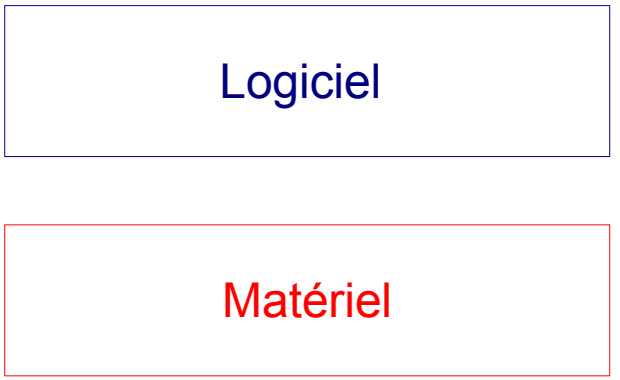
Systèmes d'exploitation

Chapitre 6 : Les entrées sorties

Chargé de cours :
Emery Kouassi Assogba
Tél : 95 22 20 73
Emery.assogba@uac.bj/
emery.assogba@ptgfengineering.com

Entrées sorties

- Interface entre l'ordinateur et le monde extérieur
  - Vision matérielle
    - Comment le CPU interagit avec les dispositifs d'entrées/sorties ?
  - Vision logicielle
    - Comment le système d'exploitation gère-t-il les opérations d'entrées/sorties ?



Entrées/Sorties

Dispositif	I/O	Utilisateur	Débit (KB/sec)
Clavier	I	Humain	0.01
Souris	I	Humain	0.02
Voix	I	Humain	0.02
Scanner	I	Humain	400
audio	O	Humain	0.6
Impr. laser	O	Humain	200
écran	O	Humain	60000
Modem	I/O	Machine	8
Réseau Lan	I/O	Machine	500-6000
floppy	stockage	Machine	100
Disque opt.	Stockage	Machine	1000
Disque dur	Stockage	Machine	2000-10000
Lect. Bandes	Stockage	Machine	2000-10000

- Nombreux dispositifs avec des caractéristiques souvent très différentes

Disques magnétiques

- Disques durs
  - Caractéristiques
    - 1-15 plateaux
    - rotation : 3600-15000RPM
    - 1k-5k pistes par plateau
    - 64 à 200 secteurs par piste
    - secteur : 512 bytes
  - Performances
    - seek time
      - temps pour déplacer la tête de lecture au dessus de la piste choisie
    - minimum seek time
    - average seek time
    - maximum seek time
    - taux de transfert (MB/sec)

Disques magnétiques(2)

- Exemple
  - disque dur
    - vitesse de rotation : 5400 RPM
    - average seek time : 12 msec
    - nombre de secteurs par piste : 100
  - temps d'accès à un secteur de 512 bytes ?
  - Solution
    - accéder à la piste : 12 msec
    - accéder au secteur dans la piste : 0.5 rotation
      - 5400 RPM = 90 RPS -> 0.5 rotation = 5.5 msec
    - 100 secteurs par piste, une piste en 11 msec
      - débit de lecture : 4.6 MBytes/sec
      - lecture d'un secteur de 512 bytes en 11/100=0.11 msec

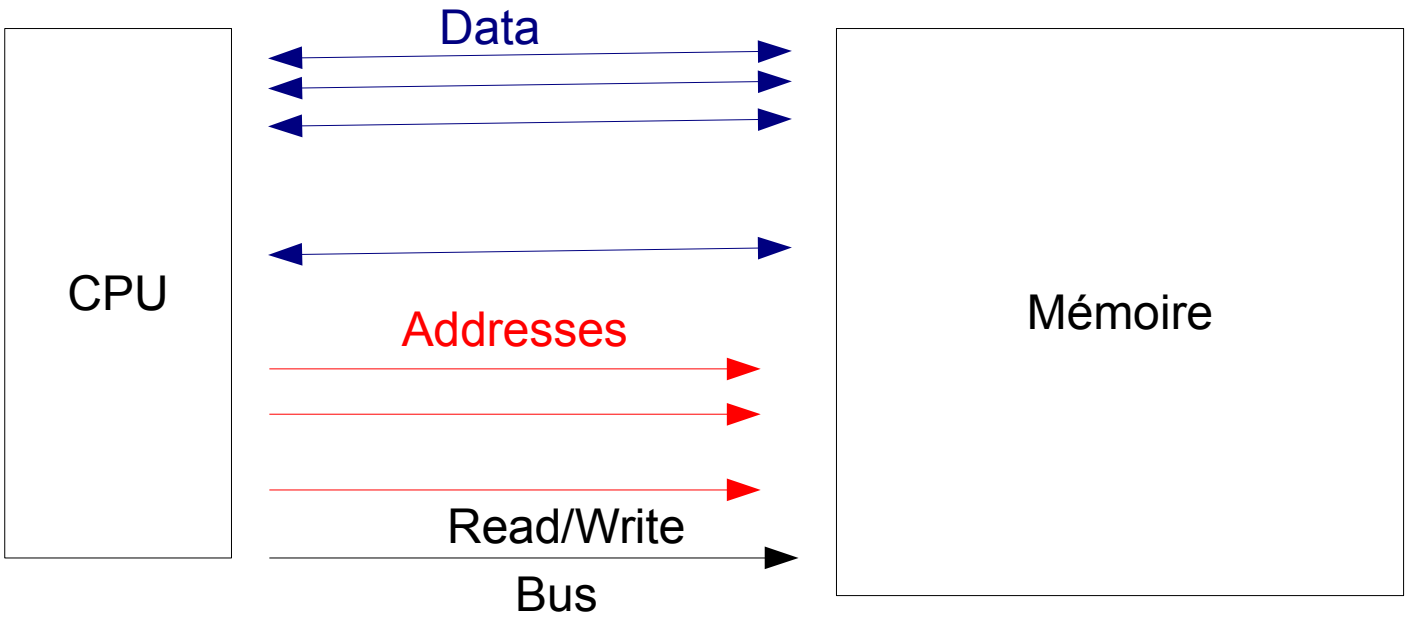
total : 17.61 msec

Carte réseau

- Permet l'échange d'informations entre deux ordinateurs connectés par un réseau local
  - Unité de transfert d'information
    - Paquet : quelques dizaines à quelques milliers d'octets
    - Chaque paquet contient une identification de l'ordinateur source et de l'ordinateur destination
    - le réseau local est responsable de l'acheminement du paquet
  - Débits
    - quelques millions de bits par seconde en bas de gamme
    - environ cent de millions de bits par seconde en milieu de gamme
    - un milliard de bits par seconde et plus en haut de gamme

