Cours système

# PID

PID : Process ID

PPID : Parent Process ID

Pid 0 est idle

Pid 1 est le premier process exécuté

# Fonctions :

#include <sys/types.h>

#include <unistd.h>

pid\_t getpid (void); /\*pid\*/

pid\_t getppid (void); /\*parent pid\*/

Affichage des valeurs :

printf ("Mon PID=%jd\n", (intmax\_t) getpid ()); /\*notez le casting\*/

printf ("PID du Parent =%jd\n", (intmax\_t)getppid ()); /\*notez le casting\*/

# Pere-Fils

Un processus qui engendre est le père, celui engendré est le fils.

Arborescence depuis init vers les processus fils

Un pus fils hérite de l’uid (user id) et gid (group id) de son père.

# Nouveau pus : exec et fork

Sous linux : - chargement d’un programme en mémoire ou création d’un new pus par duplication.

## Commande exec

Pour charger une nouveau programme, le pid, le ppid, le uid et le guid + ensem des fichiers ouverts ne changent pas

int execl (const char \*path, const char \*arg, ...); /\* les paramètres dans arg \*/

int ret;

ret = execl ("/bin/ls", "monls", NULL); /\* NULL stoppe la prise de paramètres \*/

if (ret == −1) perror ("execl");

## Commande fork

Duplique le pus, même programme exécutable mais pid différent, ppid = pid du père, même image du programme,

#include <sys/types.h>

#include <unistd.h>

pid\_t fork (void); /\* si pid 0 on est dans le pus fils, si pid -1 == erreur, sinon on est dans le processus père et ça retourne le pid du fils \*/

### exemple :

#include <unistd.h>

#include <stdio.h>

#include <sys/types.h>

#include <sys/wait.h>

int main(void)

{

printf("pid du processus pere %d",getpid());

pid\_t pid;

pid = fork();

if (pid > 0)

{

printf(" pid du fils=%d!\n", pid);

}

else

{

if (!pid)

{

printf("\nOn est dans le pus enfant! .. %d, pid du new pus %d\n",pid,getpid());

}

else

{

if (pid == -1)

{

printf("erreur fork");

perror("erreur fork");

}

}

}

}

Autre exemple :

⇒ penser à include stdlib.h pour les constantes EXIT\_\*

int main() {

pid\_t pid = fork();

**if** (pid == -1) {

*// Il y a une erreur*

perror("fork");

**return** EXIT\_FAILURE;

} **else** **if** (pid == 0) {

*// On est dans le fils*

printf("Mon PID est %i et celui de mon père est %i**\n**", getpid(), getppid());

} **else** {

*// On est dans le père*

printf("Mon PID est %i et celui de mon fils est %i**\n**", getpid(), pid);

}

**return** EXIT\_SUCCESS;

}

## exec/fork

pid\_t pid;

pid = fork ();

if (pid == −1) perror ("fork"); /\* fils \*/

if (!pid)

{

const char \*args[] = { "ls", NULL };

int ret;

ret = execv ("/bin/ls", args);

if (ret == −1) {

perror ("execv");

exit (EXIT\_FAILURE);

}

}

# Terminer pus : exit, atexit

## exit

– appel des triggers

– écriture des données I/O

– destruction des fichiers temporaires

– libération des ressources allouées par le noyau:

mémoire, fichiers ouverts, sémaphores.

– notification au parent de la fin de son processus

fils

#include <stdlib.h>

void exit (int status);

## atexit

#include <stdlib.h>

int atexit (void (\*function)(void));

#include <stdio.h>

#include <stdlib.h>

void out (void)

{

printf ("atexit() fonctionne bien!\n");

}

int main (void)

{

if (atexit (out)) fprintf(stderr, "atexit()

erreur!\n");

return 0;

}