UE NSY103

Introduction au Noyau Linux

Première Partie
Processus & Ordonnanceur

I- Le Noyau Linux

II- Le Concept de Processus

III- Les Threads

IV- L'Ordonnancement

V- L'Ordonnanceur Temps Partagé

VI- Les Ordonnanceurs Temps Réel

I- Le Noyau Linux

Le noyau Linux comporte :

- **Les ordonnanceurs qui permettent d'exécuter un programme ou plusieurs programmes de façon simultanée (multitâche) ,**
- **Les routines d'interruption qui sont des programmes qui se lancent en réponse à une interruption,**
- **Les bibliothèques de primitives système qui sont appelées par les programmes utilisateurs pour accéder aux ressources de la plateforme.**

** Le noyau Linux de référence est le "Vanilla Kernel" maintenu par l'équipe de Linus Torwald.*

- En dehors des ordonnanceurs, le noyau Linux ne prend la main que lorsqu'une interruption est levée ou lorsqu'un programme utilisateur effectue un appel à une primitive système.
- Le noyau fait appel à quelques programmes en tâche de fond (c'est-à-dire qui ne s'exécutent que lorsque le processeur est disponible), pour réaliser des tâches spécifiques système : Réorganisation de la mémoire virtuelle, gestion de la mémoire cache etc.

I- Le Noyau Linux

Les Niveaux d'Exécution

Sur les plateformes PC, les processeurs de la famille Intel x86 supportent 4 niveaux d'exécution : Ring 0 (avec tous les privilèges) à Ring 3 (avec privilèges limités).

Le système Linux (le système Windows également) n'utilise que 2 des 4 niveaux :

- Mode Noyau : Ring 0.
- Mode Utilisateur : Ring 3 (privilèges limités).

- Le noyau Linux fonctionne exclusivement en mode noyau.

- Les programmes utilisateur s'exécutent en mode utilisateur, mais lorsqu'ils feront appel à une primitive système, le code de cette primitive sera exécuté en mode noyau.

- En mode utilisateur, un programme ne peut pas masquer les interruptions et donc en particulier ne peut pas empêcher l'ordonnanceur de l'interrompre.

I- Le Noyau Linux

La Réentrance

En mode noyau hors section critique (interruptions masquées) si une interruption apparaît alors le noyau s'interrompt lui-même, le noyau est donc réentrant.

- il peut fonctionner en plusieurs niveaux d'imbrication.

Cette réentrance est indispensable, certaines interruptions matérielles doivent être traitées de façon quasi immédiate pour ne pas perturber le périphérique correspondant.

- C'est la pile noyau qui gère l'empilement des contextes des différents niveaux d'imbrication du noyau.

II- Le Concept de Processus

Un processus est une unité d'exécution élémentaire. Un programme (ou application) comporte un ou plusieurs processus interconnectés.

- On distinguera les processus à mémoire privée (ou processus tout court) et les processus à mémoire partagée dits processus légers (ou thread en anglais)

Un processus à mémoire privée possède :

- Un programme en langage machine en mémoire
- Un espace mémoire privé
- Une pile d'exécution

II- Le Concept de Processus

Espace mémoire des Processus

Zone mémoire	Contenu
Environnement	- Variables système - Variables d'environnement - Arguments du processus.
Heap (Tas)	Données globales dynamiques
Stack (Pile)	- Données locales dynamiques - Arguments des sous-programmes - Contextes d'exécution des sous-programmes
Bss[5]	Données statiques non initialisées
Data	Données statiques initialisées
Text	- Code source compilé en langage machine - Constantes

II- Le Concept de Processus

La Protection

L'espace mémoire des processus à mémoire privée est protégé, cette protection est forte car elle relève du mode noyau et de composants matériels : MMU (Memory Management Unit) et MPU (Memory Protection Unit).

Un processus à mémoire privée ne peut donc pas accéder à l'espace mémoire d'un autre processus à mémoire privée.

- La zone **Text** est protégée en écriture, les zones **Data** et **Bss** sont protégées en exécution
- La taille des 3 zones Code, Données et BSS est fixée à la compilation, l'option **-m64** du compilateur C permet d'afficher ces tailles (cf. exercices)
- La taille de la Pile est fixe pour tous les processus elle peut être consultée et modifiée par la commande **ulimit -s**
- Le Tas est extensible en fonction des demandes d'allocation mémoire dynamiques et cela tant qu'il reste de la mémoire dans le système.