Carte réseau

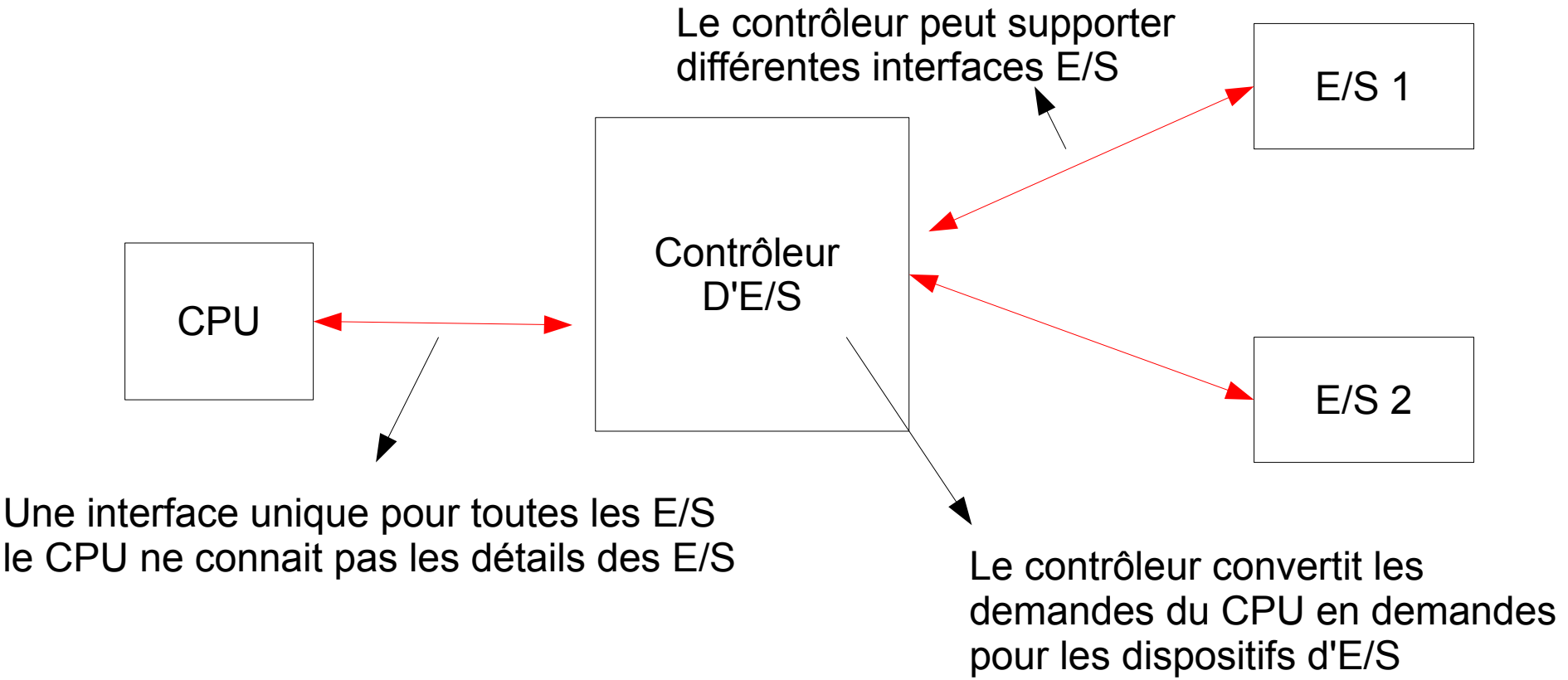
- Opérations supportées
  - Ecriture en mémoire à une adresse spécifiée
  - Lecture en mémoire à une adresse spécifiée



  - Dans certaines architectures, les mêmes fils sont utilisés pour les données et les adresses

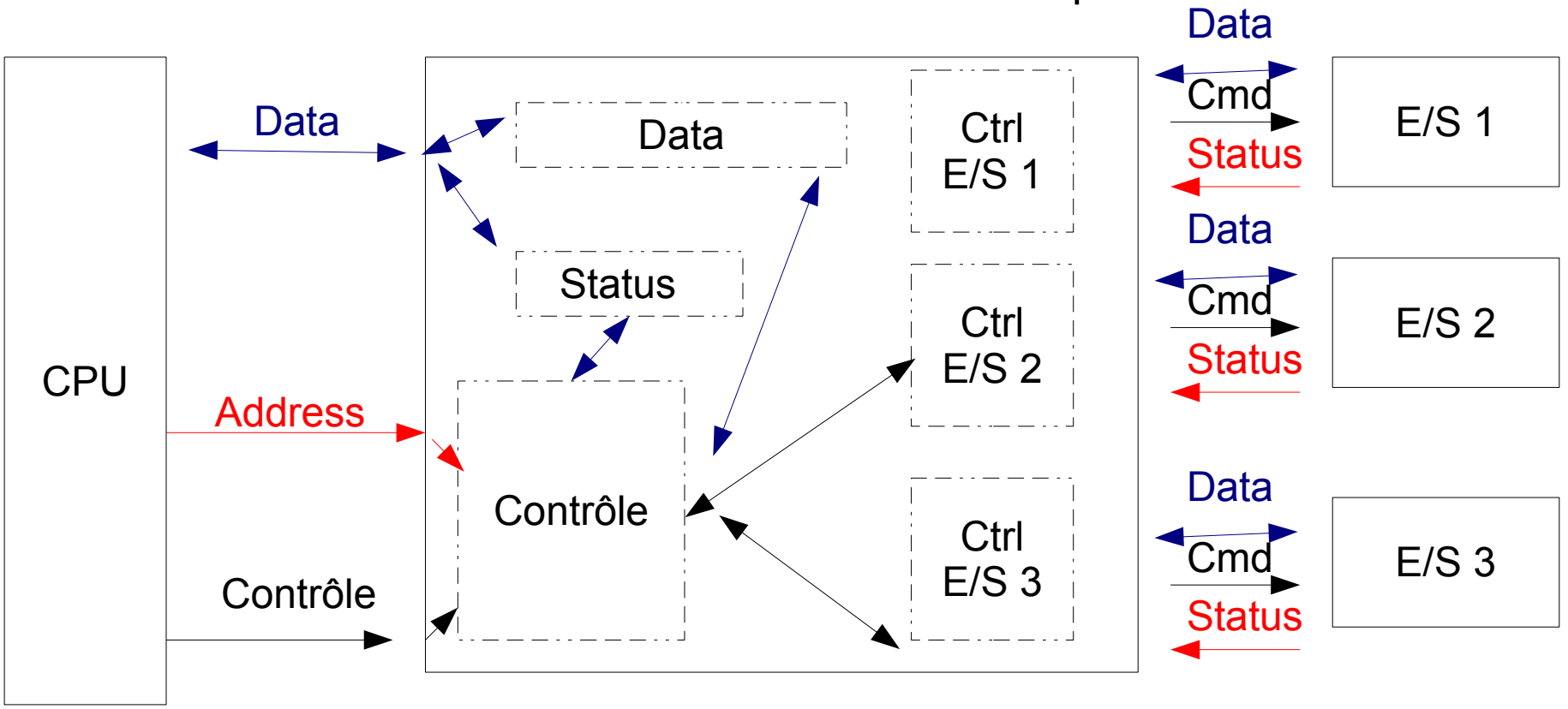
Interactions CPU-E/S

- Problème
  - Nombreux dispositifs avec caractéristiques diverses
  - impossible de supporter tous dispositifs sur le CPU
- Contrôleur d'entrées-sorties

