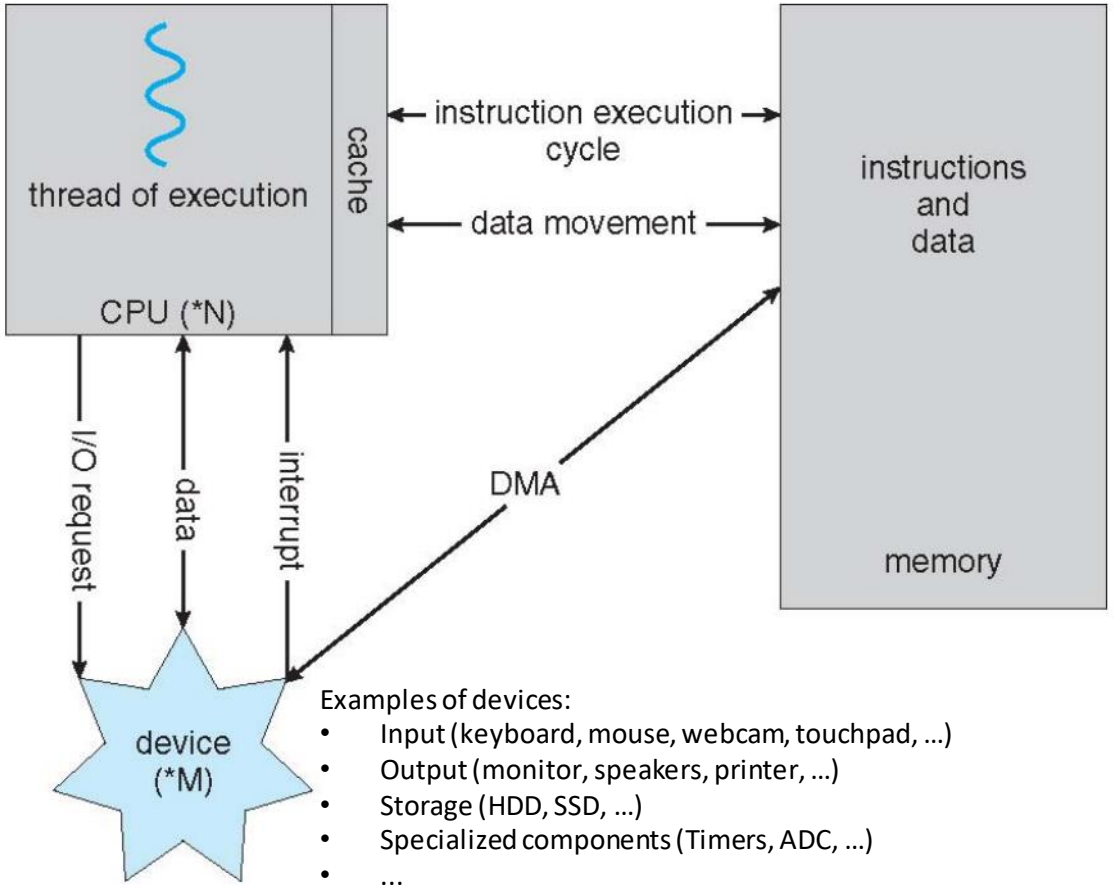
Systèmes d'exploitation

Demo Processus

Rappel: Architecture



Examples of devices:

- Input (keyboard, mouse, webcam, touchpad, …)
- Output (monitor, speakers, printer, …)
- Storage (HDD, SSD, …)
- Specialized components (Timers, ADC, …)
- …

Thread: Séquence d'instructions

Un **CPU** contient un nombre fini de processeurs/coeurs.

Interrupt

1. Trigger: Timer, Complétion d'une operation IO, …
2. Request (IRQ): Gestion, priorité, …
3. Service Routine (ISR): Context Switch, saute au handler du interrupt
4. Return from interrupt (RTI): Restore context switch, retour a l'éxecution normale

Les interrupts sont très important pour un système d'exploitation.