III- Les Threads

Un processus à mémoire privée peut être partagé en 2 ou plusieurs processus à mémoire partagée, dits processus légers ou threads.

Tous les threads d'un même processus se partagent l'espace mémoire du processus, à l'exception de la zone "Pile" : Chaque thread possède sa propre pile (un thread est une unité d'exécution indépendante).

Un processus léger possède :

- Un programme en langage machine en mémoire (programme partagé avec les autres threads du même processus).
- Un espace mémoire partagé avec les autres threads du même processus
- Une pile d'exécution privée

Les threads d'un même processus peuvent communiquer entre eux par l'intermédiaire de leurs données qui sont communes (**Data**, **Bss** et **Tas**), mais en contrepartie un thread peut endommager les données d'un autre thread du même processus.

III- Les Threads

Threads vs Processus

- ✓ **Choisir une application en mode multiprocessus c'est privilégier la sécurité** : Protection maximale.

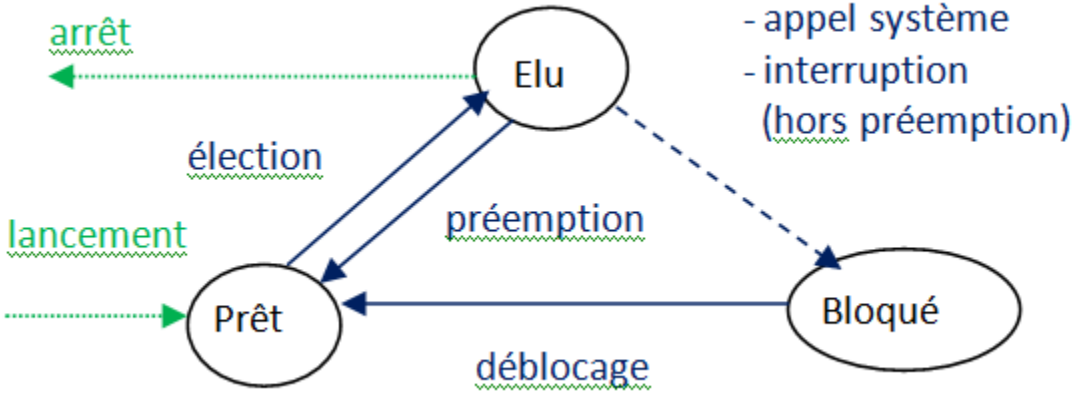
- ✓ **Choisir une application en mode multithread c'est privilégier les performances :** Communication inter-thread facilitée et mécanismes de protection mémoire moins couteux en ressources système.

** Dans tous nos supports de cous nous utiliserons le terme générique de "processus" pour désigner une unité élémentaire d'exécution qu'il s'agisse de processus ou de thread, sauf lorsque le contexte nous obligera à distinguer les 2 cas.*

IV- L'ORDONNANCEMENT

Les ordonnanceurs Linux sont tous de type "préemptif", ils peuvent interrompre le processus en cours d'exécution pour répartir le temps CPU en fonction de leur stratégie.

Etats des Processus



IV- L'Ordonnancement

Processus Elu et Préemption

Le processus Elu est celui qui est en cours d'exécution. Le processus Elu conserve la main jusqu'à la fin de son quantum de temps d'exécution, sauf s'il est préempté par l'ordonnanceur dans les cas suivants :

- Un processus plus prioritaire que lui passe à l'état Prêt, auquel cas c'est le processus plus prioritaire qui sera immédiatement élu et le processus moins prioritaire retournera à l'état Prêt.
- Il fait appel à un service du noyau (appel système), auquel cas il pourrait passer à l'état Bloqué, cela dépendra du service appelé ainsi que du contexte.
- Une interruption matérielle ou une exception est levée, dans ce cas il pourrait passer à l'état Bloqué, cela dépendra de l'interruption levée ainsi que du contexte.

Attention : Lorsqu'un processus exécute du code noyau, il ne peut pas être préempté, il pourrait cependant être interrompu par un autre niveau du noyau et reprendre la main juste après. Nous verrons par contre qu'en fonctionnement multiprocesseur un processus pourra être préempté durant l'exécution de code noyau.

IV- L’Ordonnancement

Processus Prêt

C’est un processus Prêt à être exécuté.

- Ce sont les ordonnanceurs (cf. chapitre suivant) qui vont décider lequel de ces processus prêt va être le processus Elu suivant.

Processus Bloqué

Un processus Bloqué est un processus en attente de ressource système.

- Il retournera à l'état Prêt en fin de blocage.