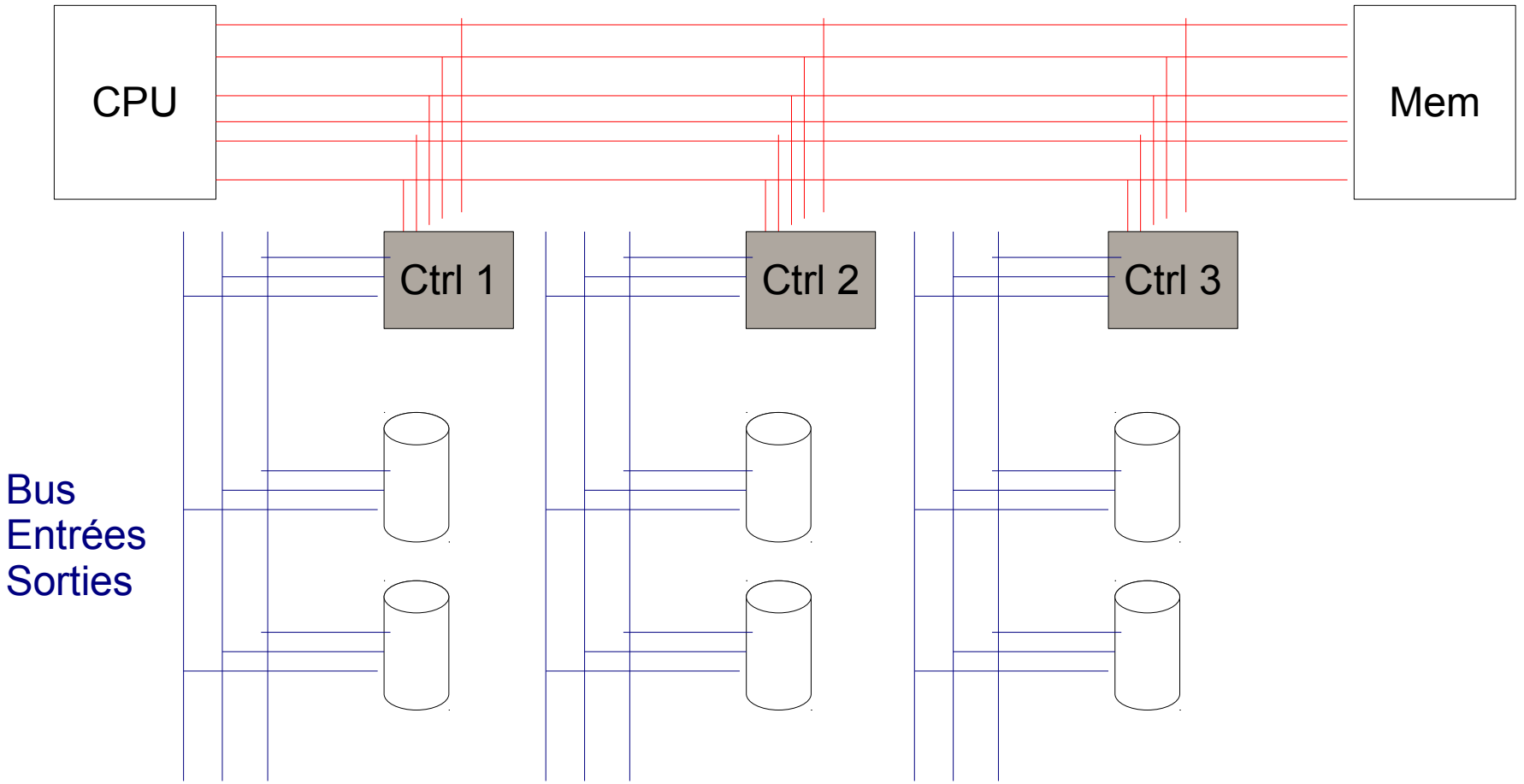


Interactions CPU-Contrôleur E/S

- Opérations à effectuer
  - Ecriture d'une information sur un dispositif d'E/S
  - Vérification de l'état d'un dispositif d'E/S
    - vérifier si une information est arrivée
    - vérifier si le dispositif est prêt à accepter une info à écrire
  - Lecture d'une information venant d'un dispositif d'E/S



Organisation de l'ordinateur



- Un bus rapide entre le processeur et la mémoire sur lequel les contrôleurs sont connectés
- Des bus plus lents pour les dispositifs d'E/S

Accès aux E/S

- Comment accéder aux E/S ?
  - A) memory-mapped
    - chaque dispositif d'I/O est accessible via des adresses particulières de la mémoire RAM
      - adresse spéciale pour écrire une donnée dans le dispositif
      - adresse spéciale pour lire une donnée dans le dispositif
      - adresse spéciale pour lire l'état du dispositif
      - adresse spéciale pour envoyer une commande à un dispositif
    - utilisation des instructions normales d'accès à la mémoire
- B) instructions d'E/S spéciales
  - le CPU contient des instructions pour lire/écrire dans un dispositif
    - chaque instruction prend comme paramètre le numéro du dispositif
      - in reg, port
      - out port, reg

Accès aux E/S (2)

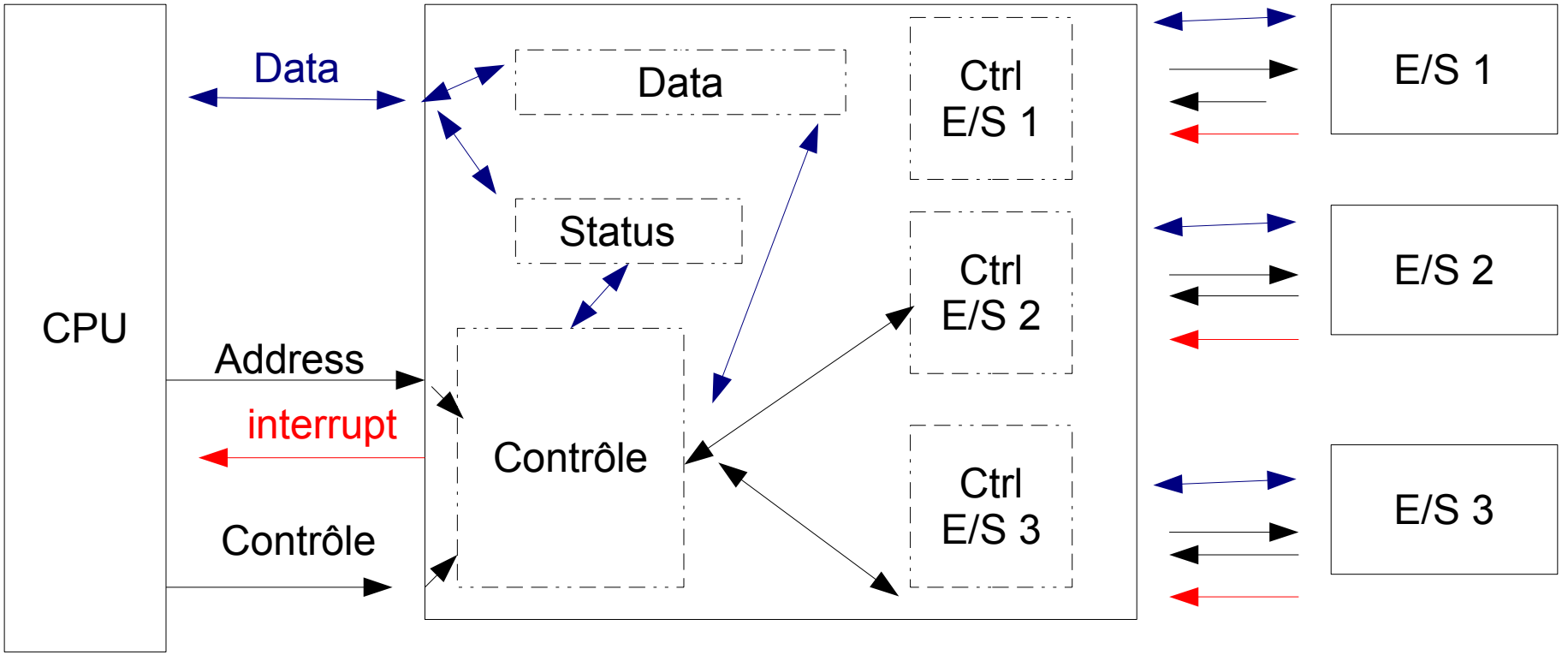
- Programmed I/O

```
int pio (int *ctrl, int *w, int data)
{
    for(;*ctrl!=READY;) {
        /* device not ready */
        schedule();
    }
    /* device is ready */
    *w=data;
    for(;*ctrl!=DONE;) {
        /* data not yet written */
        schedule();
    }
}
```

- Problèmes
  - Le CPU teste l'état du dispositif jusqu'à ce qu'il soit prêt
  - Le CPU est le relais entre E/S et mémoire pour les gros transferts de données

Interactions CPU-Contrôleur E/S

- Amélioration
- permettre au dispositif d'annoncer, via un signal hardware de contrôle lorsqu'il a fini une commande
  - signal de contrôle = interruption



•contrôleur peut rassembler plusieurs lignes d'interruption différentes en un seul signal transmis au CPU

Accès aux E/S avec interruptions

- Exemple
  - Envoi par kernel sur port série, un caractère à la fois

```
write...
/* count bytes dans k_buf */
 enable_interrupts();
  while(*serial_status!=READY)
     {}
  i=0;
  *serial_data=k_buf[0];
```

```
interrupt service routine
{}
if(count>0)
{
        *serial_data=k_buf[i];
         count--;
         i++;
}
else
{ /* copy finished */
}
acknowledge_interrupt();
return_from_interrupt;
```

