Memòria Virtual

Fonaments dels Computadors – Grau en Intel·ligència Artificial – 2022-2023 Q1

Facultat d'Informàtica de Barcelona – Departament d'Arquitectura de Computadors

En aquesta pràctica de laboratori veurem com s'organitza la memòria dels programes que executem. Durant l'execució, és necessari gestionar el codi i les dades dels programes en un entorn que quedi aïllat de la resta de programes per evitar problemes de seguretat. Per això cada procés en execució té el seu propi espai de memòria virtual o espai lògic.

Aquesta sessió continua fent servir els exemples de multiplicació de matrius que vam fer servir a la sessió anterior (S8 – Anàlisi del Rendiment), per entendre millor l'estructura que té l'espai virtual dels processos, l'entorn d'execució, i la manera en què el Sistema Operatiu proporciona la informació al respecte als usuaris.

El programa d'exemple: matmul

El programa de multiplicació de matrius que usem en aquesta sessió té uns petits canvis respecte al de la pràctica S8, en particular:

- A partir del mateix codi font en generem dues versions executables, dinàmica ("matmul") i estàtica ("matmul.static"). La versió dinàmica és l'habitual, enllaçada dinàmicament amb les llibreries (C, ...) com fem habitualment. La versió estàtica està enllaçada estàticament amb les llibreries, de forma que tot el codi està contingut en el fitxer executable, que no depèn de cap llibreria addicional. Farem l'anàlisi dels dos tipus de fitxers executables i els processos que els executen.

- Les matrius A i B es defineixen com a variables globals, ja pre-dimensionades:

```
#define TYPE double
TYPE A[SIZE_N][SIZE_T];
TYPE B[SIZE_T][SIZE_M];
```

  per la matriu C, es declara només un punter a les dades:

```
TYPE * C;
```

  i es demana memòria dinàmicament per C, fent servir el servei de Linux "malloc", des de dins de la funció matrix_allocate:

```
C = matrix_allocate("C", sz_n, sz_m);
```

  A la versió de la sessió S8, les 3 matrius es demanaven de forma dinàmica.

- L'opció "verbose" està desabilitada per defecte. Per la sessió no hauria de ser necessària, però pots definir la macro del preprocessador de C:

```
#define ENABLE_VERBOSE
```

  si la necessites.

- Després d'inicialitzar les matrius, el programa indica quin és el seu "pid" (identificador de procés), en un missatge com:

```
Starting matmul, pid = 18808
```

Durant la sessió, faràs servir el "pid" dels diferents processos que executis, per mirar informació que té Linux sobre ells.

Descarrega el paquet amb aquest exemple, fent servir aquest enllaç:
http://docencia.ac.upc.edu/FIB/GIA/FC/documents/Lab/S9/S9-FC_GIA.tar.gz.
Mira el codi font del "matmul.c", i familiaritzat amb aquesta nova versió. Localitza les diferències que hem comentat en aquesta introducció de la pràctica.

Compilació i execució del programa d'exemple

El "Makefile" que proporcionem amb el paquet d'exemple conté 5 objectius ja preparats. Podeu llistar-los, simplement fent "make":

```
$ make
How to use this Makefile and compile the samples
------------------------------------------------
make matmul.static      # compiles static version of matmul.c
make matmul             # compiles dynamic version of matmul.c
make all                # compiles both versions
make read-symbols       # sorts and displays the symbol table of matmul.static
make read-dyn-symbols   # sorts and displays the dynamic symbol table of matmul
```

Compila les dues versions del programa que podem generar a partir del "matmul.c": "matmul" i "matmul.static".

Executa el programa "matmul.static":

```
$ ./matmul.static      # executarà el producte de matrius amb mides
                       #  1224x96000(A),
                       # 96000x1446(B) i  // Si el programa no s'executa
                       #  1224x1446(C).   // correctament perquè les matrius
Matrix sizes 1224 96000 1446             // són massa grans, pots canviar
init(A): 1.323696 s                      // el SIZE_T per baixar-lo a 9600
...
Starting matmul, pid = 27078
```