V- L'Ordonnanceur Temps Partagé

Linux dispose d'un seul ordonnanceur de type temps partagé : SCHED_OTHER, c'est l'ordonnanceur par défaut des processus.

Priorités temps partagé

Ce que l'on appelle les priorités temps partagé, ne sont pas vraiment des priorités, ce sont les "quantum" de temps partagé attribués aux processus et on les qualifie de **nice** (**nice -20** à **nice +19), l**a priorité **nice -20** représente la priorité la plus grande et donc le quantum le plus grand.

Lors de sa création un processus temps partagé se voit attribuer le **nice** de son père (son processus créateur) qui est par défaut **0**, ensuite ce **nice** évolue inversement proportionnellement à la consommation de CPU par le processus, ainsi les processus faisant souvent appel aux entrées/sorties ne seront pas pénalisées par rapport aux tâches de purs calculs.

** Cf. en exercices comment modifier le quantum de temps partagé*

V- L’Ordonnanceur Temps Partagé

Stratégie SCHED_OTHER

Les processus temps partagé s'exécutent dans l'ordre "premier arrivé, premier servi".

Un processus temps partagé sera préempté :

- S'il se présente un processus temps réel prêt (cf. chapitre correspondant).
- S'il arrive à la fin de son quantum de temps partagé.

Un processus temps partagé pourrait être bloqué :

- S'il fait appel à un service du noyau (appel système).
- Si une interruption matérielle ou une exception est levée.

VI- Les Ordonnanceurs Temps Réel

Linux propose 2 ordonnanceurs temps réel SCHED_FIFO et SCHED_RR

C'est l'appel système **sched_setscheduler()** qui permet de rattacher un processus à un ordonnanceur temps-réel.

** Il faut disposer des privilèges **root** pour rattacher un processus à un ordonnanceur temps réel.*

Priorités temps réel

Les priorités temps réel varient de **0** à **99** et c'est la plus petite valeur qui correspond à la priorité la plus grande (**0** = plus grande priorité).

Les processus temps partagé ont l'équivalent d'une priorité temps réel de **100** (priorité strictement inférieure à celle de tous les processus temps réels).

VI- Les Ordonnanceurs Temps Réel

Stratégie SCHED_FIFO

*(**FiFo** pour « First In First Out »)*

C'est le processus de plus forte priorité qui est élu. A priorités égales c'est la règle FIFO ou "premier arrivé = premier servi" qui est appliquée.

Il n'y a pas de notion de "quantum" de temps d'exécution, un processus n'est préempté que s'il se présente un processus de priorité supérieure.

Un processus pourrait être bloqué :

- S'il fait appel à un service du noyau (appel système).
- Si une interruption matérielle ou une exception est levée.

*Attention : Ne pas confondre les priorités temps réel et les priorités temps partagé **nice**.*

VI- Les Ordonnanceurs Temps Réel

Stratégie SCHED_RR

*(**RR** pour "Round-robin" ou politique du tourniquet)*

Stratégie identique à celle de SHED_FIFO mais avec quantum de temps réel.

- C'est le processus de plus forte priorité qui est élu. A priorités égales c'est la règle "premier arrivé = premier servi" qui est appliquée.
- Un processus est préempté :
  - s'il se présente un processus de priorité supérieure.
  - s'il arrive à la fin de son quantum de temps réel.
- Un processus pourrait être bloqué :
  - s'il fait appel à un service du noyau (appel système).
  - si une interruption matérielle ou une exception est levée.

*Attention : Ne pas confondre les quantum temps réel et les quantum temps partagé **nice**.*

Seconde Partie
Interruptions & Signaux

VII- Les Interruptions Microprocesseur 80X86

VIII- Les Interruptions Linux

IX- Les Signaux Linux

VII- Les Interruptions Microprocesseur 80X86

Vous trouverez les interruptions microprocesseur 80X86

matérielles et logicielles

dans le support de cours « Les Plateformes PC ».

VIII- Les Interruptions Linux

Linux gère théoriquement 256 interruptions numérotées de 0 à 255.

Linux distingue 4 catégories d'interruptions :

- **Interruptions matérielles masquables** (correspondent à INTR)
- **Interruptions matérielles non masquables** (correspondent à NMI)
- **Exceptions** (correspondent à TRAP)
- **Interruptions logicielles** (instruction correspondent à INT)

La table des vecteurs d'Interruptions

- Il s'agit d'une table qui contient un descripteur (ou vecteur) par interruption*, cette table fait partie du fonctionnement des plateforme PC.
- Le descripteur d'interruption contient :
  - l'adresse en mémoire du gestionnaire (ou routine) d'interruption.
  - un indicateur masquable / non-masquable.
  - le mode minimal autorisé pour lancer cette interruption par programmation (instruction int).

