4. Codi del arxiu fuente "map1.c"

3.1 Exercici 2: Mapa de memòria d'un procés bàsic

Compileu i executeu el codi de *map1.c*, per a ello haga lo següent:

```
$ gcc –m32 map1.c –o map1
```

```
$ ./map1
```

Cuestión-1: Analitzeu el mapa de memòria mostrat per map1 i consultant tant la figura 2 com el manual del sistema ($man proc), responeu a les següents preguntes:

1. ¿Quin és el rang d'adreces lògiques ocupat pel procés "*map1*"?
2. Identifiqueu la regió del codi i anoteu la seua adreça inicial i final. Feu el mateix amb la regió de pila.
3. Quantes regions de les estudiades podeu identificar en el mapa de memòria del procés "map"? Identifiqueu almenys, una regió de dades recolzada, una regió de dades no recolzada i una de codi de biblioteca
4. ¿Quines regions de "*map1*" es troben suportades per dispositius físics? Justifiqueu la necessitat d'aquest suport.
5. ¿Quin nombre de node-i tenen assignades les regions que no es troben suportades?
6. ¿Quines regions de "*map1*" podrien ser compartides per diversos processos?
7. Identifiqueu en quina regió, de les indicades en la Taula 1, es troba la funció *main*, així com la funció f.

Cuestión-2: Indiqueu en quines regions, de les indicades en la Tabla 1, ha ubicat el sistema cada una de les diferents variables del programa.

Tipus de Variable	**Regió en la qual s'ubica**
Variables globals inicialitzades	

Variables globals no inicialitzades:
`path_maps[80]`
`ni_glob[4095]`	

Variables locals inicialitzades	
Variables locals no inicialitzades	
Paràmetres de funció	

3.2 Exercici3: Mapa d'un procés amb variables locals de gran grandària

Treballeu amb l'arxiu map2.c, proporcionant amb el material de pràctiques, el codi del qual és similar al de l'exercici anterior, figura-4, excepte que conté una nova funció f2 com mostra la figura 5.

```
void f2() {
    int vloc[1024*1024];
    printf("Local variable vloc address: %p\n", vloc);
    printf("\n PROCESS MEMORY MAP (in function f2) \n");
    fflush(stdout);
    system(path_maps);
    printf(" ---------------\n\n");
}
```

Figura 5. Codi de la funció f2, inclosa en map2.c, amb variable local de gran grandària.

En la funció f2, es defineix una variable local de gran grandària: *int vloc[1024*1024]*. El programa *map2.c* imprimeix el mapa de memòria abans de cridar a la funció f2 i dins d'aquesta funció. Compareu ambdos mapes de memòria i observar quins canvis s'han produït relacionant-los amb la variable *vloc*.

Per a això compileu *map2.c* i executeu-lo:

```
$gcc –m32 map2.c –o map2
$ ./map2
```

Analitzeu el mapa de memòria mostrat per map2 i responeu a les següents qüestions:

***Cuestion-3**:* Indiqueu en quina regió del mapa de memòria es troba la variable local *vloc*.

***Cuestion-4**:* Compareu els mapes resultants dels processos *map1 i map2,* i indiqueu si la grandària de les seues respectives piles és diferent. Indiqueu en quin d'ells és major i justifiqueu-lo.

4 Mapa de memòria: Reserva dinàmica de memòria

La reserva dinàmica de memòria és aquella que sol·licita un procés durant la seua execució. En Java, la reserva dinàmica de memòria es fa mitjançant l'operador *new*, mentre que en C es realitza mitjançant la funció *malloc*. En l'anexo d'aquesta pràctica té més informació sobre la funció *malloc* i la reserva dinàmica de memòria en C.

La reserva dinàmica de memòria està suportada per una regió de memòria denominada ***heap***. Es tracta d'una regió de dades no recolzada per arxiu (inicialment s'ompli de zeros), que creix segons es reserva memòria per al procés i decreix quan l'allibera.