Després de la inicialització, mentre s'estigui executant, pots aturar la seva execució (ATENCIÓ, NO significa finalitzar l'execució, sinó posar el procés en "mode pausa" per a què no consumeixi CPU) prement la combinació de tecles CTRL-z. Prement CTRL-z aturem el procés i l'intèrpret de comandes genera un treball ("job"), amb un identificador de treball ("jobid") que podem fer servir per deixar continuar el procés:

```
....
CTRL-z
^Z
[1]+  Stopped        ./matmul    // jobid de la shell = 1
$
            // mentre el procés està aturat,
            // pots veure la tota informació que en té el sistema
....
```

```
                // posteriorment, pots deixar continuar el programa, fent
$ fg            // o bé
$ fg %1         // per referir-se al jobid = 1
./matmul
              // i pots esperar que el programa acabi,
              // o acabar-lo amb CTRL-c
CTRL-c
^C
$ ...
```

L'espai d'adreces virtual (lògic) del procés

En el programa de multiplicació de matrius ("matmul.static"), durant la seva execució, o aturat com hem explicat a dalt, fes els següents exercicis, i escriu les teves respostes en el fitxer "respostes.txt" que creïs per aquesta sessió.

Exercici 1

Executa la versió estàtica de l'exemple de multiplicació de matrius:

```
$ ./matmul.static
...
Starting matmul, pid = 18922
```

Amb el "pid" que ens mostra podem fer operacions sobre el procés que està executant el "matmul.static". En particular, podem examinar la informació que Linux ens permet veure com a usuaris, entre altres:

- Estat del procés, percentatge de processador que està fent servir...
- Propietari i grup, terminal al qual està connectat,
- Temps d'execució (real, d'usuari, de sistema),
- Regions de memòria virtual que fa servir, espai resident a memòria...

Pots obtenir aquesta informació amb la comanda "ps" (process status), i també mirant als fitxers del directori "/proc" que serveixen per donar aquesta informació del procés. Si el "pid" del procés és 18922, trobaràs un directori "/proc/18922/", amb els fitxers amb la informació:

```
$ ls /proc/18922
arch_status   environ    mountinfo    personality   statm
attr          exe        mounts       projid_map    status
...
```

Tots els fitxers d'aquest directori es generen dinàmicament quan els consultem, amb la informació actual en cada moment.

Escriu les teves respostes en el fitxer "respostes.txt":

a) Si et teu procés "matmul.static" té el pid 18922, mira el contingut del fitxer /proc/18922/maps

```
$ cat /proc/18922/maps
```

Aquest fitxer ens permet veure les regions de memòria virtual (espai lògic) del procés que consultem. Copia el llistat en el teu fitxer "respostes.txt" i segueix amb la següent explicació:

El llistat que obtindràs serà semblant a:

```
$ cat /proc/18922/maps
00400000-00401000 r--p 00000000 00:2d 5028292
                          /mnt/c/Users/xavie/GIA-FC/S9/S9-FC_GIA-prep/matmul.static
00401000-0047e000 r-xp 00001000 00:2d 5028292
```

```
                          /mnt/c/Users/xavie/GIA-FC/S9/S9-FC_GIA-prep/matmul.static
0047e000-004a2000 r--p 0007e000 00:2d 5028292
                          /mnt/c/Users/xavie/GIA-FC/S9/S9-FC_GIA-prep/matmul.static
004a3000-004a9000 rw-p 000a2000 00:2d 5028292
                          /mnt/c/Users/xavie/GIA-FC/S9/S9-FC_GIA-prep/matmul.static
004a9000-7a83b000 rw-p 00000000 00:00 0
7b990000-7b9b3000 rw-p 00000000 00:00 0                                  [heap]
7f3f99554000-7f3f9a2d5000 rw-p 00000000 00:00 0
7ffcdb2a8000-7ffcdb2c9000 rw-p 00000000 00:00 0                          [stack]
7ffcdb3d6000-7ffcdb3d9000 r--p 00000000 00:00 0                          [vvar]
7ffcdb3d9000-7ffcdb3da000 r-xp 00000000 00:00 0                          [vdso]
```

Observa que aconseguim veure les regions de memòria que constitueixen el procés. Per cada regió podem saber la següent informació:

`@origen-@final permisos desplaçament-dins-el-fitxer id-disc id-fitxer nom-fitxer`
(mira també la secció del /proc/[pid]/maps a la pàgina de manual del "proc", "$ man proc")

Per cadascuna de les línies amb un "id-disc" i "id-fitxer" diferents de "00:00" i "0", diem que aquesta **regió de memòria està mapejada en un fitxer**, concretament en el fitxer indicat pel "nom-fitxer". Això significa que les dades que tenim en aquesta regió de memòria s'han copiat des del fitxer, començant en el desplaçament indicat per "desplaçament -dins-el-fitxer".

