Systèmes d'exploitation, 2ème année

Multi-Threading

Yves Stadler

Université de Lorraine - IUT de Metz

2 avril 2012

Agenda

Plan du cours

- Différence entre Threads et Processus
- Utilisation des threads
- Synchronisation et ordonnancement
- Implémentation du multi-threading avec pthread

Thread

Définition

- Exécution de code en parallèle.
- Partage de données
- Être plus efficace que les processus (changement de contexte coûteux)
- Exploiter les capacités multi-processeurs.

Différence entre Threads et Processus

Points communs

- Permet d’obtenir plusieurs instructions s’exécutant en parallèle.
- Doivent se partager les ressources

Différences

- Les threads sont comme des processus au niveau d’une application
- Les threads partages leur mémoire
- L’ordonnancement des threads peut être contrôlé
- Les changments de contextes sont plus efficaces

Comportement du threads

Partage

- Partage : du tas, des fonctions
- Pas de partage de la pile (mais accès possible ! !)
- Partage du temps alloué au processus entre les threads

Utilisation des piles

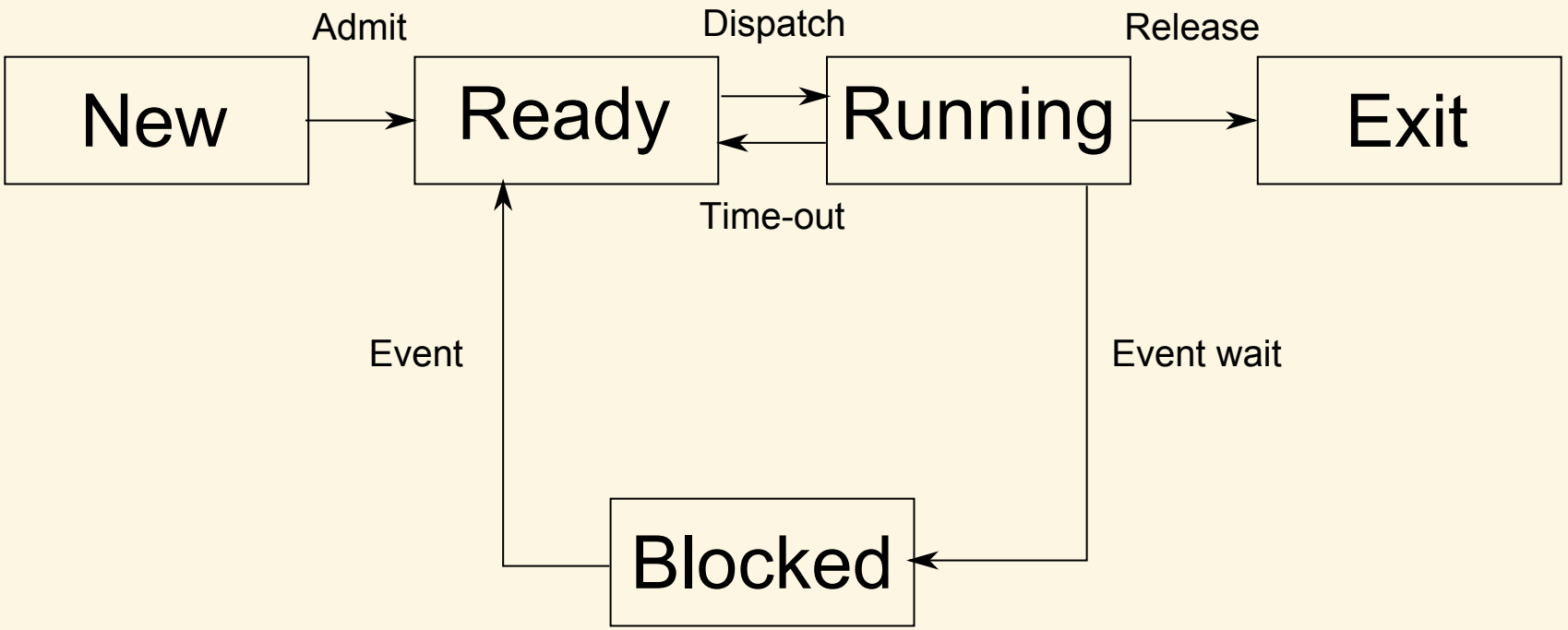
- Une pile applicative
- Une pile système
- Un thread est toujours créer par un autre thread (allocation + appel système)
- Les threads se détruisent eux-même (pas de retour de l’appel, ne peut pas désallouer sa mémoire applicative)

Thread et langages

Bibliothèques

- ADA : type task
- Java : class Thread
- C, C++ : pthread

État des threads



Ordonnancement

Remarque

- Un processus choisi comment répartir son temps entre les threads qui le composent.

Classe d'ordonancement

- Ancienneté (FIFO)
- Priorités (Fixes, variables)
- Quantum de temps durée max.
- Échéances
- Tourniquet
- Priorité et quantum

User thread, Kernel thread

User thread

- Géré par les librairies
- Le système n’en a pas conscience
- S’ordonnange entre eux.
- Ne peuvent pas de multi-threading
- Si un appel bloquant est fait par un thread, les autres sont bloqués.

Kernel thread

- Au moins un thread kernel par processus.
- Reconnus par le systèmes qui gère leur ordonnancement.

Modèles

One to one (1 :1)

- Les threads utilisateurs sont associés à une entité ordonnançable du kernel. (LWP)
- Win32, Linux, Solaris, NetBSD, FreeBSD
- double pile

Many to on (N :1)

- Tous les threads utilisateurs d’un processus sont associés à un seul thread kernel.
- Pas de multi-threading
- On peut toujours éviter les appels systèmes pour éviter les bloquage (quand c’est possible)
- Implémentable sur des machines qui ne connaisse pas le threading.

Many to manuy (M :N)

- Sur certaines machines seulement
- Il faut ordonnancer les différents user-threads sur les kernel threads
- Assez complexe.

Pthreads

Création / Destruction

- type `pthread_t`
- `pthread_create(3)`
- `pthread_t`, des attributs, la fonction de démarrage (pointeur)
- se termine en invoquant `pthread_exit(3)`
- Un programme peut attendre la terminaison d'un trhead avec `pthread_join(3)`

Mutex

- `pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;`
- `pthread_mutex_[un]lock`

Attributs

Type

- `pthread_attr_t`

Ordonnancement

- `SCHED_FIFO`, FIFO, temps-réel
- `SCHED_RR`, round-robin
- `SCHED_OTHER`, ordinaire

Autres paramètres

- Taille de la pille
- Taille de la zone de protection de la pile
- Joinable and detached
- Priorité

Attention

Tread-safety

- Les fonctions qui utilises des variables globales ne sont pas sûres
- Il faut user de mutex dans ce cas.
- De plus en plus de fonctions ont des équivalents "thread safe" (`strtok` et `strtok_r`)

Threads en Java

Dérivation de la classe Thread

- Implémentation de la fonction run.
- Lancement avec la fonction start.

Implémentation de l’interface runnable

- Implémenation de la fonction run
- Lancement avec la fonction start.

Pourquoi deux méthodes

- Pas d’héritage multiple en java
- Applets