4.1 Exercici 4: mapa d'un procés amb reserva dinàmica

Treballeu amb l'arxiu map3.c, que se proporciona amb el material de pràctiques, i que conté les següents declaracions e instruccions:

```
/* map3.c code */
int *vdin;
/*** Dynamic memory allocation ***/
vdin = (int *) malloc(100*sizeof(int));
……
/** Free previously allocated memory**/

free(vdin);
```

Figura 6. Línias de codi de map3.c on es fa la reserva dinàmica de memòria

La línia **`malloc`**`(100*sizeof(int))` fa una reserva dinàmica de memòria per a un array de 100 sencers i es equivalent a la sentència de Java: *vdin = new int[100];*
Compileu *map3.c* i executeu-lo, per a això cal fer el següen:

```
$gcc -m32 map3.c –o map3
$ ./map3
```

En *map3* imprimeix el mapa de memòria abans i després de realitzar la reserva de memòria. Compareu ambdos mapes de memòria i observeu quins canvis s'han produït i la seua relació amb la variable *vdin*.

Qüestió-5*:* Analitzeu els mapes de memòria mostrats per map3 i compareu-los

1. Indiqueu si apareix o desapareix alguna regió. Intenteu justificar a què son deguts els canvis.
2. En el cas de que aparega alguna regió, ¿Quin tipus de regió és? ¿Quins són els seus permisos?
3. Indiqueu en quina regió es troba el contingut del vector *vdin*.

4.2 Exercici 5: Augmentant la reserva dinàmica

Treballeu amb el codi de map3.c i augmenteu la reserva dinàmica de memòria com s'indica en la taula, compileu i executeu cada vegada.

Qüestió-6: Analitzeu què ocorre amb la regió on es troba la variable *vdin*, i intenteu calcular la seua grandària

Reserva dinàmica a realitzar	**Efectes sobre la regió**
`malloc(1000*sizeof(int));`	
`malloc(10000*sizeof(int));`	

`malloc(20000*sizeof(int));`	
`malloc(30000*sizeof(int));`	

5 Mapa de Memòria: Uso de Biblioteques

Les biblioteques són arxius binaris que contenen codi de rutines o subprogrames útils per a l'usuari que poden provenir de diferents entorns com: biblioteques matemàtiques, del sistema operatiu, biblioteca per a efectuar crides al sistema (API del so.) o be crear-les el propi usuari. Sota aquesta perspectiva les biblioteques són una forma senzilla i versàtil de fer modular i reutilitzable el codi.

El mecanisme que fa accessible a les aplicacions aquests mòduls se li diu enllaç i són de dos tipus, enllaç estàtic (no es pot compartir) i enllaç dinàmic (perme compartir). En Windows, arxius de biblioteques dinàmiques posseeixen extensió `.DLL` *(Dynamic Link Library),* mentre que les estàtiques generalment acaben en `.LIB`. En *Unix i Linux*, les biblioteques dinàmiques tenen extensió `.so` (*Shared Object)* i les estàtiques `.a` *(Archive)*.

En Linux una biblioteca puede ser enlazada al codi d'un programa de dos formas diferentes:

• **Enllaç estàtic**: L'executable és autocontingut: inclou tot el codi que necessita l'aplicació, és dir, el codi propi del procés més el de les funcions externes que necessita. En el caso de que es dispose de dues versions de la mateixa biblioteca, estàtica i dinàmica cal utilitzar l'opció `-static` del compilador. L'ordre de compilació tindria la forma:

```
$gcc -m32 program.c -static -lXYZ -o program
```

on `-lXYZ` indica utilitzar la biblioteca `libXYZ`. Per exemple, `-lm` per a utilitzar la biblioteca `libm` (biblioteca matemàtica).

•**Enllaç dinàmic implícit:** La càrrega i el muntatge de la biblioteca es duu a terme en temps d'execució del procés. Es per tant en temps d'execució quan s'ha de resoldre les referències del programa a constants i funciones (símbols) de la biblioteca i la reubicació de regions. Aquesta és l'opció per defecte del compilador (sinó s'especifica el contrari) ja que cerca en primer lloc la versió dinàmica de la biblioteca. L'ordre de compilació és com l'anterior però sense `-static`:

```
$gcc -m32 program.c -lXYZ -o program
```

Como part del procés de muntatge, sempre que s'utilitze almenys una biblioteca dinàmica, s'inclou en l'executable un mòdul de muntatge dinàmic, que s'encarregarà de realitzar en temps d'execució, la carrega i el muntage de les biblioteques dinàmiques usades en el programa. Aquesta opció genera un arxiu executable de menor grandària que l'estàtico.

