600 Gestion de la mémoire

INF3173
Principes des systèmes d'exploitation

Jean Privat

Université du Québec à Montréal

Hiver 2021

Rôles du système d'exploitation

Répartir (allouer) la mémoire

- Pour les processus
- Pour lui-même

$\rightarrow$ Efficacement et sans gaspiller

Contrôler et protéger

- Isoler la mémoire des processus

$\rightarrow$ Chaque processus a l'impression d'être seul

Offrir des services

- Allocation dynamique
- Mémoire partagée
- Configuration de politiques
- `brk(2)`, `mmap(2)`

Objectifs du chapitre

Comprendre

- Comment la mémoire est gérée par le système d'exploitation (et le matériel)
- Les possibilités offertes par la gestion moderne de la mémoire
- Les algorithmes liés à la gestion de la mémoire (utilisables dans d'autres contextes)

Mémoire

La bonne mémoire

- est rapide
- est grande
- est bon marché
- est non volatile,
- mais elle n'existe pas (encore)

En attendant : hiérarchie de mémoire

- Caches processeur
- RAM
- Disques

Nos hypothèses

- Mémoire = grand tableau d'octets
- Accès matériel efficace (CPU $\leftrightarrow$ RAM)
- Les processus vont et viennent

Dans la vraie vie ?

- Processeurs modernes (caches, pagination avancée, etc.)
- Parallélisme et des architectures multiprocesseurs et multicœurs
- Architectures hétérogènes (NUMA, *non uniform memory access*)
- `/proc/meminfo`, `/proc/vmstat`

Problèmes à résoudre

- Adresses explicites dans le code machine d'exécutables
  Comment lancer plusieurs processus ?
- Agrandissement de la mémoire allouée à un processus
  Que faire si on n'a pas la place autour ?
- Clone de processus avec `fork(2)`
  Mais que faire avec les pointeurs existants ?
- Comment partager de la mémoire entre processus ?
- Comment avoir des modes d'accès (écriture,exécution) spécifiques ?

Question

- Pourquoi on voudrait partager la mémoire entre processus ?

Solution : un nouveau niveau d'indirection

- Ne plus permettre aux processus de pointer directement la mémoire
- Les adresses utilisées par les processus (pointeurs, opérandes des instructions machine, etc.) ne sont pas des adresses absolues en RAM

→ On **convertit** les adresses logiques (des processus) En adresses physiques (en RAM)

```
#include "machins.h"
int main(void) {
  pid_t p = fork();
  printf("pid: %d ; adr: %p ; val: %d\n", getpid(), &p, p);
  pause();
  return 0;
}
```

Mémoire physique (ou réelle)

Mémoire physique (ou réelle)

- La RAM (un grand tableau d'octets)
- `free(1)`, `/dev/mem`, `/proc/meminfo`, etc.

Adresse physique (ou réelle)

- Un numéro d'octet dans la RAM

Espace d'adressage physique (ou réel)

- L'ensemble des adresses physiques possibles
- En général la taille du bus d'adresse
- x86-64: actuellement $\approx$ 48bits (256To)

Mémoire logique (ou virtuelle)

Adresse logique (ou virtuelle)

- Les adresses utilisables par le logiciel
- Pointeurs, registres, opérandes, etc.
- Attention: adresse « linéaire » est parfois (abusivement) utilisé

Espace d'adressage logique (ou virtuel)

- L'ensemble des adresses logiques possibles
- En général la taille d'un pointeur ou d'un registre
- x86-64: actuellement $\approx$ 48bits (voire 57bits)
  Les pointeurs sont pourtant 64bits (économie!)

Mémoire logique

- La mémoire associée au logiciel qui s'exécute (processus)
- `/proc/PID/mem`, `pmap(1)` et colonne `VSZ` de `ps(1)`

$\rightarrow$ On y reviendra pour les détails

Unité de gestion mémoire (MMU)

- MMU = *memory management unit*
- Composante matérielle, sur le **microprocesseur**
- Traduit **automatiquement** et **efficacement**
  Adresses logiques $\rightarrow$ adresses physiques
- Les opérandes et pointeurs sont en adresse logique
- Ce qui circule sur le bus d'adresse est en adresse physique

$\rightarrow$ C'est transparent pour le logiciel

Bonus

- Les paramètres de traduction sont configurables (en mode noyau)
- MMU s'occupe aussi de vérifier la légalité des accès mémoire
  Faute CPU si accès à une adresse mémoire logique non valide (selon les paramètres configurés)

Matériel

- Accès direct du matériel (DMA) reste en adresses physiques

Système d'exploitation et processus

Chaque processus

- A des paramètres de traduction mémoire spécifiques
- C'est sa « vue » personnelle de sa mémoire
- Son espace d'adressage logique est **automatiquement** (MMU) associé à des morceaux de mémoire physique (ou à des fautes CPU)

Changements de contexte

- Le système d'exploitation reconfigure le processeur
- Et paramètre la traduction à celle du processus actif

Changements de contexte, en pratique

- Quelques registres à mettre à jour (voire un seul, CR3 chez x86)
- Coût non négligeable sur les CPU modernes (on y reviendra)

Méthodes de traduction

- Chaque architecture matérielle est différente et spécifique
- Plusieurs possibilités en fonction des microprocesseurs
- Certains microprocesseurs offrent ou combinent plusieurs approches

$\rightarrow$ Nombreux détails techniques

Base et limite (historique)

- Technique historique, très simple et très limitée
- Deux registres spéciaux privilégiés : `base` et `limite`
- Traduction : `physique` = `logique` + `base`
- Vérification : `logique` < `limite`

Fonctionnalités

- Chaque processus a son bloc de mémoire qui part de l'adresse 0 pour lui
- Le système d'exploitation peut redimensionner ou déplacer les blocs de façon relativement transparente

Limites

- Mémoire relativement contiguë (sinon gaspillage)
- Pas de partage mémoire entre processus
- Pas de droits fins

Segmentation (historique)

- Technique historique, complexe, et limitée
- Le CPU permet d'avoir plusieurs segments paramétrés indépendamment
- Un segment $\approx$ un bloc base-limite $+$ droits spécifiques
- Résout les limites du base-limite
- N'existe plus en x86-64
  Sauf via deux registres spéciaux `FS` et `GS`

Pagination

- La solution à tous les problèmes ?