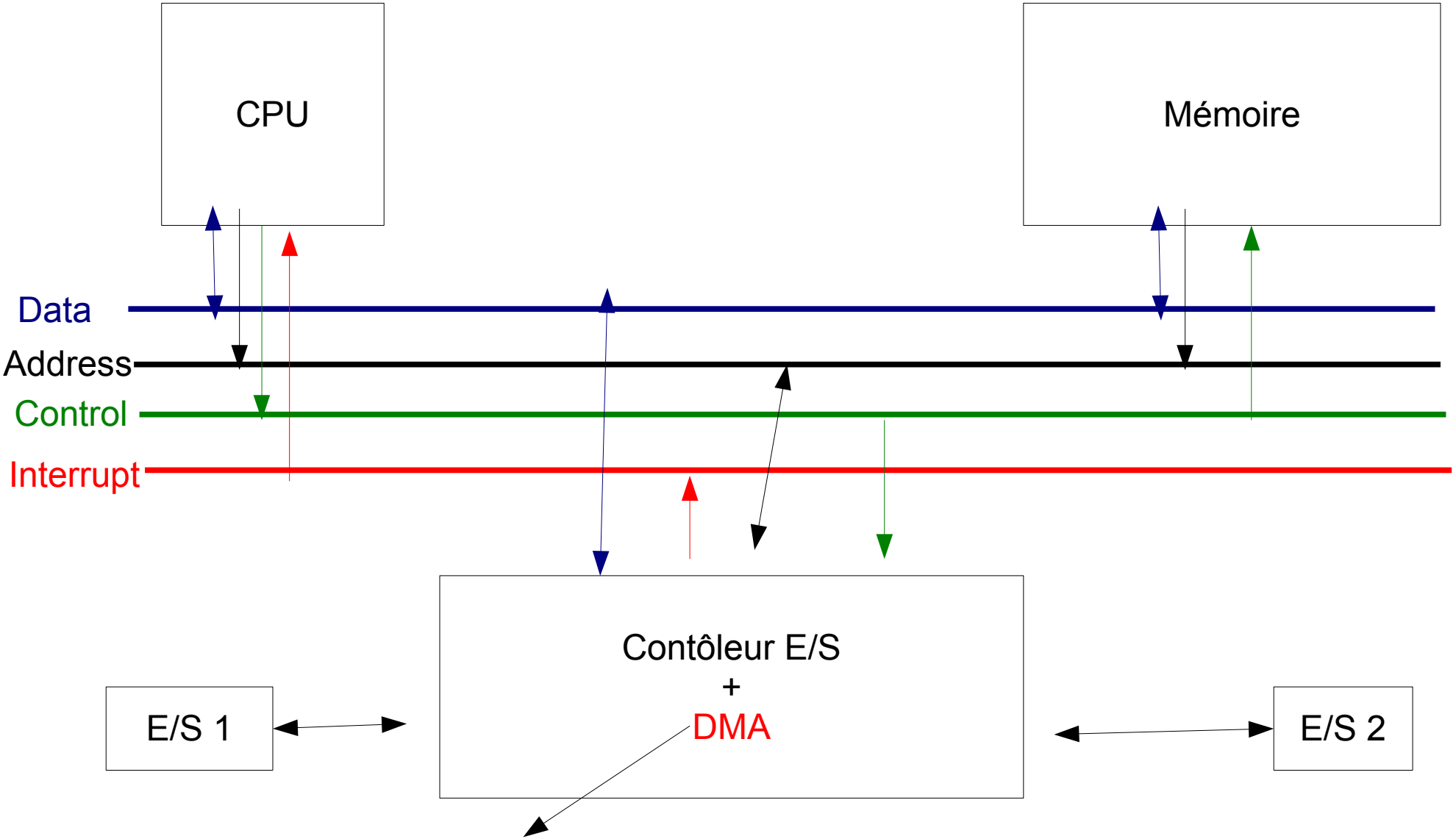
Traitement des interruptions

- Comment le CPU réagit face à une interruption ?
  - A la fin de l'exécution de chaque instruction le CPU vérifie si il y a eu une interruption
  - Si interruption, le kernel
    - sauvegarde l'état des registres
    - prépare contexte pour la routine d'interruption
      - éventuellement avec interaction avec le TLB
    - prépare une pile pour la routine d'interruption
    - envoye un acquit pour l'interruption au contrôleur
    - sauvegarde du contexte dans la table des processus
    - exécution de la routine d'interruption sur base int. vector
      - la routine peut avoir à vérifier l'état de plusieurs dispositifs
    - Choix du processus à exécuter après
    - Réactivation du processus utilisateur
      - peut nécessiter une modification de TLB/MMU, ..

Programmed I/O versus Interrupt-driven I/O

- Programmed I/O
  - avantage
    - simple à implémenter
  - inconvénients
    - le CPU attend la réussite de chaque commande (busy wait)
    - le CPU sert de relais entre E/S et mémoire
- Interrupt driven I/O
  - avantages
    - le CPU ne doit pas attendre la réussite d'un commande
    - pendant l'attente d'une E/S, l'OS peut exécuter un autre processus
  - inconvénients
    - le CPU sert de relais entre E/S et mémoire
    - Pour améliorer les performances, il faudrait éviter que le CPU ne serve de relais pour les copies entre la mémoire et les E/S