Per cadascuna de les línies que tenen un id-disc igual a "00:00" i un id-fitxer igual a "0", diem que és una **regió anònima**. És un tros de la memòria que no té cap fitxer al darrera (no té nom).

Observa també que les regions abarquen un tros de l'espai virtual que té una mida múltiple de 0x1000 (4096_{10}). Aquesta és la mida de cada "pàgina virtual" de la nostra arquitectura (virtual or logical page). També l'anomenem simplement "pàgina". La gran majoria de les arquitectures actuals tenen una mida de pàgina virtual mínima de 4KB, com aquesta. En molts casos l'arquitectura pot suportar altres mides de pàgina més grans, tant com 2MB, o 1GB.

Per fer-les servir, les pàgines virtuals o lògiques (són sinònims i, de fet, la bibliografia fa servir un o l'altre terme) s'han de mapejar sobre pàgines físiques, les que existeixen a la memòria RAM de l'ordinador. L'espai físic que ocupa una pàgina virtual l'anomenem marc de pàgina, o *frame (mira les últimes transparències de teoria de la primera part d'aquest tema)*. Com que un *frame* s'associa exactament a una pàgina virtual, la mida dels *frames* serà també de 4KB, 2MB, o 1GB.

La mida que el sistema operatiu té de pàgina per defecte es pot obtenir amb la comanda "getconf" (veure també l'exercici 1d, a continuació, per més detalls):

```
$ getconf PAGE_SIZE
4096
```

En el teu llistat, i també mirant la transparència 9 del tema 3: Espai Lògic de Memòria (Part 1), identifica les regions més comunes comentades a classe, i la seva disposició a la memòria lògica del procés. Respon a les següents preguntes:

b) *Primer, determina quin és el rang d'adreces que ocupa cada regió i calcula la seva mida (tant en bytes com en nombre de pàgines). Truc: pots fer servir el "bash" per fer aritmètica entera, per exemple:*

```
$ echo $((0x47e000 - 0x401000))
 512000     # bytes en decimal
$ printf "%x\n" $((0x47e000 - 0x401000))
 7d000      # bytes en hexadecimal
```

c) *Veus un alineament concret en les adreces d'aquestes regions? Quin és? Per què pot ser que totes les regions estan alineades a la mateixa mida? Pista: potser hi ha alguna restricció en la mida de memòria mínima que podem assignar als processos.*

d) *Pots determinar quina és la mida bàsica (la més petita) de les pàgines virtuals en la teva arquitectura? Pista: fes servir la comanda getconf, que en Linux et proporciona la mida de la pàgina virtual:*

```
$ getconf PAGESIZE
```

e) *Determina els permisos que té el procés sobre cadascuna de les regions de memòria. Es poden llegir, escriure i executar? Podeu també buscar informació sobre què significa que una regió sigui privada (p).*

f) *I finalment, fes servir el llistat que has obtingut del sistema (i les respostes a les preguntes anteriors) per explicar quina part del programa conté cada regió, quin nom rep dels indicats a la transparència 9 del Tema3 – Part 1. En el llistat mostrat anteriorment, aquestes són les regions particularment interessants, observeu que n'hi ha que no tenen « nom », tot i que estan realment separades de les adjacents:*

```
00401000-    0047e000 r-xp 00001000              matmul.static
004a3000-    004a9000 rw-p 000a2000              matmul.static
004a9000-    7a83b000 rw-p 00000000
7b990000-    7b9b3000 rw-p 00000000              [heap]
7ffcdb2a8000-7ffcdb2c9000 rw-p 00000000          [stack]
```

Exercici 2

Ara, anem a veure el "matmul", la versió del programa que està enllaçada dinàmicament:

```
$ make matmul
$ ./matmul
...
Starting matmul, pid = 19207
CTRL-z
^Z
...
$ less /proc/19207/maps
```

