Conservatoire National des Arts et Métiers
FOAD Ile De France

La Gestion de la Mémoire Linux

Exercices et Travaux Pratiques
20 Juillet 2020

SOMMAIRE

EXERCICES ET TRAVAUX PRATIQUES

TP 1 : Disponibilité Mémoire

Pour connaitre la taille totale et la disponibilité de la mémoire centrale il faut utiliser la commande **free** :

```
free -m                 #  m pour mega-octet

              total        used        free      shared  buff/cache   available
Mem:           3907         569        2661          19         676        3083
Swap:          3907           0        3907
```

shared *: Utilisée par tmpfs (ramdisk)*
buff/cache *: Cache disque*
available *: Estimation de la mémoire disponible pour démarrer de nouvelles applications sans swapper (cf. chapitre "La Pagination Linux").*

Compléments de cours : L'ordre des octets dans le mot

Il serait logique de ranger les octets des mots mémoire dans l'ordre arithmétique c'est à dire octet fort en tête et octet faible en dernier. Cependant pour des raisons technologiques certains microprocesseurs dont la famille Intel 80x86 rangent les octets dans l'ordre inverse, par exemple pour des mots de 32 bits :

Le nombre hexadécimal 12 34 56 78 se voit transformé en 78 56 34 12

A la relecture du mot mémoire les octets sont remis dans l'ordre, ce qui fait qu'à priori cette inversion est transparente pour le programmeur. Cependant elle n'est plus transparente lors d'un dump mémoire par exemple.

Les microprocesseurs qui rangent les octets du mot dans l'ordre arithmétique sont dits "gros-boutistes" et les microprocesseurs qui les rangent les octets dans l'ordre inverse sont dits "petits-boutistes". Ces termes gros-boutiste (big-endian) et petit-boutiste (little-endian) viennent du roman "Les Voyages de Gulliver" où l'on découvre une tribu qui contrairement aux autres casse les œufs par le petit bout !

Exercice 1 : L'ordre des octets dans le mot

Testez le programme ci-dessous qui alloue un entier non signé en mémoire (32 bits) et ensuite lui affecte la valeur hexadécimale 12345678 puis le relit soit comme un mot soit comme 4 octets consécutifs, grâce à un pointeur **`void`** casté en **`char`**. Cela fait apparaitre le comportement petit-boutiste des plateformes PC :

```
#include <stdio.h>
#include <stdlib.h>
void main () {
  int i;
  unsigned *ptn;
  ptn = (unsigned *) malloc (sizeof(unsigned));
  printf("taille unsigned = %d\n",sizeof(unsigned));
  *ptn = 0x12345678;
  void *ptn2 = ptn;
  printf("mot mémoire = %x\n",*ptn);
  for (i = 1; i <= 4; i++) {
    printf ("octet %d = %x\n",i+1, *((char*)ptn2++));
  }
}
```

TP 2 : Utilisation de la Mémoire

La commande **vmstat** permet de suivre en temps réel l'utilisation de la mémoire :

vmstat -w 1 **#** Afficher toutes les secondes en format large **w**

```
procs ---------------memory-------------- ---swap-- -----io---- -system-- ------cpu-----
 r  b      swpd      free      buff     cache   si   so    bi    bo   in   cs us sy id wa st
 2  0         8   2938616    343704   2351364    0    0    22    15  318  393  5  2 93  0  0
 1  0         8   2938608    343704   2351364    0    0     0     0 4205 10462  4  2 94  0  0
 0  0         8   2933060    343704   2351412    0    0     0   888 4366 10795  5  2 93  0  0
 2  0         8   2934372    343704   2351364    0    0     0     0 4220 9886  4  2 94  0  0
 0  0         8   2933572    343704   2351364    0    0     0     0 4327 9998  4  3 93  0  0
 2  0         8   2932856    343704   2351364    0    0     0     0 4165 9886  4  3 94  0  0
```

si et **so** *représentent le nombre de "swap in" (swap vers mémoire) et "swap out" (mémoire vers swap) durant la seconde considérée (ici 0).*
bi et **bo** *représentent le nombre de "bloc in" (lecture périphérique type bloc) et "bloc out" (écriture périphérique type bloc) durant la seconde considérée.*

TP 3 : La zone de swap

- La zone de swap correspond à une partition indépendante "swap", il est conseillé de mettre en place une zone de swap de une à 2 fois la taille de la mémoire centrale.