Hiérarchie de stockage:

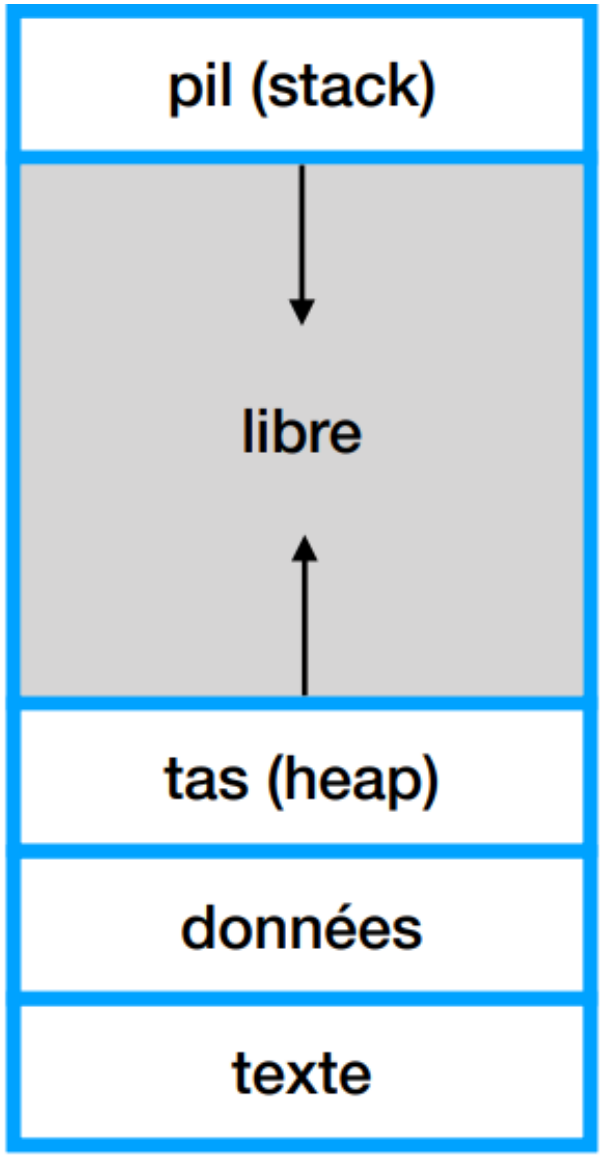
1. Registres
2. Mémoire cache
3. Mémoire central
4. Disk (Pas volatile)

C'est quoi un processus?

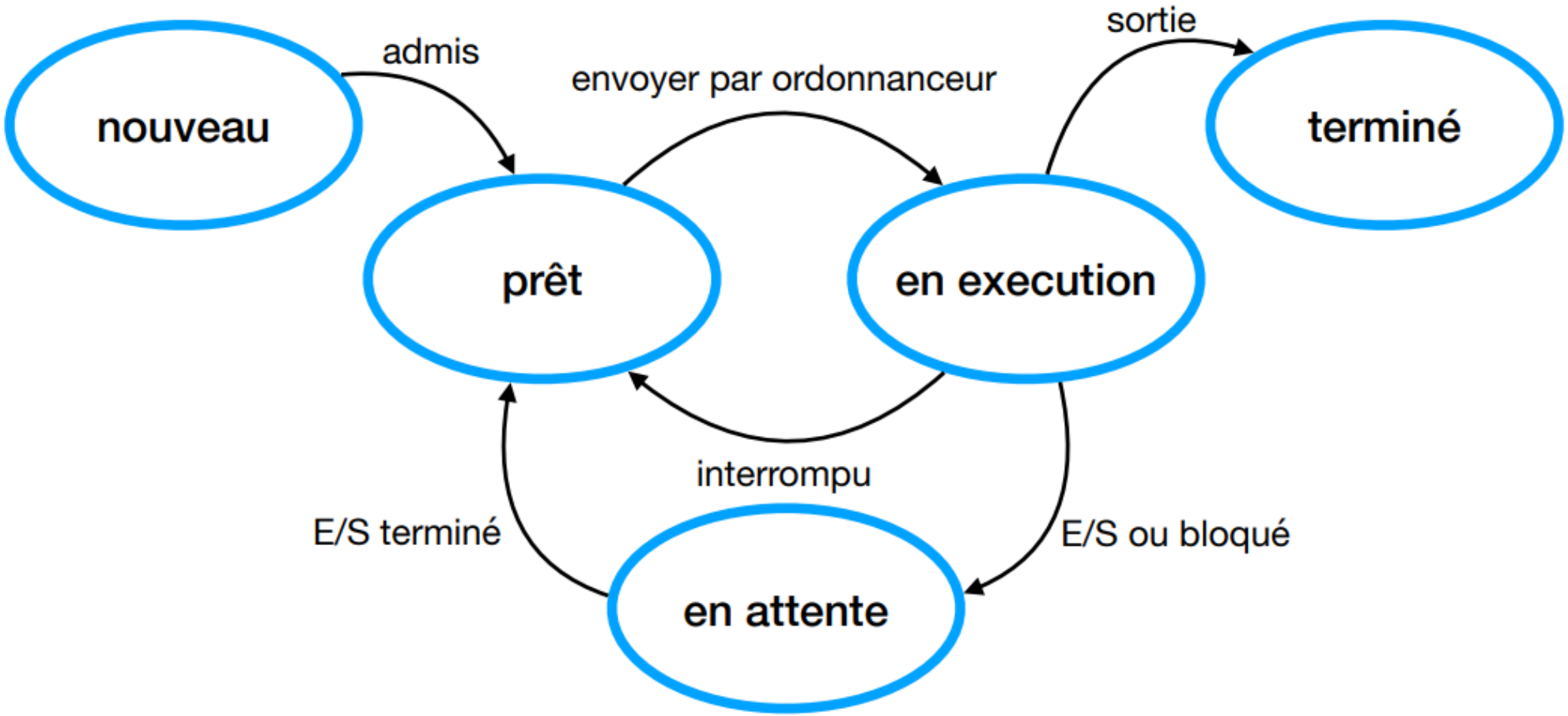
Un processus est un programme en execution. Cette abstraction est fourni par le SE.

