R3.05 – Programmation Systèmes

Interruptions – Exceptions – Signaux

C. Raïevsky

IUT A Valence
Université Grenoble Alpes
Département Informatique

2023

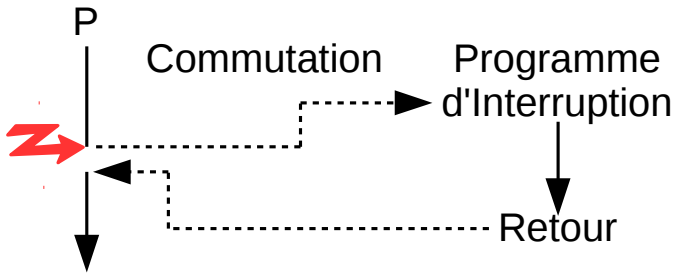
Sources d'interruptions

- Entrées/Sorties
- Erreurs à l'exécution (division par 0)
- Exceptions de sécurité (*segfault*)
- Horloge – *Timer*
- Signaux reçus

Dans ces cas
Le flot d'exécution n'est plus séquentiel

Fonctionnement général

Similaire à un appel de fonction



Différences

- Procédures par défaut à des adresse prédéfinies
- Traitement spécial des interruptions

Caractéristiques des interruptions (1/2)

« Internes »

- Déclenchées par le processus lui-même
- Involontairement (division par 0)
- Volontairement ("exceptions" logicielles, envois de signaux)

« Externes »

- Déclenchées par l'environnement du processus
  - Entrées/Sorties
  - Timers
  - Signaux reçus

Caractéristiques des interruptions (2/2)

Filtrables

- Possibilité de filtrer **certaines** exceptions
- Pour ne pas les recevoir

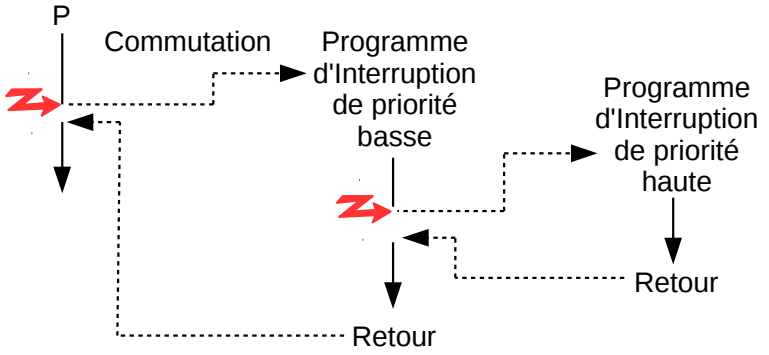
Traitement modifiable

- Possibilité de redéfinir la procédure de traitement

Priorités des interruptions (1/2)

Priorités

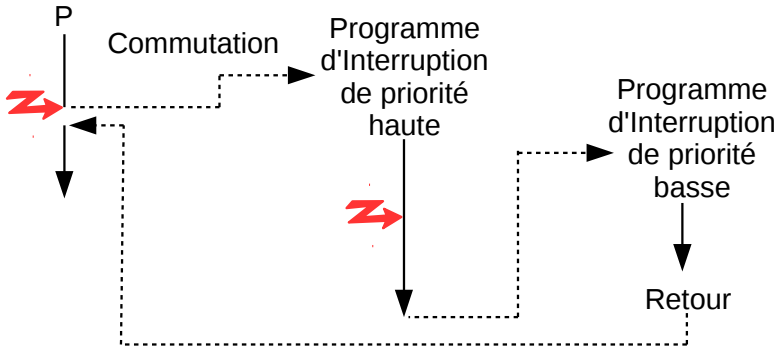
- Certaines interruptions sont plus prioritaires que d'autres
- Une interruption peut arriver durant une procédure de traitement d'interruption