DMA : Direct Memory Access

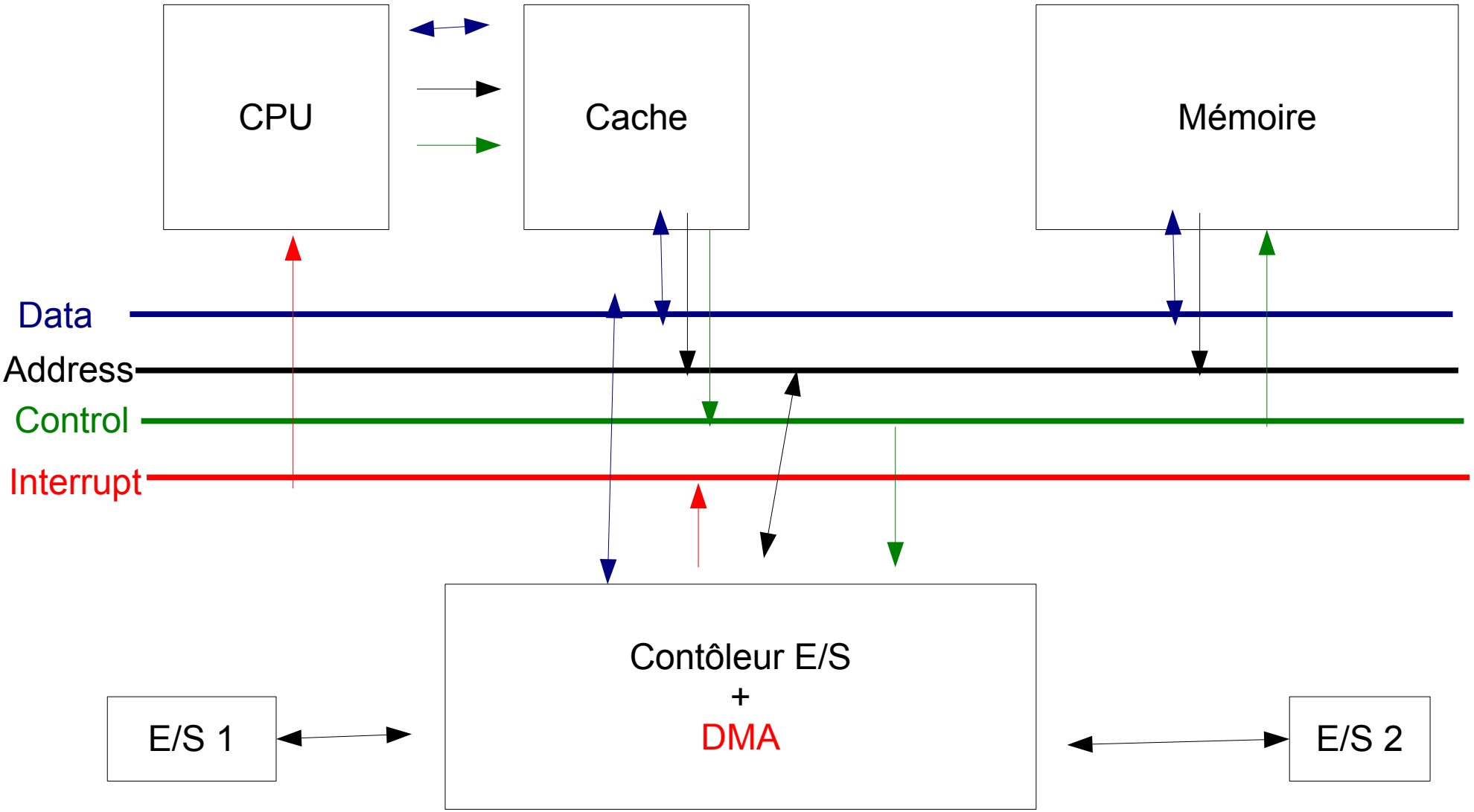


Processeur très simple capable de réaliser
des copies E/S<->Mémoire à la demande du CPU

Accès au DMA

- Comment utiliser le DMA ?
  - Le CPU programme le DMA en lui fournissant
    - L'adresse mémoire source du transfert
      - adresse en RAM ou mappée en mémoire
    - L'opération (lecture/écriture) à réaliser
    - L'adresse destination du transfert
      - adresse en RAM ou mappée en mémoire
    - La quantité d'information à transférer
  - Le DMA réalise le transfert
    - pendant ce temps, le CPU peut faire autre chose
    - cependant, le DMA doit utiliser le bus, comme le CPU
  - Le DMA avertit le CPU de la fin du transfert par une interruption
  - Le CPU traite l'interruption
    - CPU acquitte l'interruption
    - CPU récupère les données transférées par le DMA

DMA et mémoire cache



Comment s'assurer de la cohérence de la cache et de la mémoire ?

DMA et mémoire cache (1)

- Lecture d'une donnée d'E/S vers mémoire
  - Comment éviter incohérence entre cache et mémoire ?
  - Faire passer toutes les lecture à travers la cache
    - évite les problèmes d'incohérence
    - performances faibles car pendant une opération d'E/S, ce n'est en général pas le processus qui a demandé l'opération d'E/S qui occupe le CPU (et la cache)
  - Invalider les lignes de la cache affectée
    - marquer comme invalides les lignes de la cache qui contiennent des zones mémoires utilisées par le DMA
      - peut se faire en hardware avec une cache qui surveille le DMA
      - peut se faire en software par l'OS
        - cache flush pour une partie de la cache
        - cache flush pour la totalité de la cache

DMA et mémoire cache (2)

- Ecriture d'une donnée de mémoire vers E/S
  - Comment éviter incohérence entre cache et mémoire ?
  - Forcer la cache à fonctionner en write through
    - mauvaises performances
  - Cache fonctionnant en write-back
    - avant d'armer le DMA, l'OS doit s'assurer que la cache a bien écrit les données en mémoire
      - forcer une écriture complète des zones modifiées de la cache en mémoire
      - forcer une écriture des zones modifiées et affectées par le DMA en mémoire

Quelques mesures de performance

```
/* start timer */
{
        lseek(fd, 0, 0); /* reset offset at beginning */
        for (i = 16384000; i > 0; i=i-Bytes ) {
                total += write(1, buffer, Bytes);
        }
}
/* read timer */
```

Bytes	Throughput (Mbytes/sec)
16	1.7
256n	25
512	44
1024	70
4096	106
16384	148

- Mesures sur un Pentium utilisant Linux et un disque dur IDE
- Comment expliquer ces résultats ?

Quelques mesures de performance (2)

```
/* start timer */
{
    lseek(fd, 0, 0); /* reset offset at beginning */
    for (i = 16384000; i > 0; i=i-Bytes ) {
        total += write(1, buffer, Bytes);
        err=fsync(fd);
    }
}
/* read timer */
```

Bytes	Throughput (Mbytes/sec)
1024	0.15
4096	0.5
16384	2

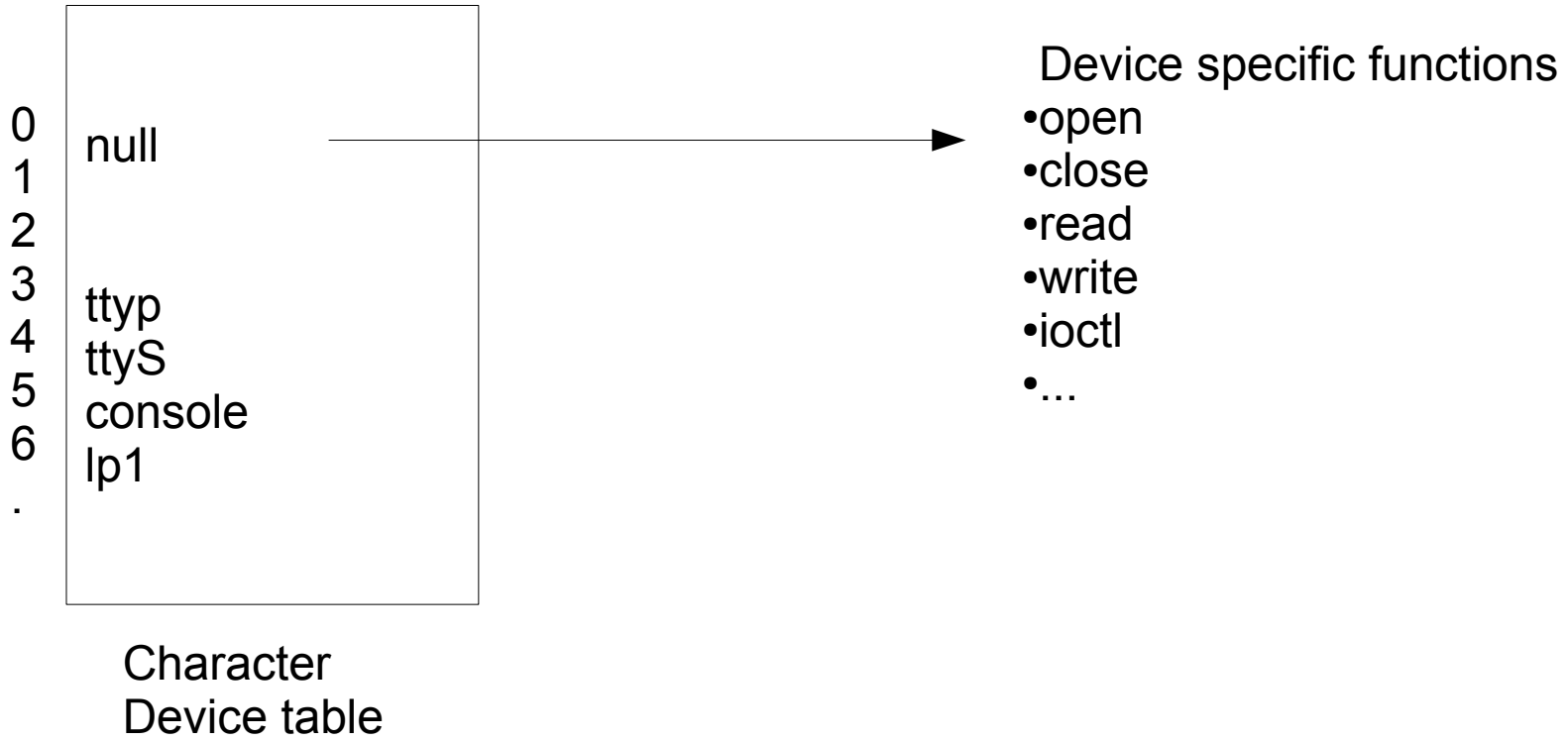
- Mesures sur un Pentium utilisant Linux et un disque dur IDE
- Comment expliquer ces résultats ?

