

Systèmes d'exploitation, 2ème année

Primitives Execs

Yves STADLER

Université Paul Verlaine - Metz

22 novembre 2011

Agenda

Plan du cours

- Concept
- Fonctions
- Réalisation

Notion

- Lancer un programme existant depuis un autre programme
- Idée : remplacer l'image mémoire du processus par une autre
- Les primitives exec vont nous permettre de faire ça.

Remarque

- Attention, lorsque l'on appelle une primitive exec celle-ci ne se terminera pas.
- Un processus qui appelle une primitive exec ne changera pas d'identité.

Parmètres

- Un programme
- Des paramètres
- Des variables d'environnement.

Example

```
{\subsectitle}  
int main(int argc, char * argv[], char * arge[]);
```

man exec

NAME

execl, execlp, execl, execv, execvp - execute a file

SYNOPSIS

```
#include <unistd.h>
```

```
extern char **environ;
```

```
int execl(const char *path, const char *arg, ...);  
int execlp(const char *file, const char *arg, ...);  
int execl(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[]);
```

Différences

- Recherche du programme :
 - vide : répertoire en cours (execl, execv, execl)
 - p : path (execlp, execvp)
- Arguments du programme :
 - vide : tableau (terminé par NULL)
 - l : liste explicite (terminée par NULL)
- Utiliser :
 - vide : environnement courant
 - e : environnement sous forme de tableau (terminé par NULL)

Fonctionnement

- Rechercher le fichier pointer par arg1
- Vérifier que le fichier est exécutable (permissions)
- Vérifier dans l'entête que le fichier peut être chargé
- Copier les paramètres de l'exec courant dans l'espace système
- Détacher toutes les régions du programme
- Allouer de nouvelles régions
- Attacher les nouvelles régions
- Charger la région en mémoire
- Copier les paramètres de l'exec dans la pile utilisateur.

Ne pas oublier

- Sauf si il échoue à se lancer, on ne revient pas d'un exec
- Tout le code écrit après un exec qui a réussi n'existe plus ! (écrasement)

Contrôle

- `pid_t wait(int *status);`
- `pid_t waitpid(pid_t pid, int *status, int options);`
- `waitpid(-1, &status, 0);`
- Comment exploiter la variable status.

Fin de processus

- Fonction de tests
- `WIFEXITED(status)` : indique si un processus s'est terminé correctement
- `WEXITSTATUS(status)` : code d'erreur retourné par le fils (`exit/return`)
- `WIFSIGNALED(status)` : vrai si fils terminé par un signal
- `WTERMSIG(status)` : renvoi le numéro du signal qui a terminé le fils

Fin de processus

Communication intrasèque

- Comment communiquer avec les fils ?
- Redirection d'entrée standard et de sortie standard (dup)
- Utilisation des pipes nommés
- Utilisation de popen (pipe + fork + shell)

Divers

- `int system(const char *command);` si on veut juste exécuter une commande.
- `vfork` : système dédié nécessitant performance -> père bloqué tant que le fils existe et n'a pas fait un `exeve`
- Dans le cas du `vfork`, il y a un partage de mémoire temporaire. A utiliser avec extrême précaution.
- Si on veut travailler avec des mémoires partagées, il existe `clone`
- `Clone` est utilisé pour créer des threads.