5.1 Exercici 6: Mapa d'un procés que fa ús de biblioteques

L'arxiu *lib_cos.c* proporcionat amb el material de pràctiques, conté el codi d'un programa que utilitza la funció cosinus la qual es troba dins de la biblioteca matemàtica. La figura-7 correspon a aquest codi, el mapa de memòria del qual s'imprimeix abans i després d'utilitzar la funció cosinus amb la finalitat d'analitzar els canvis que experimenta aquest el mapa del procés.

```
/* lib_cos.c */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#define PI 3.14159265
/* Global variables */
char path_maps[80];
char ni_glob[4095];
long i_glob=20;

void f(int param)
{
    printf("Address of funcion f parameter: %p\n",&param);
}
int main()
{
    /* Local variables */
    long i_loc=20; /* Initialized local variable */
    long ni_loc; /* Non initialized local variable */
    float c;

    printf("Process PID: %d\n\n", getpid()); /* Shows process PID */

    /**** ADDRESS VISUALIZATION ***/
    printf("main function address: %p \n", main);
    printf("f function address: %p \n", f);
    printf("Initialiazed global variable i_glob address: %p\n", &i_glob);
    printf("Non initialized global variable ni_glob address \n");
    printf(" 1st ni_glob element address: %p\n", &ni_glob[0]);
    printf(" Last ni_glob element address: %p\n", &ni_glob[4095]);
    printf("Initialized local variable i_loc address: %p\n", &i_loc);
    printf("Non initialized local variable ni_loc address: %p\n", &ni_loc);
    f(40);

    /** Mathematical operation **/
    c = cos(45*PI/180);
    printf("The mathematical operation result is: %f\n", c);

    printf("\n PROCESS MEMORY MAP /proc/%d/maps \n", getpid());
    sprintf(path_maps,"cat /proc/%d/maps",getpid());
    fflush(stdout);
    system(path_maps);
    printf(" ---------------\n\n");

    return 0;

}
```

Figura 8. Línias de codi de *lib_cos.c*

Compileu el programa, tant amb enllaç de la biblioteca estàtica, como dinàmica implícita i executeu-lo redireccionant l'eixida a un arxiu, per a fer-ho feu:

```
$ gcc –m32 lib_cos.c –static –lm –o biblio_estatica
$ gcc –m32 lib_cos.c –lm –o biblio_dinamica
$ ./biblio_estatica > resul_biblioe
$ ./biblio_dinamica > resul_bibliod
```

Qüestió 7: Visualitzeu els fitxers generats, *resul_biblioe* i *resul_bibliod,* compareu-los i responeu

1. Quines diferències trobeu al comparar els mapes de generats amb els dos tipus d'enllaç
2. Executeu l'ordre "*ls -lh*" en els dos arxius executables i intenteu justificar les diferències de grandària observada
3. Apliqueu el comandament *size* als arxius executables resultants i intenteu justificar les diferències de grandària trobades en les regions.

6 Annex

6.1 Generació d'un executable

En general, una aplicació estarà formada por un conjunt de mòduls que contenen codi font i que han de ser compilats i muntats, com descriu la figura-6. El compilador genera el codi màquina de cada mòdul font i el muntador genera un únic arxiu executable agrupant tots els mòduls i resolent les referències entre ells.

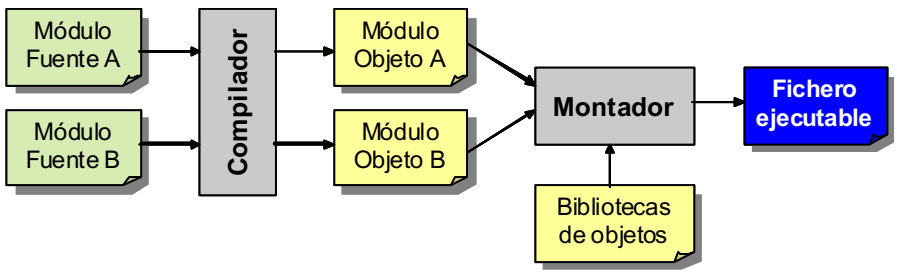

Figura-6.- Etapes en la generació del fitxer executable

6.2 Format de l'arxiu executable

Un arxiu executable Linux està estructurat en una capçalera i un conjunt de seccions (figura-7). La capçalera conté informació de control que permet interpretar el contingut del executable. Cada executable té un conjunt de seccions diferents, però com a mínim apareixen tres: codi, dades amb valor inicial i dades sense inicialitzar. Aquesta última secció apareix en la taula de seccions de la capçalera, però no s'emmagatzema normalment en l'arxiu executable ja que el seu contingut es irrellevant.

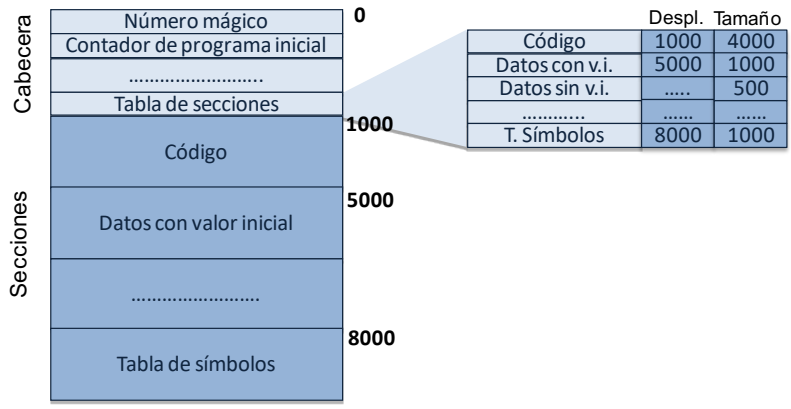

Figura-7.- Format simplificat d'un arxiu executable en Linux (ELF).

Quan se sol·licita l'execució d'un procés, se creen varies regions del mapa a partir de la informació de l'arxiu executable, figura-7. Les regions inicials del mapa del procés es corresponen bàsicament amb les distintes secciones de l'arxiu executable.

6.3 Reserva dinàmica de memòria en llenguatge C

La reserva dinàmica de memòria és aquella que sol·licita un procés durant la seua execució. Això significa que la reserva de memòria es realitza dinàmicament en temps d'execució, no sent necessari haver d'especificar en la declaració de variables la quantitat de memòria que se va a requerir. La reserva de memòria dinàmica afegeix una gran flexibilitat als programes, ja que permet al programador reservar la quantitat de memòria exacta en el precís instant en el que es necessita, sense haver de realitzar una reserva per excés en prevenció de la que es puga arribar a necessitar.

Per a reservar memòria dinàmica en C s'utilitza la funció ***malloc(),*** que reserva una porció contigua de memòria. Està definida com:

*void *malloc(size_t size);*

malloc torna un punter de tipus void *, amb l'adreça a partir de la qual es troba reservada la porció de memòria de grandària *size*. Si no pot reservar eixa quantitat de memòria la funció torna un punter a NULL. Un punter és una variable que conté l'adreça d'un altre objecte.

La funció malloc torna un punter a *void,* o punter genèric. El compilador de C requereix fer una conversió del tipus, mitjançant un casting. Per exemple:

*char *cp;*
*cp = (char *) malloc(1024);*

En este ejemplo se intenta reservar 1024 bytes i la adreça de inicio se almacena en cp. El puntero genérico devuelto por malloc es convertido a un puntero de tipo char mediante el casting "(char *)".

Es usual usar la función sizeof() per a indicar el número de bytes. La función sizeof() puede ser usada per a encontrar el tamaño de cualquier tipo de dato, variable o estructura, por ejemplo:

*int *ip;*
*ip = (int *) malloc(1024 * sizeof(int));*

En este ejemplo se esta intentando reservar memòria per a 1024 enteros (no bytes).

Quan se ha terminado de usar una porción de memòria siempre se deberá liberar usando la función free(). Esta función permite que la memòria liberada este disponible nuevamente quizás per a otra llamada de la función malloc(). Ejemplo

free(ip);

La función free() libera la memòria a la cual el puntero *cp* hace referencia.