Systèmes avancés

Entrées/Sorties

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

Présentation

Les E/S désignent l'ensemble des transferts de données qui permettent au processeur et à la mémoire de communiquer avec "l'extérieur". Elles proviennent des périphériques.

État des lieux

- Les E/S sont le maillon faible du système : les périphériques sont généralement lents
- Il faut maximiser leur utilisation afin de ne pas réduire l'utilisation du processeur
- Il faut organiser et abstraire la vision des périphériques

Pour chaque périphérique, il existe un module d'E/S qui sert d'interface entre le périphérique et le processeur.

Types d'E/S

Trois types d'E/S

- E/S programmées
- E/S avec interruptions
- DMA

Trois techniques principales pour communiquer entre CPU/mémoire et un périphérique.

E/S programmées (1/2)

Mode de fonctionnement

1. Vérifier si le périphérique est prêt
2. Le contacter (requête)
3. Attendre la fin de la requête en interrogeant les registres de statut du périphérique (polling)
4. Lire/Écrire les données dans le module d'E/S une fois la requête complétée

Avantages / inconvénients

- Simple !
- Accès direct aux registres du périphérique
- Le CPU doit attendre le périphérique : lent

E/S programmées (2/2)

Port I/O

- registres des périphériques sont accédés par des instructions spéciales
- ils sont “mappés” dans un espace de ports
- sert essentiellement pour envoyer des commandes ou lire des réponses

Memory-mapped I/O

- accès naturel aux ressources des périphériques, comme de la mémoire classique
- permet d’utiliser la même interface pour les accès à la mémoire centrale et aux périphériques
- peut servir pour envoyer des commandes, lire des réponses, ou transférer des données

E/S avec interruption

Mode de fonctionnement

- Permet de supprimer les délais d'attente
- Accès au module d'E/S uniquement lors d'interruptions provenant du périphérique

    - Signal d'interruption émis par le périphérique
    - Le processeur termine l'instruction en cours
    - Il regarde s'il doit traiter l'interruption (notion priorité)
    - Sauvegarde du contexte courant
    - Appel de la routine de traitement de l'interruption
    - Restauration du contexte initial

Avantages / inconvénients

- Plus besoin d'attendre le périphérique
- Vitesse de transfert limitée par la vitesse à laquelle le processeur traite ces instructions

Direct Memory Access - DMA

Mode de fonctionnement

- Transfert de données direct entre le périphérique et la mémoire, sans intervention du processeur
- Transfert par blocs
- Transferts configurés par des instructions exécutées par le processeur
  - Périphérique visé
  - Plage de mémoire visée (adresse, taille)
  - Vitesse de transfert...
- Une fois un transfert configuré, le transfert DMA commence automatiquement lors de 1) un signal du périphérique ou 2) une instruction du processeur initiant un transfert

Exemple des trois types d'E/S

On veut lire 10 octets sur une carte réseau et les mettre dans un buffer

Exemple des trois types d'E/S – E/S programmées

```
while (lus < 10){
  if (isAvailable(data)){
    lire(data);
    lus++;
  }
}
```

L'attente est active !

Exemple des trois types d'E/S – E/S avec interruptions

Côté processeur

```
desactiverInterruptions();
configurerInterruption();
activerInterruptions();
faireAutreChose();
```

Code traitant

```
lire(data);
lus++;
if (lus >= 10){
  desactiveInterruption(lecture);
  signalerBufferPlein();
}
```

Exemple des trois types d'E/S – E/S avec DMA

```
desactiverCarteEtDMA();
//transfert DMA toutes les secondes
configurerCarte():
//configure DMA pour mettre en
//mémoire quand transfert déclenché
configurerDMA();
activerCarteEtDMA();
```

Bufférisation

État des lieux

- Les appels systèmes sont coûteux, notamment à cause du passage en mode noyau
- Le principe de localité spatiale est grandement pris en compte
  - Les programmeurs suivent ce principe
  - Ceux qui conçoivent les systèmes utilisent donc le même principe
  - Et aussi ceux qui conçoivent le matériel
- Si on lit une donnée dans la mémoire, le système de cache / prefetching va généralement pré-charger les données suivantes
- Lire une grande zone mémoire est donc une opération bien plus optimisée que lire une seule donnée, donc on essaie de faire des E/S de taille assez volumineuse

Solutions - utilisation de buffers temporaires

- Limiter le nombre d'appels systèmes
- Ne pas faire des E/S de taille trop petite

Poll / select

État des lieux

- Les appels systèmes `read`/`write` par exemple sont bloquants
- On peut les rendre non bloquants (cf. l'attribut `O_NONBLOCK` en TP)
- On peut aussi utiliser des fonctions pour attendre sur plusieurs entrées sorties

L'appel système `select()` permet d'attendre qu'un descripteur de fichier parmi un ensemble soit prêt à effectuer une opération d'entrées-sorties sans bloquer, ou qu'un intervalle de temps se soit écoulé.
On peut aussi utiliser `poll()`

Select

```
int select(int nfds,
fd_set *readfds, fd_set *writefds, fd_set *exceptfds,
struct timeval *timeout);
```

- nfds doit être une borne exclusive des descripteurs de fichiers
- descripteurs pour lire, écrire, exceptions
- notion de timeout

Appel bloquant jusqu'à ce que :

- un des descripteurs dans readfds soit prêt pour une lecture
- un des descripteurs dans writefds soit prêt pour une écriture
- un des descripteurs dans exceptfds soit dans une situation exceptionnelle
- un temps supérieur ou égal à timeout se soit écoulé.

Plan

Conclusion

A retenir (CCF, CCI...)

- Les différents types d'entrées/sorties, leurs avantages et inconvénients