Device drivers

- Device driver
  - Code de l'OS gérant un ou plusieurs dispositifs d'un même type
    - un graveur de CDROM
    - quatre disques IDE device driver
  - peut être compilé dans l'OS ou est chargé dynamiquement au démarrage
- Types de device drivers
  - Block device driver
    - il est possible d'accéder aléatoirement à des blocs particuliers sur le dispositifs
  - Character device driver
    - il est possible d'échanger (envoi/réception) un flux d'octets avec le dispositif

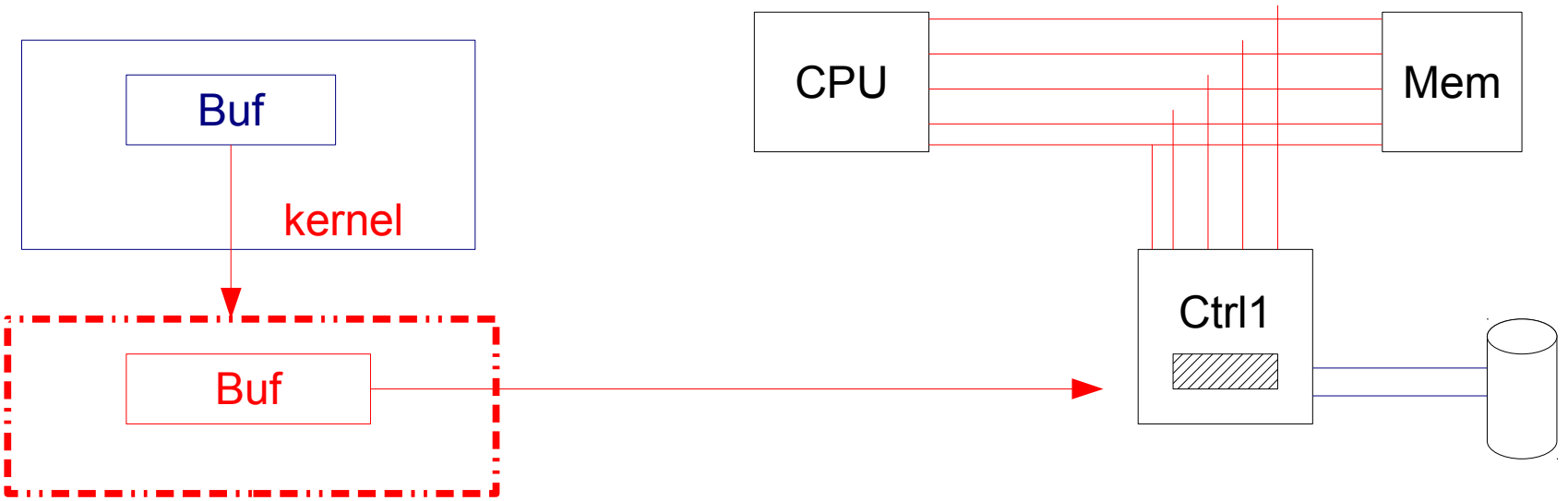
Rôle des device drivers

- Fournir interface standard aux utilisateurs
  - /dev/null, /dev/lp0, /dev/hda1, ...
- Comment utiliser ces devices ?



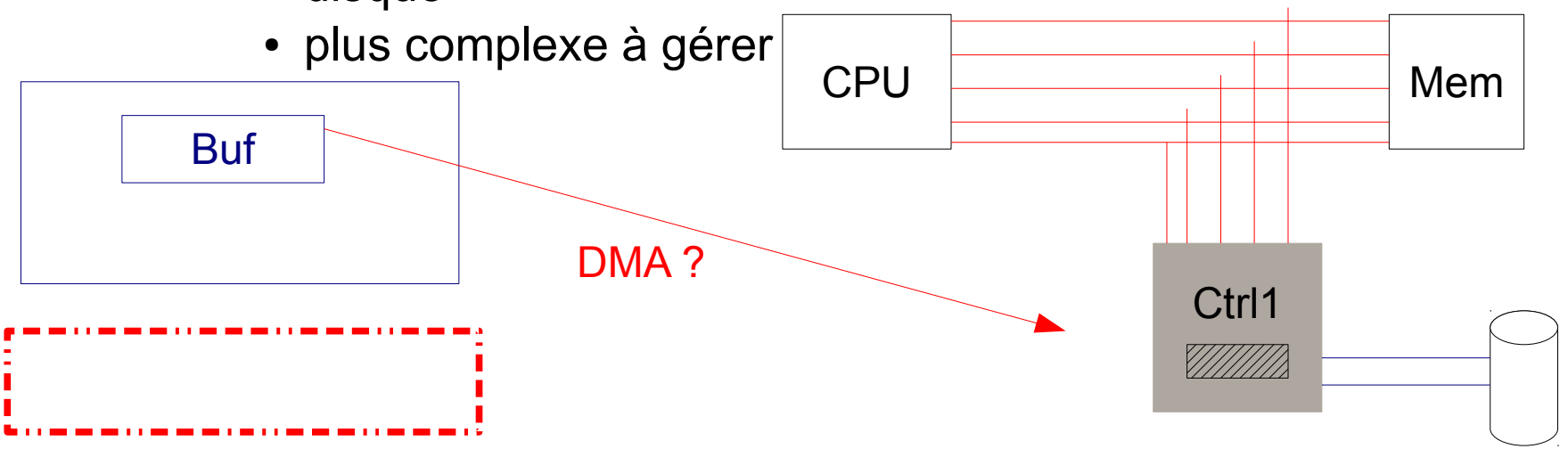
Rôle des device drivers(2)

- Comment implémenter write(fd, *buf, size) ?
  - Première solution
    - passage en mode kernel, sauvegarde du contexte
    - l'appel système copie le buffer buf dans un buffer du kernel
      - inconvénient : coût de la copie
    - le kernel localise le device et fait appel à la routine adhoc et le processus est mis en attente sur ce device
    - le DMA est armé pour transférer les données au contrôleur
    - ...



Rôle des device drivers(3)

- Comment implémenter write(fd, *buf, size) ?
  - Deuxième solution
    - passage en mode kernel, sauvegarde du contexte
    - l'appel système utilise le buffer de l'utilisateur et arme directement le DMA
      - avantage : pas de copie inutile de données
      - inconvénients
        - le DMA ne copiera pas la donnée immédiatement, il ne faut pas que le processus puisse modifier ne contenu du buffer ni que l'OS ne déplace la page correspondante sur disque
        - plus complexe à gérer



Rôles des device drivers (4)

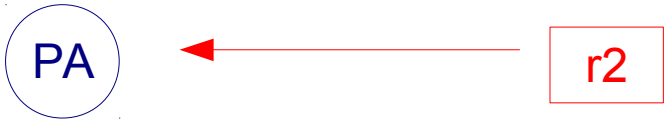
- Gestion des erreurs
  - le driver doit être prêt à faire face à toutes les erreurs possibles, jusqu'au retrait d'un device
    - erreur de l'utilisateur qui envoie de mauvaise commandes
    - mauvais fonctionnement du dispositif
    - erreurs transitoires
- Attribution des dispostifs
  - certains dispositifs doivent être utilisés de façon
    - exclusive
    - driver autorise/refuse l'accès lors de l'appel open
    - spooler

Deadlock

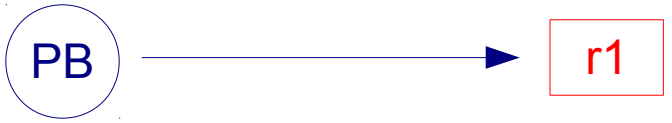
- Définition
  - Un ensemble de processus est en deadlock si chacun de ces processus attend un événement que seul un autre de ces processus peut provoquer
- Conditions pour aboutir à un deadlock
  - Mutual exclusion
    - Chaque ressource est disponible ou allouée à un processus
  - Hold and wait
    - Un processus qui possède une ressource peut en demander d'autres
  - No preemption
    - Une ressource allouée à un processus ne peut être désallouée que par ce processus
  - Circular wait
    - Il y a une liste de deux ou plus processus qui attendent chacun une processus allouée à un autre processus de la liste

Comment modéliser les deadlock?

