Systèmes centralisés : TD2

Interruptions

Q : Qu'apporte le mécanisme d'interruptions à la gestion et la supervision des E/S par le système d'exploitation ?

Est ce que l'utilisation de ce mécanisme est pertinente dans tous les cas? Pourquoi ?

Q : Du point de vue de l'application et du point de vue des mécanismes mis en jeu, quelles sont les différences entre la réception d'un signal et l'appel d'une procédure ?

Systèmes centralisés : TD2

Interruptions

Q : Qu'apporte le mecanisme d'interruptions à la gestion et la supervision des E/S par le système d'exploitation ? Est ce que l'utilisation de ce mécanisme est pertinente dans tous les cas ? Pourquoi ?

- Scrutation : le processeur doit vérifier régulièrement le périphérique pour suivre la bonne exécution des requêtes d'E/S

- Le mécanisme d'interruption libère le processeur de cette scrutation et permet de découpler les traitements et la réalisation des E/S

- Cependant, le mécanisme d'interruptions a un coût non négligeable en termes de temps d'exécution, puisqu'il induit un aller-retour entre le contexte du traitement interrompu et le contexte du traitant.

- Le mecanisme d'interruptions semble donc adapté aux périphériques lents

et, pour les périphériques rapides, si les échanges de données se font par blocs volumineux (DMA par exemple)

Systèmes centralisés : TD2

Interruptions

Q : Du point de vue de l'application et du point de vue des mécanismes mis en jeu, quelles sont les différences entre la réception d'un signal et l'appel d'une procédure ?

- Le mécanisme de transfert est le même (pile d'exécution)

- La différence essentielle est que :

* l'appel de procédure est synchrone, contrôlé par l'application (qui effectue l'appel),

* tandis que l'émission du signal est contrôlée par l'environnement externe au processus, et peut donc intervenir en tout point de l'exécution du processus.

Systèmes centralisés : TD2

Exercice

Q : Ecrire un code C qui imprime le numéro de tout signal reçu (signal émis depuis le terminal ou depuis un autre processus).

Systèmes centralisés : TD2

Exercice

Q : Ecrire un code C qui imprime le numéro de tout signal reçu (signal émis depuis le terminal ou depuis un autre processus).

Ce code indiquera toutes les 3 secondes qu'il est toujours actif et en attente de signaux.

Au bout de 5 signaux reçus ou 27 secondes il devra s'arrêter.

Le décompte du temps s'appuiera sur la programmation de l'envoi du signal SIGALRM

1. dans un premier temps, on utilisera la fonction alarm pour l'envoi de SIGALRM, et on ne distinguera pas la source des signaux SIGALRM.

Systèmes centralisés : TD2

Exercice

Q : traiter la question précédente en utilisant les timers

Systèmes centralisés : TD2

Exercice

Q : les signaux SIGALRM peuvent résulter d'appels à kill, et non de la programmation de l'horloge.

Compléter le code précédent pour réduire cette possibilité en utilisant les timers

getitimer (ITIMER_REAL, &duree);

Quelle valeur de duree si SIGALRM ?

Systèmes centralisés : TD2

Exercice

Q : les signaux SIGALRM peuvent résulter d'appels a kill, et non de la programmation de l'horloge.

Eliminer cette possibilité en utilisant sigaction

int sigaction (int *signum*, **const struct sigaction **action***, **struct sigaction **oldact*);**

La nouvelle action pour le signal signum est définie par action si action est non nul

Si oldact est non nul, l'ancienne action est sauvegardée dans oldact.

```
struct sigaction {
        void (*sa_handler) (int);                    // un handler ou (exclusif )
        void (*sa_sigaction) (int, siginfo_t *, void *);   // une sigaction
        sigset_t sa_mask;
        int sa_flags;                                // à initialiser absolument
};
```

Systèmes centralisés : API en langage C

sa_flags spécifie un ensemble d'attributs qui modifient le comportement du signal. Il est formé par un OU binaire « | » entre différentss options : comme SA_SIGINFO, SA_NOCLDWAIT (pas de fils zombie avec SIGCHLD)

sa_flags doit être absolument initialisé, car contient n'importe qulle valeur à la déclaration

```
siginfo_t {
    int     si_signo;       /* Numéro de signal         */
    ...
    pid_t   si_pid;         /* PID de l'émetteur        */
    uid_t   si_uid;         /* UID réel de l'émetteur   */
    ... }

void actif (int sig, siginfo_t *info, void *uap) {
   if (info->si_pid == 0 ) {
   /* les SIGALRM programmés sont  envoyés par le scheduler (processus 0) */
```