Systèmes avancés

Introduction

Grégoire Pichon

Lyon 1 / Laboratoire de l'Informatique du Parallélisme

MIF18 - M1. 2022/2023

Ce cours est librement inspiré du cours de B. Goglin, avec des extraits du cours de LIFASR5 (L. Gonnord et N. Louvet).

Plan

Contexte de cette UE

Cours précédents

- Systèmes d'exploitation en L2 (LIFASR5)
- Programmation concurrente en L3 (LIFASR7)

Objectifs

- Gestion assez bas niveau des ressources
- Connaître et comprendre les contraintes du bas niveau
- Programmation efficace pour le calcul scientifique
- Idem pour l'embarqué qui a des contraintes particulières

Précédente responsable L. Gonnord, qui a réalisé une partie des supports, notamment ceux sur l'ordonnancement.

Questions posées

Systèmes généralistes

- Comment programmer au niveau système ?
- Comment gérer efficacement les ressources ?

Systèmes embarqués

- Comment gérer le faible volume de ressources ?
- Comment programmer *baremetal* : programmer une puce sans système d'exploitation ?

Lyon 1

Contenu du cours/TP

CM/TD :

- Mémoire virtuelle et pagination
- Virtualisation (J.P. Gelas)
- Conteneurisation (A. Busson)
- Ordonnancement de processus, gestion des entrées/sorties
- Les étapes du boot Linux (J.P. Gelas)

TP :

- Programmation bas niveau sur un système généraliste. Expériences de gestion mémoire, d'ordonnancement
- Programmation bas niveau arduino avec gestion des E/S et des ressources de calcul (pas de système sur ces machines)
- OpenStack, Docker
- Une clef USB de boot Linux.

Évaluation

- Contrôle continu (CC de cours, CR de certains TPs)
- Évaluation finale : examen

Plan

Qu'est-ce qu'un système d'exploitation ? 1/2

Exposer une interface virtuelle indépendante du matériel. Gérer de multiples tâches et utilisateurs.

Quatre grands rôles

- Interface entre applications et matériel (gestion des périphériques)
- Organisation (des disques, de la mémoire, et des processus)
- Sécurité (des données, du matériel)
- Interaction avec le ou les utilisateurs (comptes, droits, installation)

Qu'est-ce qu'un système d'exploitation ? 2/2

Un système d'exploitation doit fournir :

- à l'utilisateur/programmeur une ≪ machine virtuelle ≫, avec
  - une vue unifiée du matériel (mémoire, disque, carte réseau, ...)
  - des objets abstraits (fichiers, répertoires, processus, threads, ...)
- au matériel
  - gestion des ressources (conflit d'accès, ordonnancement),
  - protection contre la mauvaise utilisation,
  - une gestion des évènements (interruptions, exceptions).

Divers

Modes d'exécution

- jeu d'instructions réduit selon les privilèges
- le noyau peut tout faire
- l'utilisateur est très limité, root moins !

Exceptions et interruptions

- Évènements inattendus qui interrompent temporairement l'exécution en cours
- Le processeur saute automatiquement et immédiatement à un traitant
- Suspension du programme en cours et retour au code initial à la fin du traitant

Exceptions : division par 0, segfault

Mode noyau

Pour gérer l'ordonnancement, le noyau doit reprendre la main à chaque passage en mode noyau

- Lors de la gestion des exceptions ; par exemple :
  - division par zéro,
  - accès mémoire non autorisé (*segmentation fault*),
  - instruction interdite (*illegal instruction*).
- Lors d'une interruption matérielle (IRQ, *interrupt request*) :
  - en provenance d'un périphérique
  - du timer (quantum de temps)
- Lors des interruptions logicielles via les appels système
  - à chaque fois que le programmeur fait des entrées/sorties par exemple,
  - cela explique aussi le coût des appels systèmes !

↪ Quand vous écrivez du texte, le système en profite pour travailler !

Appels systèmes

Un appel système est :

- une fonction fournie par le système,
- que tout programme peut utiliser,
- qui est exécutée en mode noyau.

Exception spéciale :

- Adresse du traitant définie par le noyau au boot
- Changement de mode d'exécution
- Passage du mode utilisateur au mode noyau
- Instruction spéciale pour revenir

Les deux changements de contexte associés à un appel système sont coûteux ! Il est bien sur impossible pour un utilisateur d'ajouter de nouveaux appels systèmes !

Processus versus threads

Processus

- Processus lourd
- Permet d'isoler des autres programmes via une mémoire virtuelle distincte
- Coût d'ordonnancement assez conséquent

Thread

- Processus léger
- Partage le tas et autres avec les autres threads d'un même processus
- La pile et les registres sont différents
- Peu coûteux à ordonnancer

Comment réagit le système lorsqu'on change de processus ?

Coût d'ordonnancement

- Changement de contexte
- Changement de la mémoire virtuelle (on vide les caches et le TLB cf. CM mémoire)

Comment réagit le système lorsqu'on change de thread ?

Coût d'ordonnancement

- On doit changer la pile et les valeurs dans les registres
- La mémoire virtuelle reste la même
- Le tas reste le même

Threads en espace utilisateur

- Pour n coeurs, pas besoin de plus de n threads noyaux !
- L'utilisateur doit lui même ordonnancer les threads, les endormir en cas d'E/S
- Difficile, mais les coûts d'ordonnancement seront bien moins importants : on ne passe quasiment pas de temps en espace noyau !

Plan

Conclusion

A retenir (CCF, CCI...)

- Ce qu'il se passe lors d'un appel système
- La différence entre processus et thread