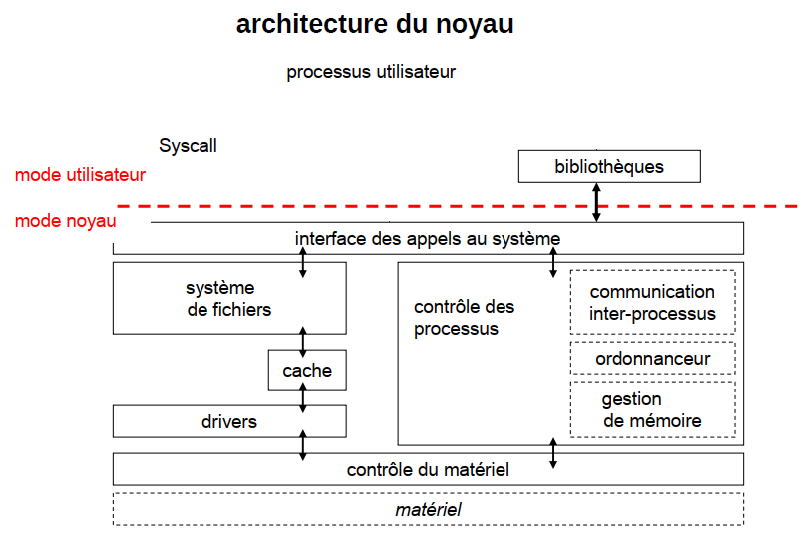
**Système Unix**

* Architecture du système
* Processus
* Fonctions sur les processus

**Synchronisation et communication**

* Signaux
* Verrous
* Sémaphores
* Tubes
* Messages
* Mémoire partagée
* Mappage de fichiers en mémoire principale

**Architecture du noyau**



**Fonctions du noyau**

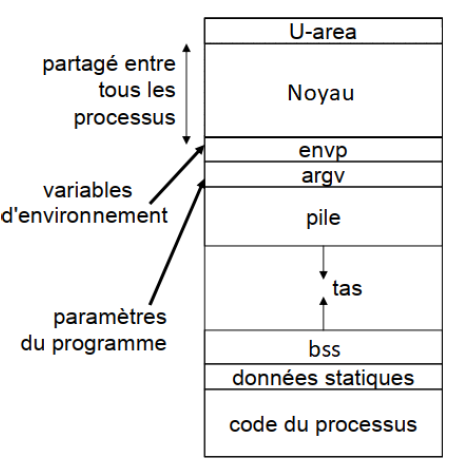
* Création des processus
* Ordonnancement
* Communication entre processus
* Contrôle des interruptions
* Gestion de la mémoire
* Gestion du système de fichiers
* Gestion des périphériques

**Processus du noyau**

Un processus est composé de :

* un espace d'adressage
* un ou plus de threads
* un ensemble de ressources (fichiers ouverts, etc.)

**Espace d’adresses du processus**



**User area : U-area :**

Contient les informations nécessaires sur le processus pendant son exécution

* sauvegarde du contexte hardware
* UID et GID réels et effectifs
* les gestionnaires des signaux
* descripteurs des fichiers ouverts
* répertoire courant
* le terminal de contrôle
* utilisation CPU, etc
* pile "noyau" du processus pour les appels système
* pointeur sur la structure proc

**Structure proc**

* identificateur du processus (PID)
* pointeur sur U-area du processus
* priorité du processus
* informations pour la gestion des signaux (signaux bloqués, à ignorer, etc.)
* information pour la gestion de la mémoire
* pointeurs vers avant et arrière de la chaîne des structures proc dans une des listes
* du noyau de processus actifs, libres ou zombies
* informations sur les processus du père, du 1er fils, ,,, (pointeurs sur proc )

**Identificateurs**

**Identification d'un processus**

* PID: identificateur unique du processus
* PPID: identificateur du processus père
* PGRP: identificateur du groupe du processus (hérité du père, permet de désigner un ensemble de processus)

**Identification de l’utilisateur (sécurité) Contrôle de l’accès au système de fichiers:**

* UID: identificateur réel (hérité du père, ex: du shell qui a lancé un programme)
* EUID: identificateur effectif (ex: exécution d’un programme avec "su")
* GID: identificateur de groupe
* EGID: identificateur de groupe effectif

**Identificateurs : Appels système**

**Identification d'un processus** : PID, PPID, PGRP, getpid( ), getppid( ), getpgrp( ), setpgrp( )

**Identification de l’utilisateur** : UID, EUID, GID, EGID, getuid( ), getruid( ), geteuid( ) setuid( ), setruid( ), seteuid( ), setreuid( ), getgid( ), getrgid( ), getegid( ), setgid( ), setrgid( ), setegid( ), setregid( )