Escriu les teves respostes en el fitxer "respostes.txt":

a) *Com és el llistat de les regions en aquesta versió? És semblant o molt diferent? Més curt? Més llarg? Per què?*

b) *Observa el llistat, i troba-hi les regions que coincideixen amb les que has llistat a la resposta de l'exercici 1f). Tot i que són les mateixes regions, quines diferències hi veus?*

c) *Ara observa les noves regions que no eren a "matmul.static". Quins objectes (fitxers) mapejen?*

d) *En aquests nous fitxers mapejats en el programa "matmul", pots determinar de quin tipus son les regions, i què deuen contenir? (fes-ho respecte a les seves proteccions, si són de només lectura, de lectura-escriptura, o si són compartides o privades.*

e) *Aquests nous objectes (fitxers) mapejats, entre quines regions que eren al "matmul.static" es mapejen?*

f) Finalment, en aquests programes, què passa amb les adreces que no tenen cap regió mapejada? Seria vàlid llegir o escriure en aquestes adreces?

Els fitxers executables

Ara, anem a determinar si aquesta informació que veiem a l'espai lògic d'adreces dels processos ve dels objectes (fitxers) que tenen mapejats en memòria. Per fer això hi ha diverses eines de GNU que ens permeten examinar el contingut dels fitxers executables (com "matmul.static", "matmul", i també els de sistema, com "/bin/ls" o "/bin/mkdir").

Un fitxer executable conté el codi i les dades del programa, un cop compilat i enllaçat. També conté informació addicional que ens permet manipular el fitxer, i entendre el seu contingut. Entre aquesta informació addicional tenim la taula de símbols del programa, amb les variables globals (A i B a "matmul.c"), les funcions (matrix_init, matmul... a "matmul.c") i la seva organització en regions (dades, codi...), noms dels fitxers/llibreries dels quals aquest programa depèn... Aquesta taula es coneix amb el nom de "**symbol table**", i serveix per tenir apuntats els noms de les funcions i variables dins el mateix fitxer executable. També està present en els fitxers objecte. Serveix per permetre l'enllaçament dels fitxer objecte en executables, o per carregar els executables enllaçats dinàmicament. També la fan servir els programes que ens ajuden a trobar errors en les aplicacions, com els depuradors (el debugger, gdb, per exemple).

Al mirar a la taula de símbols, trobaràs les funcions identificades com a tals amb el tipus FUNC. A la llista de símbols amb aquest atribut, FUNC, hi trobaràs les funcions compilades a partir del teu codi font i les funcions que venen de les llibreries (libc, libm, ... estàtiques o dinàmiques, depenent de si el programa està enllaçat estàticament o dinàmicament).

A la taula també hi trobaràs les variables del programa, identificades amb el tipus OBJECT, i també les que vinguin de les llibreries enllaçades.

I trobaràs altres tipus de símbols, com els que identifiquen les seccions del programa (SECTION) o fitxers relacionats (FILE). També veureu alguns símbols sense tipus específic (NOTYPE). Aquests habitualment són símbols de control.

Com que tenim fitxers enllaçats estàticament i dinàmica, hi ha una taula específica per símbols locals al programa – s'anomenen "static symbols" i són a la "**static symbol table**", o de vegades simplement "**symbol table**" - i una altra taula pels símbols que permetran enllaçar amb les llibreries dinàmiques – s'anomenen "dynamic symbols". La segona taula – la taula de símbols dinàmics, "**dynamic symbol table**" - només és present en els fitxers executables enllaçats dinàmicament. Parlarem més sobre els fitxers executables i el seu contingut en el tema 4 de teoria: "Execució de programes i el seu entorn".

Pels següents exercicis, observa que el Makefile disponible en el paquet d'exemples dóna ja un parell de comandes que podeu fer servir per obtenir la taula de símbols estàtics del "matmul.static", i la taula de símbols dinàmics del "matmul":

```
make read-symbols       # sorts and displays the symbol table of matmul.static
make read-dyn-symbols   # sorts and displays the dynamic symbol table of matmul
```

Aquestes regles fan l'execució de la comanda "readelf" per obtenir els símbols corresponents i els ordenen per adreça.

Exercici 3

Escriu les teves respostes en el fitxer "respostes.txt" per aquesta sessió:

a) *A la pàgina de manual de la comanda "readelf", busca quines són les opcions que et permeten obtenir les taules de símbols estàtica de l'executable que li proporciones com a argument.*

b) *Fes servir la comanda "readelf" per llistar la taula de símbols estàtica del fitxer "matmul.static".*

c) *Determina el tipus dels símbols que veus en el llistat (usa el valor de la columna "Type"):*

c.1) Entre els símbols de tipus FUNC (funcions), indica aquells que venen del codi font del programa "matmul.c". Pista: pots buscar en el llistat de símbols amb la comanda "grep", fent que la sortida del "readelf" passi al "grep" per una "pipe", i filtrar dos cops, primer els símbols FUNC i després el símbol que busques en concret:

```
$ readelf --symbols ./matmul.static | grep " FUNC " \
                | grep "symbol-that-you-are-searching"
```

c.2) Mira el símbol "matmul", compilat de la funció "matmul" a matmul.static:

- *Quina mida té aquesta funció? Quin és el contingut d'aquest símbol?*
- *Si mires al fitxer /proc/<PID>/maps (recorda de substituir el <PID> pel PID que té el procés "matmul.static"), a quina regió pertany aquesta adreça? Correspon al tipus del símbol – funció?*

c.3) Entre els símbols OBJECT, llista aquells que apareixen en el codi font del "matmul.c". Pista: fes com a la pregunta c.1, substituint FUNC per OBJECT.

c.4) Mira els símbols "A", "B" i "C". Quines mides tenen aquests objectes? Es corresponen amb les mides que tenim en les seves declaracions en el programa "matmul.c"?

- *TYPE A[SIZE_N][SIZE_T];*
- *TYPE B[SIZE_T][SIZE_M];*
- *TYPE * C;*

Exercici 4

Escriu les teves respostes al fitxer "respostes.txt" per aquesta sessió:

a) *A la pàgina de manual del "readelf", busca quina és l'opció que se li ha de passar per mostrar la taula de símbols dinàmics (**dynamic symbol table**) que és part d'un fitxer executable (hem comentat sobre la taula de símbols dinàmica en els paràgrafs abans de l'Exercici 3).*

b) *Fes servir la comanda "readelf" per llistar la **dynamic symbol table** del fitxer "matmul". A diferència del llistat de la taula de símbols estàtica (Exercici 3), a la taula de símbols dinàmica la majoria de símbols tenen un atribut addicional associat al seu nom. Per exemple, "puts@GLIBC_2.2.5 (2)". Aquest sufix indica que "matmul" necessita que la llibreria de C – d'on vindrà el símbol "puts" – estigui disponible almenys en la seva versió 2.2.5. Si no hi ha una versió de la llibreria que sigui almenys la 2.2.5, l'aplicació no es podrà executar.*

c) *Determina el tipus dels símbols que veus en el llistat (fes servir el valor que apareix a la columna etiquetada com a "Type"):*

c.1) Entre els símbols FUNC, llista aquells que apareixen en el codi font del "matmul.c". Pista: com abans, pots buscar en el llistat obtingut amb la comanda grep:

```
$ readelf --dyn-syms ./matmul | grep " FUNC " \
          | grep "symbol-that-you-are-searching"
```

c.2) Mira els símbols "printf", "malloc" and "gettimeofday"

- *Per què no estan definits (són UND)?*
- *A quina llibreria estan definitis? (Fes servir la informació respecte la versió de llibreria necessària associada al símbol per trobar la llibreria en la que estan definits)*

c.3) Entre els símbols de tipus OBJECT - variables, llista aquells que apareixen en el codi font del "matmul.c". Pista: mira la pista de l'apartat c.1, substituint el tipus FUNC per OBJECT.

c.4) Quina mida té el símbol "stderr"?

El símbol "stderr", així com els símbols "stdout" i "stdin" si apareixen, corresponen als fitxers en C que es fan servir per treure missatges d'error a la pantalla (stderr), treure missatges normals (stdout) i fer entrada de caràcters als programes (stdin). Estan disponibles a tots els programes per fer entrada/sortida de forma estàndard.

Entrega de la pràctica

Per recollir totes les contribucions per fer l'entrega de la pràctica, pots fer servir la comanda:

```
$ tar czvf session9.tar.gz respostes.txt *.c Makefile
```

I pots anar al RACÓ i pujar el fitxer "session9.tar.gz" a la pràctica S9 del teu grup de laboratori.