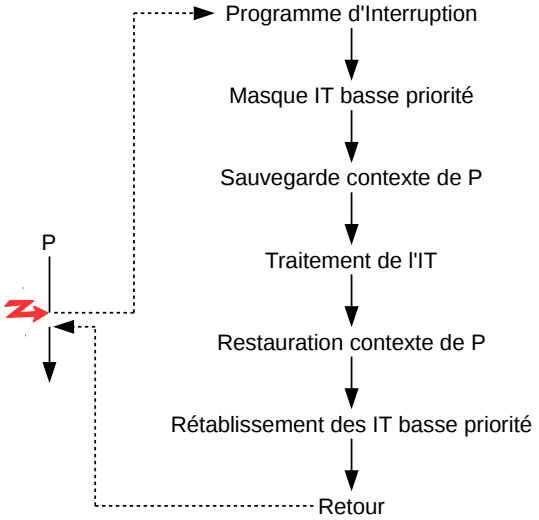
Priorités des interruptions (2/2)

Priorités

- Le traitement d'une interruption peut filtrer les interruption de plus bas niveau



Procédure générale



Appels Système

Très similaire au traitement d'une interruption

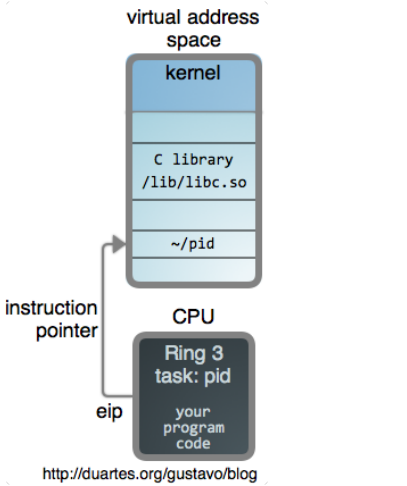
Différent d'un appel de fonction classique

- Le noyau prend la main
- Le processus a plus de droit
- Le cpu change de mode → privilégié (*ring 0*)
- Certaines interruptions sont masquées
- Le processus est potentiellement mis en attente
  - En cas d'accès I/O
  - En cas de faute de page
  - Dans ce cas, un autre processus prend la main

Exemple d'appel système (1/3)

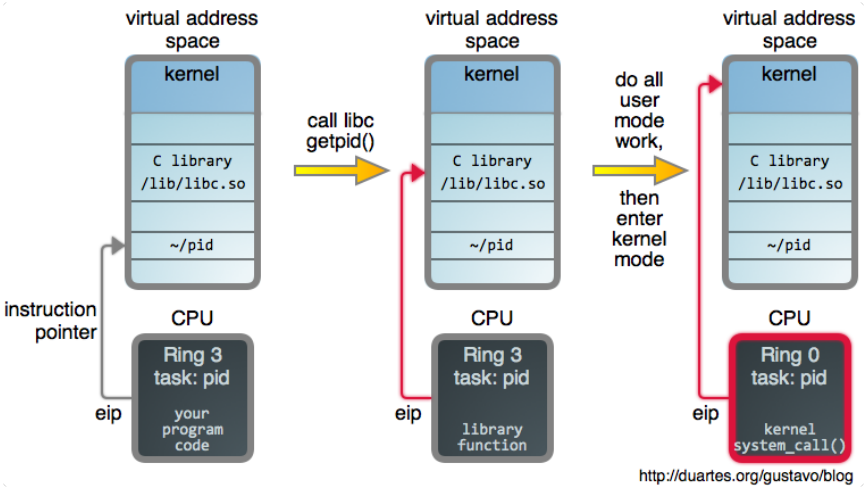
```c
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>

int main()
{
   pid_t p = getpid();
   printf("\%ld\n", p);
}
```



