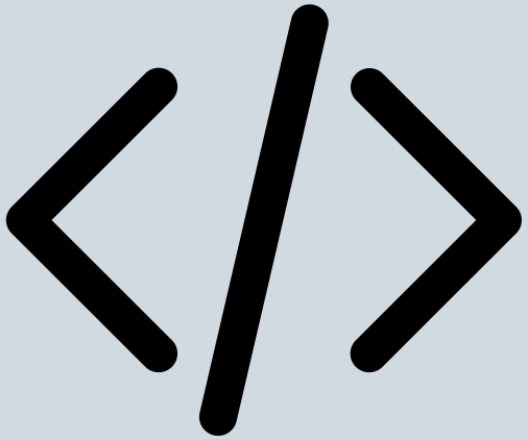

# Systèmes d’exploitation: exec

Exécution d'une commande

# La famille exec

- fork() duplique le processus appelant
- exec() écrase le code et les données du processus en les remplaçant par ceux de la commande passée en paramètre
- Voir aussi popen() et system() basées sur fork() et exec()
- Parmi les variantes de exec*() , le seul appel système réel sous Linux est execve()

# execl(), execle(), execlp(), execv(), execve(), execvp(), fexecve()

- l : arguments dans une liste terminée par NULL
- v : arguments dans un tableau comme argv[]
- e : envp[] sinon extern char **environ
- p : recherche dans le PATH sinon chemin complet

# Exemple avec execvp()

```
int main(void) {
        char * myargv[] = { "ls", "-l", "-i", (char *) NULL };
        execvp("ls", myargv);
        fprintf(stderr, "Erreur %d\n", errno);
        return EXIT_FAILURE;
}
```

Si execvp() réussit, les deux dernières lignes ne seront jamais prises en compte car le code du processus courant aura été remplacé par celui de la commande **ls –l –i**

Vérifiez le résultat avec echo $?

# execve()

```
int execve(const char *filename, char *const argv[], char *const envp[]);
```

- La chaîne filename doit contenir le chemin d'accès au programme à lancer (relatif ou absolu)
- argv[] doit contenir les arguments que l'on trouve habituellement sur la ligne de commande, avec au moins argv[0] qui contient le nom de l'application (basename uniquement)
- envp[] doit contenir les variables d'environnement transmises au processus : on peut réutiliser la variable environ mais attention en cas de Set-UID root
- Les deux tableaux terminent par des pointeurs NULL

# exemple_execve.c

```
extern char ** environ;
int main (void) {
        char * argv[] = {“bash", "-c", "echo $SHLVL", (char *) NULL };
        fprintf(stdout, "Je lance /bin/bash -c \"echo $SHLVL\" :\n");
        execve("/bin/bash", argv, environ);
        fprintf(stdout, "Raté : erreur = %d\n", errno);
        return EXIT_SUCCESS;
}
```

$SHLVL donne le niveau de shell: voir demo
Ne marche plus avec sh mais bash

# exemple_execlp.c

```
int main (int argc, char * argv []){
        char compteur[2];
        int i = 0;
        if (argc == 2)  sscanf(argv[1], "%d", & i);
        if (i < 5) {
        i ++;
                sprintf(compteur, "%d", i);
                fprintf(stdout, "execlp(%s, %s, %s, NULL)\n", argv[0], argv[0], compteur);
                execlp(argv[0], argv[0], compteur, (char *) NULL);
        }
        return EXIT_SUCCESS;
}
```

# Causes d'échec

- Si exec() réussit il n’y a pas de retour dans le fil d’exécution du programme
- Si le programme lancé par exec() termine normalement par exit(), abort() ou return de la fonction main(), le processus termine
- Si exec() revient dans le processus appelant alors il y a eu un problème
- Voir man et exemple_execv.c

# system()

```
int main(void) {
        system("ls -l");
        printf("On a fini ls\n");
        return EXIT_SUCCESS;
}
```

Problèmes :

- system() n'est pas interruptible et si la commande échoue on peut rester bloqué
- les arguments de la commande doivent être concaténés "à la main"
- on peut avoir des trous de sécurité si le programme est exécuté en suid

# popen() : vu plus tard

- Combine fork et exec
- Plus un flux d'entrée ou de sortie entre le père et le fils selon le mode d'ouverture ("r" ou "w")