Systèmes d'exploitation

Demo Threads

Comparé les processus et les threads

- Les processus ont leur propre espace mémoire.
- Les threads partagent la même mémoire.
  - Communication inter-thread plus facile
  - Context switch entre les threads d'un meme processus est plus rapide (Cache friendly)

Qu'est-ce que le parallélisme?

C'est de faire plusieurs choses en meme temp.

Parallelism de données: La meme tache est executé sur un sous ensemble des données

- Exemple: On cherche un element dans un vecteur, ou chaque thread est responsible de chercher dans un sous-ensemble des donnees.
- Exemple: Le meme `shader program` est executer pour plusieurs pixels.'
- SIMD / Vectorization: Parallelism de données au niveau des instructions (sans threads)
- Les GPU's font beaucoup de parallelism de données

Parallelism de taches: Différentes taches sont executées sur le meme ou différentes ensembles de données.

- Exemple: Un thread fait la lecture d'un fichier, un autre thread fait des calcules, un autre thread affiche l'interface...
- Différents processus sont typique un tres bonne exemple de parallelism de taches. Par exemple, mon navigateur web et mon IDE execute en parallel.

Loi d'Amdahl: Parallélisation ne vaut pas toujours la peine!

$$speedup \leq \frac{1}{S + \frac{(1-S)}{N}}$$

Expliquer les threads noyau (kernel threads) et les threads utilisateur (user threads)

Kernel Threads: Threads gérés par le système d'exploitation

- Peut être assigné a un coeur du processeur par le SE
- Au moins un thread par processus

User Threads: Threads gérés par des librairies écrit par l'utilisateur

- Aucune connaissance de ces threads par le SE
- Moins chère que les kernel threads, vu qu'on a pas besoin d'intéragir avec le SE
- Green threads, Java threads...

Models d'implémentations de user threads

- **(1:1):** Un user thread par kernel thread
- **(M:1):** Plusieurs user threads sur un kernel thread
- **(M:N):** Plusieurs user threads partagé par plusieurs kernel threads

Quels sont les interfaces qu'un API de Thread devrait fournir?

Initialization: Devrait fournir un moyen d'initialiser les threads avec une configuration, par exemple la taille de la pile.

pthread_attr_init()

Creation: Devrait fournir un moyen de crée des threads, en lui donnant un point d'entré.

pthread_create(…, int (*start_routine) (void*), void* arg)

Completion: On aimerait être capable d'attendre la completion d'un thread.

pthread_join()

Expliquer ce qui ce passe ici?

```c
int magic_value = 0;

void *runner(void *param) { magic_value = ((char*)param)[0]; }

int main(int argc, char **argv)
{
  pthread_t tid;
  pthread_attr_t attr;

  pthread_attr_init(&attr);
  pthread_create(&tid,&attr,runner,argv[1]);
  pthread_join(tid,NULL);

  printf("magic value = %d\n",magic_value);
  return 0;
}
```

Exemples threads

- Les exemples pour les threads peuvent etre trouver ici: https://github.com/IFT2245/OSC9e/tree/master/ch4