Bloc de Contrôle de Processus (PCB)

- État
- ID
- Registres
  - Program counter (PC) / Instruction Pointer (IP)
  - Stack pointer
  - Frame pointer
  - ...
- Mémoire addressable (address space) (Contient le code)
- Informations IO (Ex: listes de fichiers ouverts)
- Autres informations utile pour le SE
  - Temp CPU utilisé
  - ...



Quels sont les états d'un processus?


nouveau
admis
prêt
envoyer par ordonnanceur
en execution
sortie
terminé
interrompu
E/S terminé
en attente
E/S ou bloqué

C'est quoi un context switch?

Le processus de sauvegarde et de restauration de l'état d'un processeur afin de passer de l'exécution d'un processus a un autre.

Il faut être capable de sauvegarder et restaurer le PCB.

Sa peut etre chère en temp CPU de faire un context switch, donc l'ordonnanceur doit minimiser les context switch tout en poursuivant d'autres objectifs (Temp de réponse, …)

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

Création: Devrait fournir un moyen de créer de nouveaux processus. En POSIX, **fork** et **exec**.

Destruction: On aimerait être capable de forcer la terminaison d'un processus. En POSIX, **kill**.

Attente: Attendre qu'un processus termine son exécution. En POSIX, **wait** ou **waitpid**.

Statut: Obtenir des informations d'état sur le processus. En POSIX, **waitpid**.

Comment est-ce qu'un processus peut communiquer avec un autre (IPC)?

Mémoire Partagé: Un bloc de mémoire accessible par plusieurs processus.

Transmission de messages: Envoie des messages qui est géré par le SE.

Semaphores: Résoud des problèmes de synchronization. C'est essentiellement des entiers positifs.

Sockets: Permet une communication standard, indépendant de l'ordinateur ou du SE.
Principalement utilisé pour communiquer sur un réseau.

Est-ce que vous pensé que fork-exec c'est une bonne abstraction pour la création de processus?

Windows: CreateProcess:

1. Crée un nouveau processus
2. Charge le programme du disque

Unix: fork-exec

1. Crée un nouveau processus
2. Initialize le processus enfant avec une copie du parent
3. Hérite du context d'execution du parent (ex, les fichiers ouvert)
4. Exec peut être utilisé pour chargé un nouveau programme du disque.

Copy-on-write: Permet optimiser la copy du parent

L'appel à la fonction exec remplace complètement le processus actuel (code, données, tas, pile...). Cependant, certains attributs du processus parent sont préservés:

- Fichiers ouvert (Open file descriptors)
- Variables d'environement
- ...