**Fonctions de gestion de processus**

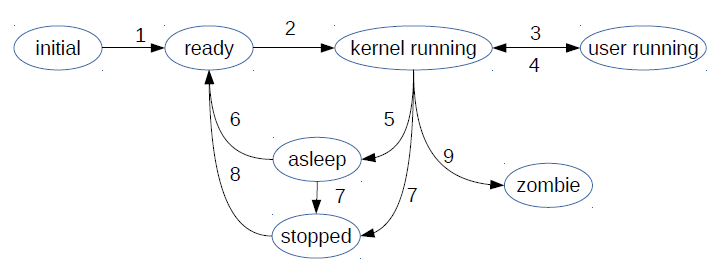
* fork
* exit
* wait
* exec

***Fonction exit***

void exit (int status);

* termine l'exécution d'un processus le paramètre status est un code de terminaison, qui est fourni au processus-père status = 0 indique une terminaison normale
* l'appel exit(...) fait passer le processus dans l'état zombie. Il restera dans cet état jusqu'à ce que son processus-père ait lu status. A ce moment le processus est terminé et disparaît complètement du système
* si le père d’un processus se termine, alors ce processus est adopté par le processus init. càd que init devient le père de ce processus.

**États de processus**



**Transitions de processus**

1. le processus a acquis les ressources nécessaires à son exécution
2. le processus vient d'être élu par l'ordonnanceur : il y a alors changement de contexte
3. le processus revient d'un appel système ou d'une interruption (quand il est élu ou quand il a terminé l'exécution du handler d'une interruption)
4. le processus a réalisé un appel système ou une interruption est survenue
5. le processus se met en attente d'un événement: appel système bloqué par un événement externe (interruption) ou interne au système (libération de ressource ou terminaison d'un processus, etc)
6. l'événement attendu par le processus s'est produit
7. délivrance d'un signal particulier (SIGSTOP ou SIGTSTP)
8. réveil du processus par le signal SIGCONT
9. le processus se termine

Lorsqu'un processus est endormi, il est en attente d'un événement.

Les événements sont représentés par une adresse mémoire dont l'état est modifié par l'occurrence de l'événement associé.

Une caractéristique importante d'Unix est que l'occurrence d'un événement fait repasser de l'état endormi à l'état prêt tous les processus en attente de cet événement.

***Fonction wait***

pid\_t wait(int \*status);

* suspend l'exécution du processus jusqu'à ce qu’un de ses fils se termine.

Exemple: ret = wait(&status)

* si le processus n'a aucun fils, l'appel se termine instantanément, et retourne la valeur –1
* s'il existe des fils, mais aucun dans l'état zombie, l'appel est bloquant. Dès qu'un fils est dans l'état zombie, l'appel se termine et retourne le PID de ce fils status contient le code de terminaison du fils = paramètre de l'appel exit( ) du fils
* pid\_t = int (process id type) #include <sys/types.h>

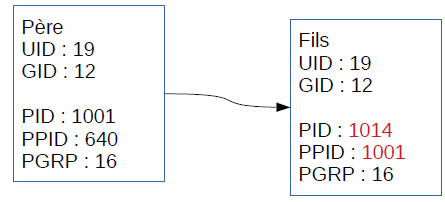
***Fonction fork***

pid\_t fork(void);

Création de processus

* fork crée un processus identique à l'appelant: même code, même pile (copiée), même zone de données (copiée si écriture), même U-area (copiée) sauf: PID différent, valeur retournée par l'appel, valeur retournée au fils: 0, valeur retournée au père: PID du fils, en cas d'erreur: -1
* pid\_t = int (process id type) #include <sys/types.h>
* UID, GID hérité par le fils
* PPID: PID du père
* PGRP: groupe PID, hérité par le fils

*Exemple :*



***Fonction exec***

int execv (char \*path, char \*argv[]);

Changement du segment de code :

* path est le nom du fichier contenant le code à exécuter
* argv[] est la liste des arguments à passer au code à exécuter
* La 1ère instruction exécutée après execv est la 1ère instruction du code désigné par le paramètre path
* permet de changer le code du processus
* variantes :

1. selon le mode de passage des arguments (vecteur ou liste),
2. selon la présence ou non de l’argument path,
3. selon le passage ou non d’un nouvel environnement au processus

* effet:

1. charge le nouveau code du processus,
2. réinitialise la pile,
3. réinitialise la zone data.

La zone U-area n'est (presque) pas touchée (uid et gid effectifs, et handlers des signaux)

les variables d'environnement restent inchangées (sauf dans la troisième variante)

* pas de retour de l'appel, sauf si l'appel échoue

***Exemples de gestion de processus***

|  |  |  |  |
| --- | --- | --- | --- |
| 1. | 2. | 3. | 4. |
| 5. | 6. |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |