621 Mémoire virtuelle avancée

INF3173
Principes des systèmes d'exploitation

Jean Privat

Université du Québec à Montréal

Hiver 2021

Mémoire virtuelle

Aller plus loin ?

- Allouer, initialiser, charger, copier la mémoire efficacement
- Offrir des services aux processus

Optimisation

- Associer pages logiques et physiques paresseusement
- Mise à zéro paresseuse de la mémoire
- Partager les pages à outrance
- Charger les fichiers paresseusement

$\rightarrow$ De façon transparente pour les processus

Services aux processus

- Allocation de mémoire
- Projection de fichiers en mémoire (`mmap`)
- Communication par mémoire partagée
- Configuration de politiques (et d'heuristiques)

Zones mémoires virtuelles des processus

- Alias: région mémoire virtuelle, *virtual memory area*, ou *mapping*

→ Les détails au fur et à mesure

Concept du système d'exploitation

- Ignoré et inconnu du processeur (et MMU)
- Existe pour des raisons de gestion (et d'implémentation)
- Permet de mieux organiser l'espace mémoire des processus

Regroupe des pages virtuelles

- En morceaux cohérents
- Correspondent aux lignes de `pmap(1)` et de `/proc/PID/maps`

→ Ça permet de pas forcement gérer chaque page à part

Autre mémoire consommée des processus

- Table des pages (celle utilisée par le MMU)
- Structures de gestion (table des processus, des descripteurs, etc.)

→ Géré à l'interne par le système d'exploitation

Mémoire résidente vs. virtuelle (le retour)

- Swap + allocation paresseuse + chargement paresseux

Mémoire virtuelle

- Les pages virtuelles dans l'espace d'adressage
- Colonne `VSZ` de `ps(1)`
- Colonne `VIRT` de `top(1)`
- `VmSize`, etc. de `/proc/PID/status`
- Colonne `Size` de `pmap(1)`

Mémoire résidente

- Les pages physiques réellement en RAM
- Colonne `RSS` et `%MEM` de `ps(1)`
- Colonne `RES` et `%MEM` de `top(1)`
- `VmRSS`, etc. de `/proc/PID/status` (Linux)
- Colonne `Rss` de `pmap(1)`
- Note: rss = *resident set size*

Copie sur écriture (COW, *copy-on-write*)

- Faire la copie paresseuse de pages mémoire
- Exemple d'utilité : rendre `fork(2)` très efficace

Stratégie

- Lors d'une demande de copie de page
- On copie rien, on utilise juste deux fois la même page physique
- On ne fait une copie de la page seulement au premier accès en écriture

Mise en œuvre COW

Lors d'une copie, on met a jour la table des pages

- La nouvelle page logique pointe la page physique originale
- On enlève les droits en écriture de l'ancienne page logique et de la nouvelle page logique

$\rightarrow$ Cout: mise à jour de la table des pages

Lors d'un accès en lecture à la page logique

- Tout se passe normalement

$\rightarrow$ Cout: 0

Mise en œuvre COW

Lors d'un accès en écriture à la page logique

- Le MMU lève une faute CPU
- Le système attrape l'interruption, puis
- Copie la page physique dans une nouvelle page physique
- Associe la page logique à la nouvelle page physique
- Positionne les droits en écriture
- Redonne la main au processus
  Qui recommence l'instruction (et réussit cette fois)
- Pas besoin de passer à bloqué: c'est un défaut de page mineur

$\rightarrow$ Cout: copie d'une seule page et mise à jour de la table des pages

Questions

- Pourquoi ne pas passer à bloqué ?
- Comment distinguer un COW d'une vraie page en lecture seule ?

Zone privée vs. partagée

Zone partagée (*shared*)

- Différents processus utilisent les mêmes pages partagées
- Si la zone est écrivable, les modifications sont vues par tous
- Une zone partagée peut être utilisée par un seul processus

Zone privée (*private*)

- Différents processus utilisent des pages privées personnelles et des pages partagées communes (en lecture seule)
- Quand une page privée est écrite : les modifications sont vues que par le processus
- Quand une page commune est écrite : copie sur écriture

Question

- Qu'est-ce que ça change pour les zones en lecture seule ?

Partage des pages sous Linux

- `pmap(1)` et `/prog/PID/maps` affiche `s` pour les zones partagées et - (ou `p`) pour les zones privées
- Colonne `SHR` (*shared*) de `top(1)`:
  somme des taille des pages partagées
- Colonne `Pss` (*proportional set size*) de `pmap(1)`:
  chaque page divisée par le nombre d'utilisateurs
- On y reviendra (détail politiques, API, etc.)