- Sous la forme d'un graphe dirigé
- Processus
- Ressource
- Ressource r2 allouée au processus A

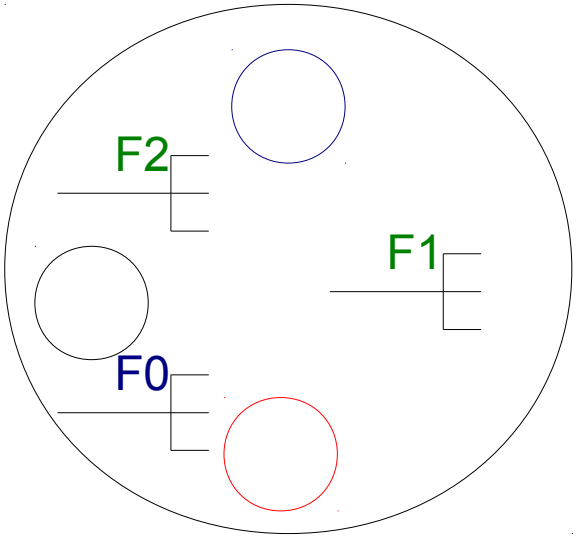


- Processus B demande la ressource r1



Le problème des « Dining philosophers »

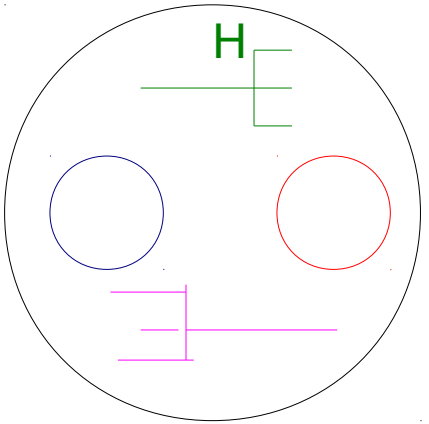
- Problème
  - N philosophes doivent se partager un repas dans une salle de méditation
  - La table contient N fourchettes et N assiettes
  - Chaque philosophe a un place réservée et a besoin pour manger de
    - La fourchette à sa gauche
    - La fourchette à sa droite
- Comment coordonner
  - efficacement l'accès à la table ?



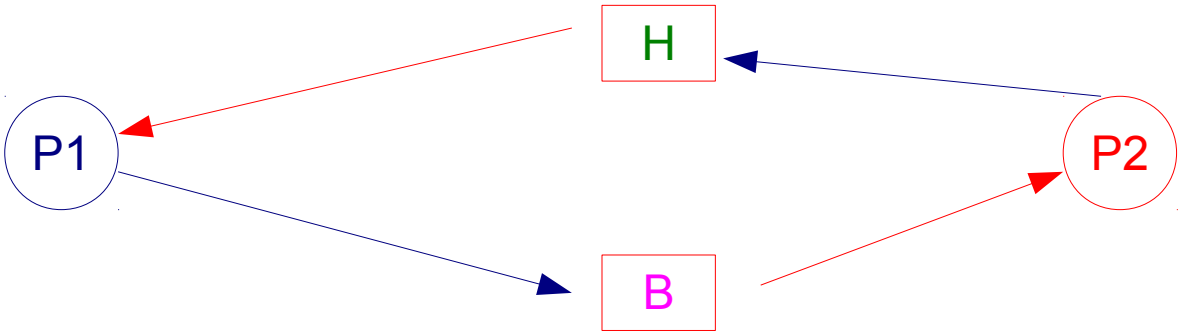
Comment modéliser les deadlock? (2)

- Problème des 2 philosophes

Philosophe1 demande H
Puis B



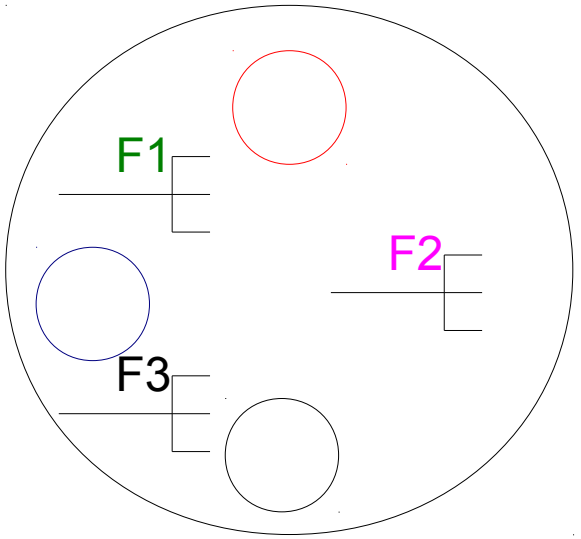
Philosophe2 demande B
Puis H



La présence d'un cycle dans le graphe dirigé indique la présence d'un deadlock.

Comment modéliser les deadlock? (3)

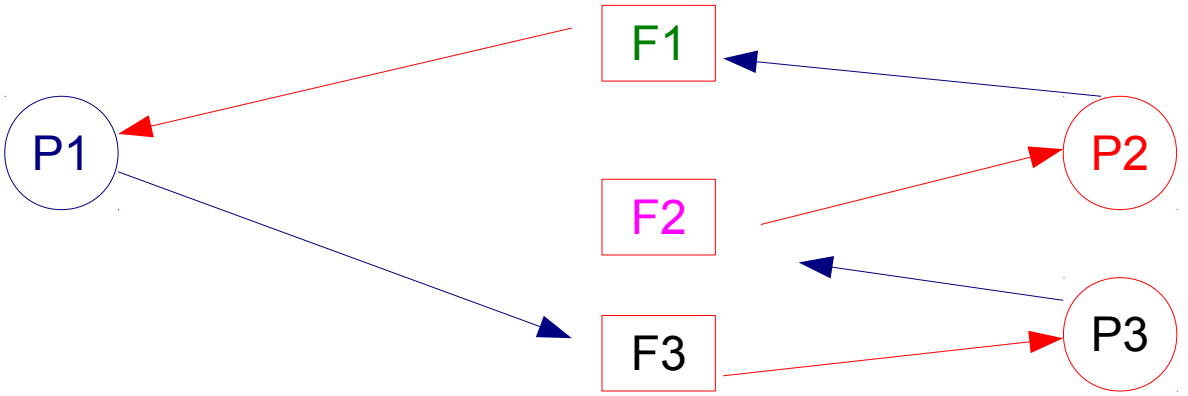
- Problème des 3 philosophes



Philosophe1 demande F1
Puis F3

Philosophe2 demande F2
Puis F1

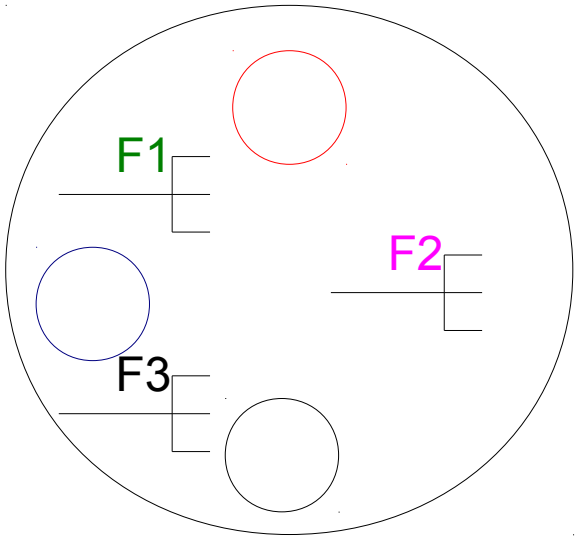
Philosophe3 demande F3
Puis F2



La présence d'un cycle dans le graphe dirigé indique la présence d'un deadlock.

