



Système d'exploitation avancé

Processus

Pierre LEROY – leroy.pierre1@gmail.com

Sommaire

- I. Processus
- II. Appels système
- III. Essentiel
- IV. Conclusion

Définitions

- Processus \neq programme :

DISTINCTIONS

PROGRAMME

- Fichier exécutable sur le disque contenant
 - ✓ en-tête
 - ✓ code binaire (instructions machine)
 - ✓ données statiques (segment data)

ELF

STATIQUE

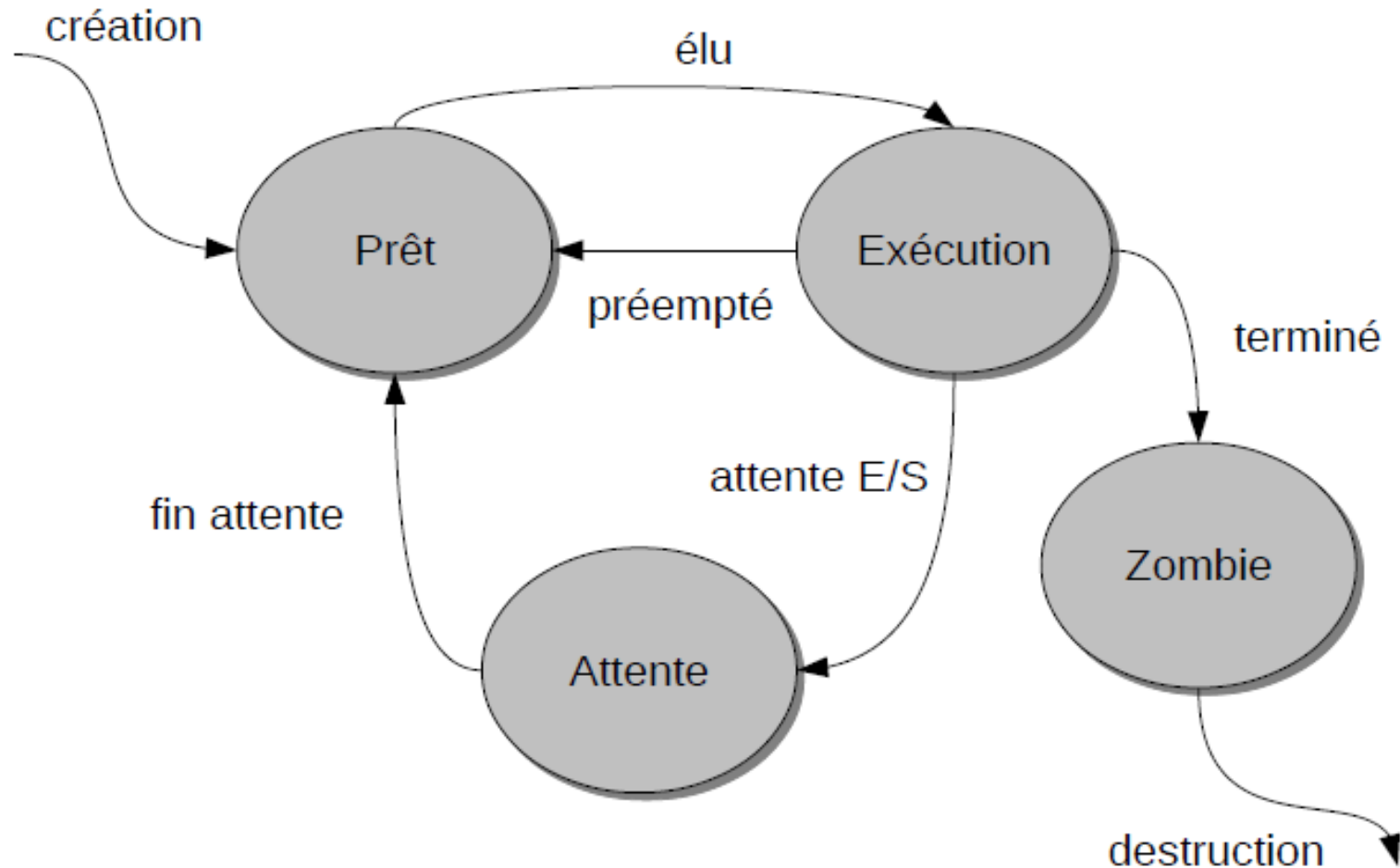
PROCESSUS

- Programme en cours d'exécution
- Constitution :
 - ✓ entrée dans la table des processus
 - ✓ ressources physiques allouées CPU/RAM/...
 - ✓ présent en RAM
 - ✓ Instruction machine
 - ✓ pile/tas
 - ✓ Variables globales

DYNAMIQUE

Etats d'un processus

- Workflow d'un processus :



Attributs

- Un processus possède des métadonnées

ENSEMBLE

METADONNEES

- Propriété d'un processus :
 - ✓ **numero (PID : Process Identifier)**
 - ✓ **numero du pere (PPID)**
 - ✓ chemin de l'exécutable
 - ✓ infos utilisation processeur
 - ✓ infos ordonnancement (priorite, ...)
 - ✓ **fichiers ouverts**
 - ✓ localisation memoire (code, pile, ...)
 - ✓ **proprietaire (uid, gid)**
 - ✓ **code retour**
 - ✓ ...