VIII- Les Interruptions Linux

Stratégie

Si une interruption est levée :

- **Elle ne sera prise en compte qu'à la fin de l'instruction en cours (une interruption n'interrompt pas l'instruction courante).**
- A la fin de l'instruction en cours, si les interruptions ne sont pas masquées, le gestionnaire d'interruption correspondant est lancé :

  1) Stocker le contexte d'exécution dans la pile noyau : registres d'état et compteur ordinal.

  2) Mettre dans le compteur ordinal l'adresse du gestionnaire d'interruption.

  3) Exécuter le gestionnaire jusqu'à l'instruction iret (retour d'interruption x86)

  4) Restaurer le contexte précédent à partir de la pile du noyau.

- Sinon l'interruption est mémorisée en attendant le démasquage.

VIII- Les Interruptions Linux

Stratégie (suite)

Il n'y a pas de priorité entre les interruptions Linux :

- Une nouvelle interruption interrompt l'éventuel gestionnaire d'interruption en cours (réentrance), sauf s'il y a masquage des interruptions.
- En cas de mémorisation (suite à un masquage) les interruptions seront reprises dans l'ordre premier arrivé premier servi.

➢ *Si l'on manque d'interruptions on peut installer plusieurs ISR (Interrupt Service Routine) à l'intérieur d'un gestionnaire d'interruption, ils seront tous activés et chaque ISR vérifiera s'il est concerné avant de traiter l'interruption*

VIII- Les Interruptions Linux

INT 0-31	Int	Masquable	Ring	Source
TRAP & NMI (Interruptions Matérielles non masquables)	0	non	0	Division Overflow
	...			
	2	non	0	NMI
	...			
	4	non	0	Overflow
	...			
INT 32-47	**Int**	**Masquable**	**Ring**	**Correspondance Contrôleur PC**
INTR (Interruptions Matérielles masquables)	32	oui	0	IRQ0 System Timer
	33	oui	0	IRQ1 Keyboard
	...			
	40	oui	0	IRQ8 Real Time Clock
	...			
	44	oui	0	IRQ 12 Mouse
	...			
	46	oui	0	IRQ 14 Hard Drive Controller
	...			
INT 48-255	**Int**	**Masquable**	**Ring**	**Source**
(Interruptions logicielles masquables)	...			
	128	oui	3	Appel système
	...			

IX- Les Signaux Linux

Linux gère un système logiciel qui suit le modèle des interruptions : Un signal est un message court (identifiant sans contenu) que l'on peut envoyer, masquer, traiter ou ignorer et cela entre processus ou du noyau vers les processus.

- A la différence des interruptions les signaux sont donc accessibles aux processus en mode utilisateur.
- Grâce aux signaux, les processus vont pouvoir se synchroniser ou se contrôler : Suspension, reprise, arrêt.
- Grâce aux signaux le noyau va pouvoir relayer certaines exceptions aux processus pour leur permettre de les traiter applicativement (traitements messages, log ...) ou leur envoyer des "top" de synchronisation.
- Attention : Pour qu'un processus puisse envoyer un signal à un autre processus il faut qu'il ait les privilèges root ou qu'il ait été lancé par le même utilisateur que le processus auquel il souhaite envoyer ce signal.

IX- Les Signaux Linux

Signaux de suspension, reprise et arrêt des processus

- Le signal 15 SIGTERM, masquable permet de tuer un processus.
- Le signal 9 SIGKILL, non-masquable permet de tuer un processus sans que ce dernier puisse se protéger (masquer le signal).
- Le signal 2 SIGINT, masquable est l'équivalent du Ctrl/C qui permet à l'utilisateur de tuer le processus en avant plan.
- Le signal 17 SIGTSTOP non-masquable permet de suspendre un processus sans que ce dernier ne puisse se protéger (masquer le signal).
- Le signal 18 SIGTSTP, permet de suspendre un autre processus.
- Le signal 19 SIGSCONT, permet de faire reprendre un processus suspendu.

EXERCICES

Vous pensez avoir bien assimilé les concepts présentés dans ce cours.
Vous devez alors passer aux exercices, ce sont eux qui
vous permettront de valider vos connaissances .