Comment modéliser les deadlock? (4)

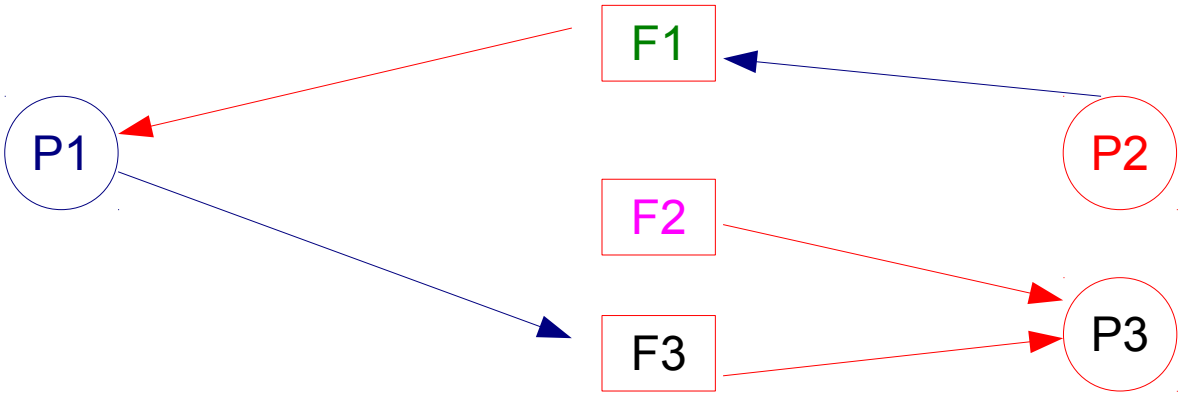
- Problème des 3 philosophes



Philosophe1 demande F1
Puis F3

Philosophe2 demande F1
Puis F2

Philosophe3 demande F3
Puis F2



La présence d'un cycle dans le graphe dirigé indique la présence d'un deadlock.

Comment éviter les deadlocks ?

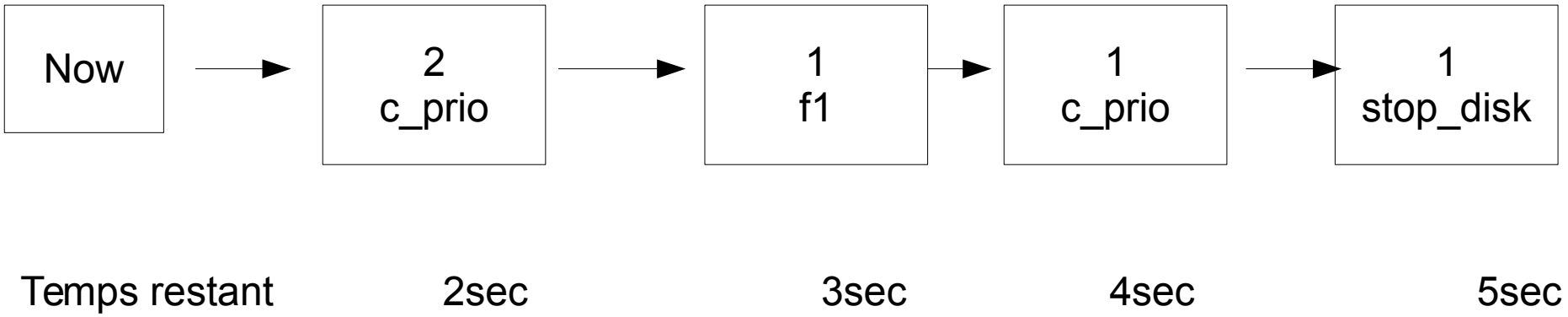
- Solutions possibles pour chaque condition
  - Mutual exclusion
    - Chaque ressource n'est réservable que par un spooler
    - Tous les accès se font via le spooler
  - Hold and wait
    - Demander à chaque processus de demander toutes les ressources qu'il utilisera au début
      - Difficile pour un processus de savoir quels ressources il utilisera
      - Provoque un gaspillage de ressources
  - No preemption
    - Impossible de préempter certaines ressources
  - Circular wait
    - Numéroter les ressources et forcer chaque processus à les demander uniquement dans cet ordre
      - parfois il est difficile de parvenir à ordonner les ressources

Horloge

- Gestion de l'horloge
  - le kernel Unix maintient dans une variable l'heure courante
    - compteur 64 bits
    - deux compteurs sur 32 bits
  - le kernel programme le chip d'horloge temps réel pour qu'il génère une interruption toutes les 10msec
  - A chaque interruption d'horloge
    - Mise à jour de l'heure courante
    - Eviter que des processus ne monopolisent le CPU
      - via activation régulière du scheduler
    - Comptabiliser le temps CPU utilise par chaque processus
    - Gestion de l'appel système alarm
    - Gestion des temporisateurs
      - internes au kernel
      - utilisés par les processus utilisateurs

Temporisateurs

- Dans les processus utilisateurs
  - fonction à exécuter à un moment donné
- Dans le kernel
  - fonctions à exécuter dans x secondes
- Gestion des temporisateurs
  - Exemple
    - toutes les 2 secondes : c_prio
    - dans 3 secondes : f1
    - dans 5 secondes : stop_disk



Les appels système setitimer et alarm

- Utilisés par un processus pour demander une notification
  - toutes les n secondes pour setitimer
  - dans n secondes pour alarm
- A l'expiration du délai, le kernel enverra un signal ITERM/SIGALARM au processus
  - signal=notification asynchrone
  - ce signal provoquera l'exécution d'une routine de traitement définie par le processus utilisateur

Les signaux

- Quelques exemples
  - SIGALRM
    - expiration d'une alarme
    - action par défaut : fin du processus
  - SIGKILL
    - force la terminaison du processus
  - SIGUSR1/SIGUSR2
    - signaux utilisables entre processus
    - action par défaut : fin du processus
  - SIGCHLD
    - fin d'un processus fils du processus courant
    - action par défaut : ignoré
  - SIGSTOP
    - processus bloqué temporairement
  - SIGCONT
    - processus bloqué redémarre

Utilisation des signaux

- Dans un processus utilisateur

Dans main

```
...
    /* handler for SIGINT */
    if (signal(SIGINT, sig_int) == SIG_ERR)
       printf("signal(SIGINT) error");
    /* handler for SIGCHLD */
   if (signal(SIGCHLD, sig_chld) == SIG_ERR)
     printf("signal(SIGCHLD) error");
...
```

```
static void sig_int(int signo)
{   printf("caught SIGINT\n");
    return;
}
```

```
static void sig_chld(int signo)
{ printf("caught SIGCHLD\n");
   return;
}
```

Gestion des signaux par le kernel

- Contenu de la table des processus
  - Plusieurs bitmasks relatifs aux signaux
    - bitmask des signaux reçus
      - mis à jour par le kernel lorsqu'un signal arrive
    - bitmask des signaux ignorés
      - installé avec les valeurs par défaut et mis à jour via l'appel système signal
  - Tableau des gestionnaires de signaux
    - un pointeur vers chaque signal handler installé par le processus
- Arrivée d'un signal
  - kernel consulte table processus et modifie bitmask
- Réactivation d'un processus
  - avant de faire le changement de contexte, kernel vérifie si un signal est en attente
    - si oui, activer le handler ou l'action par défaut sinon