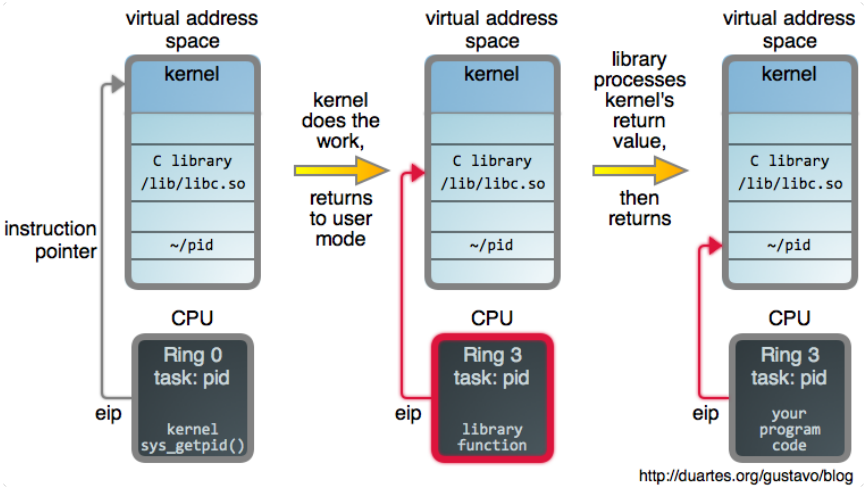
Exemple d'appel système (2/3)

Entrée en mode noyau



Exemple d'appel système (3/3)

Retour au mode utilisateur



Signaux

Mécanisme pour signaler les interruptions aux processus

Les interruptions sont **signalées** aux processus

Principaux signaux

`man 7 signal`

- SIGSEGV : Segmentation fault (Accès mémoire interdit)
- SIGCHLD : Changement de statut d'un enfant (mort, réveil)
- SIGILL : Instruction illégale (Généré par le CPU)
- SIGFPE : Floating point exception (Division par 0)
- SIGTSTP : Pause demandée au clavier (Ctrl-Z)
- SIGINT : Interruption clavier (Ctrl-C)
- SIGUSR1 : User defined
- *SIGKILL* : Mort imminente
- *SIGSTOP* : Pause du processus

Signaux – actions par défaut

Actions par défaut (*disposition*) des rincipaux signaux

`man 7 signal`

- SIGSEGV : Fin du processus et core dump
- SIGCHLD : Ignoré
- SIGILL : Fin du processus et core dump
- SIGFPE : Fin du processus et core dump
- SIGTSTP : Processus mis en pause
- SIGINT : Fin du processus
- SIGUSR1 : Ignoré
- *SIGKILL* : Fin du processus
- *SIGSTOP* : Processus mis en pause

OS, CPU, User, *Non filtrable*

Filtrage des signaux

Il est possible de filtrer (bloquer) les signaux reçus par un processus

`sigprocmask`

- Installation d'un filtre (mask) pour
- Bloquer certains signaux (SIG_BLOCK)
- Débloquer certains signaux (SIG_UNBLOCK)
- Spécifier l'ensemble de signaux bloqués (SIG_SETMASK)

Deux signaux non filtrables :

- SIGKILL
- SIGSTOP

Traitement des signaux

`sigaction`

- Redéfinit l'action à effectuer,
- en fonction du numéro de signal reçu.
- Fournit la possibilité de sauvegarder l'action précédemment associé au signal

Prototype :

```
int sigaction(int signum,
              const struct sigaction *act,
              struct sigaction *oldact);
```

Traitement des signaux – Exemple minimal

Redéfinition du programme d'interruption

```
1
2  static void simpleHandler(int sig) {
3     printf("SIGSEGV␣(%d)␣recu,␣ouch.\n", sig);
4     exit(EXIT_FAILURE);
5  }
6  int main(int argc, char *argv[]) {
7     struct sigaction sa;
8     sa.sa_handler = simpleHandler;
9
10    sigaction(SIGSEGV, &sa, NULL);
11
12    char *buffer;
13    int i = 0;
14    while (1){
15       printf("i:␣%d␣\n", i);
16       buffer[i] = 'a';
17       fflush(stdout);
18       i++;
19    }
20    return EXIT_SUCCESS;
21 }
```