

Résumé 1 - Système d'Exploitation

Introduction aux SE / Programmation de processus

- **Appels système** : Ensemble d'instructions étendues, spécifiques d'un SE, qui constitue l'interface entre un SE et les programmes utilisateurs.
 - Exemples d'actions possibles
 - Création d'un processus (fils) par un processus actif : `fork`
 - Attendre la fin d'un processus fils : `wait`
 - Destruction d'un processus : `kill`
 - Mise en attente, réveil d'un processus : `sleep, wait`
 - Suspension et reprise d'un processus grâce à l'ordonnanceur de processus (scheduler)
 - Demande de mémoire supplémentaire ou restitution de mémoire inutilisée : `allocation dynamique, malloc, free`.
- Les processus
 - Informations sur les processus
 - Notion de père/fils
 - `pid` : identifiant unique du processus. Chaque processus est identifié par son PID unique
 - `ppid` : identification du processus père
 - `uid` : l'identification de l'utilisateur exécutant le processus / uid réel
 - `euid` : uid effective / qui correspond au privilèges
 - `gid` : l'identification groupe de l'utilisateur exécutant le processus / gid réel
 - `egid` : gid effective
 - Hiérarchie des processus
 - Lorsqu'un processus crée un autre processus, les processus père et fils continuent d'être associés d'une certaine manière. **Chaque processus connaît le PPID de son parent.**
 - **Le processus enfant** peut lui-même créer **plusieurs processus**, formant une hiérarchie de processus.
 - **Un processus a un seul parent** mais peut avoir aucun ou plusieurs fils.
 - Création d'un processus
 - Appel à l'appel système – fonction `fork()` de C ou `os.fork()` de Python. Création par clonage (mitose) : un processus (**le père**) demande, en appelant la primitive `fork()`, la création dynamique d'un nouveau processus (**le fils**). Le fils s'exécute ensuite de façon concurrente avec le père.
 - `pid_t fork (void);` `pid_t` est un type (long int) défini dans la librairie `<sys/types.h>`
 - Processus fils
 - partage le segment de texte du père
 - dispose d'une copie de son segment de données
 - hérite du terminal de contrôle
 - hérite d'une copie des file descripteur ouverts
 - n'hérite pas du temps d'exécution, ni de la priorité
 - n'hérite pas des signaux en suspend
 - Appel à la primitive `fork()`

- Retourne 0 au fils
- Retourne le PID du fils au père, ou -1 si échec
- Synchronisation
 - `exit (int etat)`
 - Termine un processus normalement, `etat` est un octet (donc valeurs possibles : 0 à 255) renvoyé dans une variable du type `int` au processus père.
 - Usage de la variable `etat` : 0 → ok ; ≠ 0 → code erreur
 - Constantes `stdlib.h` : `EXIT_SUCCESS`, `EXIT_FAILURE`
 - `pid_t wait (int *status)`
 - `pid_t waitpid (pid_t pid, int *status, int options)`
 - Suspend le processus jusqu'à la terminaison de l'un de ses fils
 - Achèvement du père : fils pris en charge par le processus init (PID = 1)
 - `status` : valeur de `exit` ou autre (dans le cas d'un signal)
 - `wait` attend la fin de n'importe quel fils et renvoie son PID ou -1 dans le cas où il n'y a pas (ou plus) de fils
 - `waitpid` pour attendre un processus particulier dont on connaît son pid
 - `getpid ()` et `getppid ()`
 - Pour obtenir l'identification d'un processus et l'identification du processus père