- Pour connaitre la taille totale et les disponibilités en zone swap il faut utiliser la commande **free** :

```
free -m
              total        used        free      shared  buff/cache   available
Mem:           3907         569        2661          19         676        3083
Swap:          3907           0        3907
```

- Pour reconfigurer la partition swap il faut utiliser l'utilitaire de partitionnement **"fdisk"**.

TP 4 : Segmentation de Code C

Le fichier virtuel **/proc/<pid>/maps** : permet de voir l'organisation segmentée de l'espace mémoire d'un processus.

Par exemple soit le programme test.c ci-dessous lequel affiche "Hello Everybody" toutes les minutes :

```
#include <stdio.h>
#include <unistd.h>
void main(){
  for(;;) {
    printf("Hello Everybody\n");
    sleep(60);
  }
  return;
}
```

- En le compilant avec l'option **-m64** on obtient une image synthétique de son espace mémoire :

```
linux-fxun:/home/emile # cc -m64 test.c -o test
linux-fxun:/home/emile # size test
  text    data     bss     dec     hex filename
  1251     568       8    1827     723 test
```

- Maintenant lançons le programme : **./test &**

```
[1] 2750
Hello Everybody
```

- Et affichons **/proc/2750/maps :**

```
linux-fxun:/home/emile # more /proc/2750/maps

00400000-00401000 r-xp 00000000 08:03 12344          /home/emile/test
00600000-00601000 r--p 00000000 08:03 12344          /home/emile/test
00601000-00602000 rw-p 00001000 08:03 12344          /home/emile/test
0096b000-0098c000 rw-p 00000000 00:00 0                    [heap]
7fe4f7e49000-7fe4f7ffa000 r-xp 00000000 00:2c 16071  /lib64/libc-2.26.so
7fe4f7ffa000-7fe4f81f9000 ---p 001b1000 00:2c 16071  /lib64/libc-2.26.so
7fe4f81f9000-7fe4f81fd000 r--p 001b0000 00:2c 16071  /lib64/libc-2.26.so
7fe4f81fd000-7fe4f81ff000 rw-p 001b4000 00:2c 16071  /lib64/libc-2.26.so
7fe4f81ff000-7fe4f8203000 rw-p 00000000 00:00 0
7fe4f8203000-7fe4f8228000 r-xp 00000000 00:2c 16063  /lib64/ld-2.26.so
7fe4f8407000-7fe4f8409000 rw-p 00000000 00:00 0
7fe4f8428000-7fe4f8429000 r--p 00025000 00:2c 16063  /lib64/ld-2.26.so
7fe4f8429000-7fe4f842a000 rw-p 00026000 00:2c 16063  /lib64/ld-2.26.so
7fe4f842a000-7fe4f842b000 rw-p 00000000 00:00 0
7ffde7030000-7ffde7051000 rw-p 00000000 00:00 0            [stack]
7ffde71c3000-7ffde71c6000 r--p 00000000 00:00 0            [vvar]
7ffde71c6000-7ffde71c8000 r-xp 00000000 00:00 0            [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0    [vsyscall]
```

Interprétation

- La première ligne **`/home/emile/test`** est le segment texte de l'exécutable de même nom, en effet; on peut le reconnaître grâce à ses permissions (**`r`** = read, **`x`** = execute).
- La seconde ligne **`/home/emile/test`** est la première partie du segment de données en lecture seule (**`r`** = read).
- La troisième ligne **`/home/emile/test`** est la seconde partie du segment de données en lecture écriture (**`r`** = read, **`w`** = write).
- La ligne **`[heap]`** correspond bien évidemment au Tas.
- Les lignes **`/lib64/libc...`** correspondent aux Bibliothèques dynamiques*, elles sont suivies d'une ligne de données en **`rw`**.
- Les lignes **`/lib64/ld...`** correspondent aux éditeurs de liens des bibliothèques dynamiques*, elles sont suivies d'une ligne de données en **`rw`**.
- La ligne **`[stack]`** correspond bien évidemment à la Pile.
- La ligne **`[vvar]`** est associée aux zones **`[vdso]`** ou **`[vsyscall]`** ci dessous.
- La ligne **`[vdso]`** correspond à la "virtual dynamic shared object", correspond à la gestion des appels système.
- La ligne **`[vsyscall]`** correspond à la ligne **`[vdso]`** pour les versions 32 bits (compatibilité).

** Il y a plusieurs segments car les permissions d'accès sont différentes. Ces lignes n'existeraient pas dans le cas d'une compilation avec bibliothèques statiques.*

Exercice 2 : Mémoire Segmentée et Paginée

On considère une mémoire segmentée paginée pour laquelle les cadres mémoire sont de 4Ko. La mémoire centrale compte au total 15 cadres numérotées de 0 à 14.

Dans ce contexte, on considère deux processus A et B.

Le processus A a un espace d'adressage composé de trois segments S1A, S2A et S3A qui sont respectivement de 8 Ko, 12 Ko et 4 Ko, seules les pages 0 et 1 du segment S1A, la page 1 du segment S2A et la page 0 du segment S3A sont chargées en mémoire centrale respectivement dans les cadres 3, 4, 9, 5.

Le processus B a un espace d'adressage composé de deux segments S1B et S2B qui sont respectivement de 16 Ko et 8 Ko, seules les pages 1 et 2 du segment S1B et la page 0 du segment S2B sont chargées en mémoire centrale respectivement dans les cadres 10, 1 et 14.

1) Représentez sur un dessin les structures allouées : Occupation mémoire centrale, table des segments et tables des pages.

2) Dans ce contexte, donnez pour l'adresse linéaire 4098 du processus A son équivalent en adresse virtuelle, puis son adresse réelle correspondante.

- Idem pour l'adresse linéaire 8212 du processus B.

Exercice 3 : Mémoire Paginée

On considère une mémoire paginée mais non segmentée pour laquelle les cadres mémoire sont de 512 octets.

Et on considère le programme "test" ci-dessous lequel fonctionne en adressage linéaire, son adresse linéaire de début (instruction I1) est 1000 :

I1 : Charger le mot d'adresse 6200 dans le registre R0
I2 : Empiler le registre R0
I3 : Appeler une procédure située à l'adresse 5200, le contexte d'appel de la procédure se placera automatiquement dans la pile et on supposera qu'il ne dépassera pas 100 octets et que la procédure ne déclenchera pas elle-même d'appel de pages.
I4 : Comparer la valeur en sommet de pile à la constante 1000
I5 : Effectuer un branchement à l'adresse 5652 si égalité

Précisions :
- Les registres, mots et les adresses sont sur 4 octets et chaque instruction est supposée être codée sur 4 octets.
- La pointeur de pile est situé à l'adresse 8192, il pointe sur le dernier mot de 4 octets entré dans la pile. Attention la pile progresse vers les adresses décroissantes, c'est à dire que le prochain mot entré dans la pile sera stocké en 8188-8191 et le pointeur passera en 8188.

➢ Donner la liste ordonnée des pages appelées par ce programme.

Annexe A : Corrigé des Exercices

- Exercice 1

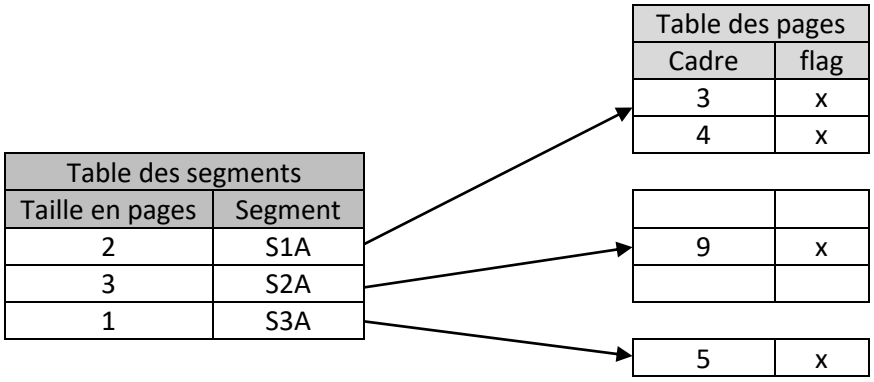
  - Le programme affichera : 12345678 pour le mot entier
  - Puis 78 56 34 12 pour les 4 octets qui le composent.

Nous savons que le compilateur C, proche de l'assembleur lit et écrit les entiers 32 bits comme des mots de 32 bits sans aucune transformation intermédiaire, mais attention certains langages de programmation peuvent rajouter une couche de brassage supplémentaire à l'ordre des octets.

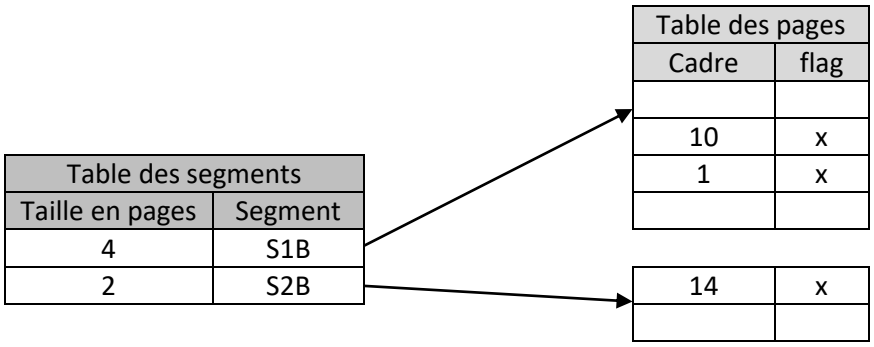
- Exercice 2

1) Structures Allouées

  - Processus A : On a 2 pages pour le segment S1A, 3 pages pour le segment S2A et 1 page pour le segment S3A.



  - Processus B : On a 4 pages pour le segment S1B et 2 pages pour le segment S2B.



- Occupation de la mémoire

Table mémoire centrale		
Cadre	Page	flag
0		x
1	S1B page 2	
2		x
3	S1A page 0	
4	S1A page 1	
5	S3A Page 0	
6		x
7		x
8		x
9	S2A page 1	
10	S1B page 1	
11		x
12		x
13		x
14	S2B page 0	

2) Adresses virtuelles et réelles

- On considère l'adresse linéaire 4098 du processus A.

Cela correspond à la page 1 du premier segment et au décalage de 2 dans cette page (4098 - 4096), donc l'adresse virtuelle est : "S1 / page 1 / décalage 2".

Or la page S1-1 du processus A se trouve dans le cadre n° 4 de la mémoire, donc l'adresse réelle est : "cadre 4 / décalage 2".

- On considère l'adresse linéaire 8212 du processus B.

Cela correspond à la page 2 du premier segment et au décalage de 20 dans cette page, (8212 - 8192) donc l'adresse virtuelle est : "S1 / page 2 / décalage 20".

Or la page S1-2 du processus B se trouve dans le cadre n° 1 de la mémoire, donc l'adresse réelle est : "cadre 1 / décalage 20".

- Exercice 3

Page	Adresse
0	0
1	512
2	1024
3	1536
4	2048
5	2560
6	3072
7	3584
8	4096
9	4608
10	5120
11	5632
12	6144
13	6656
14	7168
15	7760
16	8192

  - P1 : Instruction I1
  - P12 : accès donnée 6200
  - P1 : Instruction I3
  - P15 : accès pile
  - P1 : Instruction I3
  - P10 : accès procédure 5200
  - P15 : accès Pile basculement contextes
  - P1 : Instruction I4
  - P15 : accès pile
  - P1 : Instruction I5
  - P11 seulement si égalité sinon P1