Permet de réaliser facilement des actions utiles. Par exemple:

```
wc p3.c > newfile.txt
```

Ce que le shell va faire:

- Création du processus enfant avec fork
- Ferme le standard output
- Ouvre le ficher
- Execute le programme wc avec exec

```c
int main(int argc, char *argv[])
{
  int rc = fork();
  if (rc < 0) { // fork failed; exit
    fprintf(stderr, "fork failed\n");
    exit(1);
  } else if (rc == 0) {
    // child: redirect standard output to a file
    close(STDOUT_FILENO);
    open("./p4.output", O_CREAT | O_WRONLY | O_TRUNC, S_IRWXU);

    // now exec "wc"...
    char *myargs[3];
    myargs[0] = strdup("wc");   // program: "wc" (word count)
    myargs[1] = strdup("p4.c"); // argument: file to count
    myargs[2] = NULL;           // marks end of array
    execvp(myargs[0], myargs);  // runs word count
  }
  else { // parent goes down this path (original process)
    int wc = wait(NULL);
  }
  return 0;
}
```

Qu'est-ce qui est imprimé?

```c
int main(int argc, char *argv[])
{
    printf("hello world (pid:%d)\n", (int) getpid());
    int rc = fork();
    if (rc < 0) {
        fprintf(stderr, "fork failed\n");
        exit(1);
    } else if (rc == 0) {
        printf("Child (pid:%d)\n", (int) getpid());
        sleep(1);
    } else {
        wait(NULL);
        printf("Parent of %d (pid:%d)\n", rc, (int) getpid());
    }
    return 0;
}
```

Hello world (pid: 29146)

Child (pid: 29147)

Parent of 29147 (pid: 29146)

Qu'est-ce qui est imprimé?

```c
int main()
{
  int pid;

  pid = fork();
  pid = fork();

  printf("1 "); // how many times with this get printed?

  return 0;
}
```

1 1 1 1

Qu'est-ce qui est imprimé?

```c
int main()
{
  int pid, pid1, pid2;
  pid1 = getpid();
  pid = fork();
  pid2 = getpid();
  if (pid == 0){
    printf("child: pid = %d \n", pid);
    printf("child: pid = %d \n", pid1);
    printf("child: pid = %d \n", pid2);
  }
  else{
    wait(NULL);
    printf("parent: pid = %d \n", pid);
    printf("parent: pid = %d \n", pid1);
    printf("parent: pid = %d \n", pid2);
  }

  return 0;
}
```

child: pid = 0

child: pid = 51283

child: pid = 51284

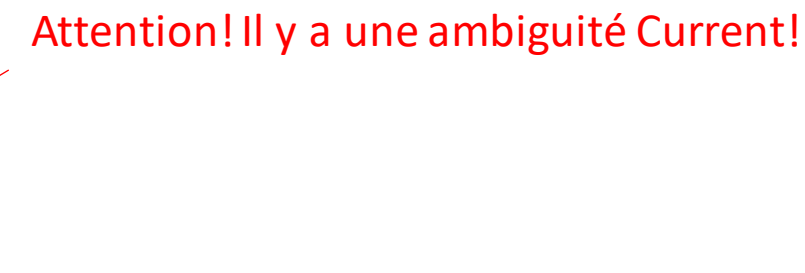
parent: pid = 51284

parent: pid = 51283

parent: pid = 51283

Qu'est-ce qui est imprimé?

```
int main(int argc, char *argv[])
{
  int pid1 = fork();
  printf("Current (pid:%d)\n", (int) getpid());
  if (pid1 == 0){
    printf("Executing ls -l (pid:%d)\n", (int) getpid());
    execlp("ls", "ls", "-l", NULL);
    printf("Finished executing ls -l (pid:%d)\n", (int) getpid());
  }
  else {
    wait(NULL);
    printf("Parent process (pid:%d)\n", (int) getpid());
  }
  return 0;
}
```



Current (pid:51431)

Current (pid:51432)

Executing ls -l (pid:51432)

total 24

-rwxrwxrwx 1 mattpc mattpc 16128 Jan 23 01:36 a.out

-rwxrwxrwx 1 mattpc mattpc 517 Jan 23 01:35 ex.c

-rwxrwxrwx 1 mattpc mattpc 855 Jan 23 00:54 redirection.c

Parent process (pid:51431)