Cours de révision de Gabriel Laskar

jeudi 13 juin 2019 14:07

Le cache

Prenons la programmation graphique
Dans un objet on a un vecteur position, vitesse et accélération

Quand on veut mettre à jour la vitesse et l'accélération on va prendre une passe qui va mettre à jour les valeurs

Objet {
vec 3 position
vec3 speed
vec3 accl
rgb color
}

Pour accélérer ce genre de chose, l'idée c'est de changer la manière dont on exploite les données afin de toujours utiliser le cache

ObjectCollection
vec3 position[]
vec3 speed[]
vec3 accl[]
rgb color[]

Scheduling

Différence entre IO Bound et CPU Bound
Globalement on constate qu'on a 2 type de programmes qui existent

L'idée elle est simple: on va avoir des trucs calculatoires d'un côté et interactif de l'autre

Programme calculatoire: transition classique c'est Run -> Ready (consomme le quantum) - CPU Bound
Programme interactif: transition classique c'est Run -> Waiting (ne consomme pas le quantum) - IO Bound

Round robin

Une application qui fait du réseau va se retrouver bêtement limité par l'extérieur ou le réseau en lui-même, elle est donc IO Bound

Un programme interactif c'est un truc qui dort en attendant un event qui va se rendormir après
-> Autant arriver et le réveiller souvent = "Scheduling"

Un programme calculatoire va aller plus vite si il est schedule plus longtemps
Sur le temps pendant lequel il est schedule, il y a un paquet de truc qui son cache et le reste ne l'est pas
-> beaucoup d'accès mémoire, donc plus lent au démarrage

Mais si j'arrive et que je schedule sur un temps plus long on va se retrouver

Pour avoir une impression de vitesse réactive il faut essayer de minimiser le Waiting time (le temps qu'elle prend pour passer ready)

Algo linux de scheduling: CFS (completely fair scheduling)
Edging: Algo pour faire semblant d'implémenter du LRU

File System

Sur un file system on référence toutes les données par un inode

Dans un bloc on a: indirect/double indirect/triple indirect

Le truc c'est qu'il faut regarder la quantité de métadonnées par rapport à la quantité de données

Quand on crée un fichier on a un inode et un bloc

Pour indexer les blocs: BTree (rappel: sa taille est défini à la création)
Idée: on prend un nœud qui contient un entier et un autre nœud qui va avoir l'adresse du bloc

Structure d'un dossier avec des fichiers
struct{
Unsigned inode
Unsigned nodeLen
char filename[]
} [0];

L'opération qui mappe un file system et le met à un endroit spécifique: le syscall mount

Les moyens de communication entre 2 process

- pipe, non bidirectionnel
- named pipe / fifo (le syscall c'est mkfifo) - Raccroché à un inode et se trouve sur le file system
- socket, on est vite embêtés dès qu'on veut faire de l'autorisation (pas de permissions qui sont fait dessus) - rattaché à une IP port
- unix socket / local socket - rattaché à une inode - avec ça on peut faire des permissions et tout ce qu'on peut faire avec un fs. Par exemple on peut transférer des file descriptor (un des mécanismes de base pour faire de la séparation de privilèges)