220 Espace mémoire des processus

INF3173
Principes des systèmes d'exploitation

Jean Privat

Université du Québec à Montréal

Hiver 2021

Mémoire des processus

Le SE gère l'organisation de la mémoire

- Le SE est responsable de la cohérence et du nettoyage de la mémoire de l'ordinateur
- La gestion effective de la mémoire dépend du SE et des capacités matérielles

Un processus ne voit que son propre espace mémoire

- Accéder à un espace qui n'est pas le sien est interdit
- Tout est autorisé dans son espace mémoire

$\rightarrow$ Peut corrompre ses propres données en mémoire (bogues)

$\rightarrow$ Mais ne peut corrompre les autres processus

Segments mémoires

4 segments principaux

Vision simpliste à la INF2171

- Code (*text*): le code machine du programme
- Données statiques (initialisées et non initialisées)
- Tas (qui croît vers le bas)
- Pile (qui croît vers le haut)

Extra

- Bibliothèques (code et données)
- Piles supplémentaires (threads)
- Mémoire anonyme et projection de fichiers
- Etc.

Fonctionnement de la pile (rappel INF2171)

On empile

- Des cadres d'exécution fonctionnels (*stackframe*)

Qui contiennent

- Les variables automatiques (variables locales)
- Les paramètres des fonctions
- La place pour les valeurs de retour
- Des valeurs pour la gestion des appels de fonctions (adresse de retour, base de pile, etc.)
- Taille fixée (8Mo pour Linux) mais modifiable par `ulimit (bash)`, `prlimit(1)`, `setrlimit(2)`
- Contient aussi les arguments du programme (`argv`)
- Et les variables d'environnement (`environ(7)`)

Fonctionnement du tas (rappel INF2171)

Allocation (et désallocation) dynamique

- Mémoire réservée quand elle est nécessaire
- Et libérée quand elle ne l'est plus

$\rightarrow$ Contient les données importantes des *vrais* programmes

Gestion programmative

Le programme décide des allocations et des désallocations

Les langages fournissent des mécanismes

- Fonctions bibliothèque. Ex. `malloc(3)` et `free(3)` en C
- Mots clés. Ex. `new` et `delete` en C++
- Ramasse-miettes. Ex. Java

Appels système `brk(2)` et `mmap(2)` : pour demande d'espace mémoire

Chargement des programmes

Contenu des programmes exécutables (binaires)

- Code en langage machine
- Données binaires
- Métadonnées pour chargement et édition de lien (entre autres)

Format des exécutables et bibliothèques dynamiques

- Unix (multi-plateforme): ELF (*Executable and Linking Format*) pour exécutables et `.so`
- Windows: PE (*Portable Executable*) pour `.exe` et `.dll`

Initialisation en mémoire (chargement)

- Provient du fichier binaire (presque tel quel)
  code et données initialisés
- Réservé par le système d'exploitation (et initialisé à 0)
  données non initialisées (BSS), tas et pile (avec argv et environ)

Segments mémoire - Exercice

```
#include <stdlib.h>
#include <unistd.h>
#define K 32*1024
const int L=K;
int T[K];
void foo(int t) {
  int R[K];
  if (t>0) foo(t-1);
}

int main(int argc, char **argv) {
  foo(2);
  int *S = calloc(K, sizeof(int));
  pause();
  return 0;
}
```

Questions

- Quels sont les objets du programme ?
- Quelle est leur taille ?
- Dans quels segments sont-ils alloués ?
- Quelle est leur durée de vie ?
- Quelle est la taille minimum de l'exécutable ?

Qui décide de la vraie organisation ?

- Compilateur C, éditeur de liens, éditeur de lien dynamique
- Peuvent décider d'organiser l'exécutable et la mémoire de nombreuses façons

Droits de la mémoire

Dans les processeurs modernes

- Les zones mémoires ont des droits
- Lecture (r), écriture (w), exécution (x)
  exécution = avoir compteur ordinal dessus
- Configuré par le système d'exploitation
  et par l'appel système `mprotect(2)`

Droits habituels des segments

- Code machine: `r-x`
- Données statiques en lecture seule: `r--`
- Données statiques en lecture écriture: `r-w`
- Tas: `rw-`
- Pile: `rw-`

Voir l'organisation mémoire

Pour voir l'organisation de la mémoire d'un processus

- Commande `pmap(1)`
- Pseudo-fichier `/proc/PID/maps`

→ du point de vue du système d'exploitation

```
00005616b8c7f000      4K r---- orgamem
00005616b8c80000      4K r-x-- orgamem
00005616b8c81000      4K r---- orgamem
00005616b8c82000      4K r---- orgamem
00005616b8c83000      4K rw--- orgamem
00005616b8c84000    128K rw---   [ anon ]
00005616ba1fc000    132K rw---   [ anon ]
00007eff68902000    148K r---- libc-2.31.so
[...]
00007eff68b31000      4K rw--- ld-2.31.so
00007eff68b32000      4K rw---   [ anon ]
00007ffe44262000    396K rw---   [ stack ]
00007ffe44378000     16K r----   [ anon ]
00007ffe4437c000      8K r-x--   [ anon ]
```