Table des processus

- Un noyau linux maintient une table de processus

DESCRIPTION

- Table composée structure ***task_struct***
 - ✓ une entrée = 1 processus = structure contenant w/ attributs
- Pour visualiser la table :
 - ✓ ps (options eaux)
 - ✓ pstree
 - ✓ top/htop
 - ✓ /proc/<pid>/

```
struct task_struct {
    volatile long    state;        /* -1 unrunnable, 0 runnable, >0 stopped */
    long            priority;
    int             errno;
    int             pid;
    int             pgrp;
    [...]
    unsigned short   uid,euid,suid,fsuid;
    unsigned short   gid,egid,sgid,fsgid;
    unsigned long     timeout, policy, rt_priority;
    long             utime, stime, cutime, cstime, start_time;
    [...]
    struct fs_struct  *fs;
    /* open file information */
    struct files_struct *files;
    /* memory management info */
    struct mm_struct  *mm;
    /* signal handlers */
    struct signal_struct *sig;
    [...]
};
```

Arborescence des processus

- Paradigmes fondamentaux sur l'organisation des processus :

A RETENIR

- La racine de l'arbre des processus est le processus init PID = 1
- Chaque processus possède un processus père
 - ✓ identifié par le champ **PPID**
 - ✓ une entrée = 1 processus = structure contenant w/ attributs
- Chaque processus renvoie un code retour
 - ✓ echo \$?

Sommaire

I. Processus

II. Appels système

III. Essentiel

IV. Conclusion

Fondamentaux

- 5 appels système sont essentiels

DEFINITION

- Les appels système suivants sont essentiels pour la gestion des processus :
 - ✓ fork => réalise l'opération de fork
 - ✓ `exec[l/lp/v/vp/vpe]` => réalise l'exécution d'une commande
 - ✓ wait => attend un processus relatif
 - ✓ exit => termine l'exécution du processus en retournant un code
 - ✓ dup/dup2 => redirection de fichier

Fork

- Appel système *fork*

DEFINITION

- Attribution d'une nouvelle entrée dans la table des processus
- Les 2 processus partagent le même code
 - ✓ *le processus fils travaille sur une copie des données du père, toute modification dans un processus n'est pas visible de l'autre*
- Le père transmet les descripteurs de fichiers à son fils
 - ✓ *les descripteurs du père et du fils pointent sur les mêmes entrées dans la table des fichiers ouverts*
 - ✓ *=> même position dans le fichier*

Fork

- Appel système *fork*

DEFINITION

```
pid_t  fork(void); /** >duplication du processus courant */
```

The diagram shows a vertical line with two horizontal branches. The top branch points to the `fork` function in the code block above and is labeled "0 chez le fils". The bottom branch points to the `pid_t` return type and is labeled "pid du nouveau processus chez le père".

```
int main() {  
    int f;  
    f = fork();  
    printf( "pid : %d\n", f );  
    return( 0 );  
}
```

```
Prompt > ./main  
pid : 0  
pid : 25992  
Prompt >
```

Exit

- Appel système *exit*

DEFINITION

- attribution d'une nouvelle entrée dans la table des processus
- chaque processus renvoie un code de retour
 - ✓ *soit par exit, soit implicitement (valeur de retour de la fonction main)*
 - ✓ *bash : echo \$?*
 - ✓ *0 : OK, != 0 => échec*

```
void exit(int code_de_retour);
```

Wait

- Appel système *wait*

DEFINITION

- attend la terminaison d'un processus fils
 - ✓ *status* : doit être exploitée avec les macros fournies (cf man page man 2 wait)
 - ✓ voir aussi ***waitpid()***

```
pid_t wait(int *status);
```

↑
Pid du fils mort

↑
Contient le code de retour du fils, NULL pour l'ignorer

Dup

- Appel système *dup* ou *dup2*

DEFINITION

- redirige les fichiers
 - ✓ *doit être exploité avec close()*
 - ✓ *utilisé pour les redirections de flux*

```
int dup(int desc);
```

Descripteur à dupliquer

Nouveau desc (le premier libre dans la table), ancien et nouveau desc partagent position, flags -1 si erreur

```
int dup2(int ancien_desc, int nouv_desc);
```

Idem mais force la valeur du nouveau descripteur

Exec

- Appel système *exec[...]* recouvrement d'un processus

DEFINITION

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

↑
Si succès ne revient pas
-1 en cas d'échec

↑
Chemin de
l'exécutable

↑
arguments

↑
Variables
d'environnement

Execve recouvre le code, la pile, le tas et
ne conserve que les descripteurs

Exec

- variantes Plusieurs frontaux pour `execve` :

DEFINITION `exec +`

l (liste) ou **v** (vecteur) : pour les arguments

p (path) : utiliser PATH ou non

e (environnement) : précision des variables d'environnement

```
int execl(const char *path, const char *arg, ...);  
int execlp(const char *file, const char *arg, ...);  
int execle(const char *path, const char *arg, ...,  
            char * const envp[]);  
int execv(const char *path, char *const argv[]);  
int execvp(const char *file, char *const argv[]);
```


Sommaire

I. Processus

II. Appels système

III. Essentiel

IV. Conclusion

Essentiel

- Toutes les notions abordées dans ce chapitre sont fondamentales



Conclusion



Annexes

Annexes

- Liens annexes :
 - ✓ *Format ELF* : https://fr.wikipedia.org/wiki/Executable_and_Linkable_Format
 - ✓ *Structure task_struct* : <http://www.tldp.org/LDP/tlk/ds/ds.html>