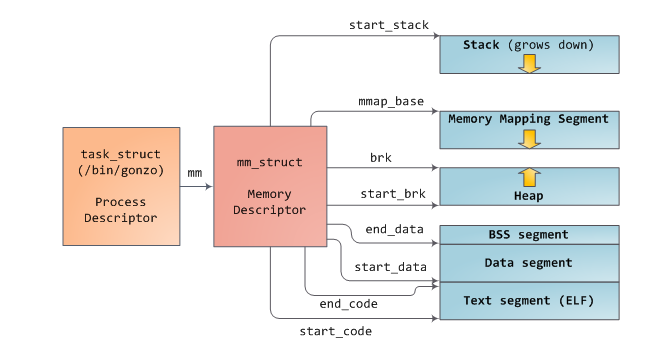
Descripteur de processus : task\_t (struct task\_struct : <linux/sched.h>)

* PC, SP…
* CPU Status, processor registers…
* PID : jusqu’au 32767 ou 32768
* Etat : running, sleeping, zombie…
* Struct task\_struct\* parent : pointeur sur son parent
* Struct list\_head children : liste des enfants : la structure contient un next et un prev qui sont des pointeurs vers les prochains/précédents list\_head
* Table des pages ?
* Process adress space
* Pending signals
* Open files
* Prio
* Se : scheduler entity rattaché à ce processus

Espace d’adressage d’un processus :



Allocation d’un descripteur de processus :

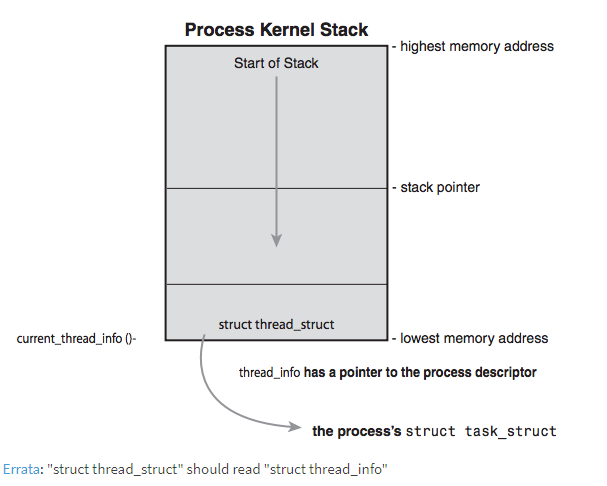
Slab allocator alloue des structures task\_struct.

Manipulation des descripteurs de processus :

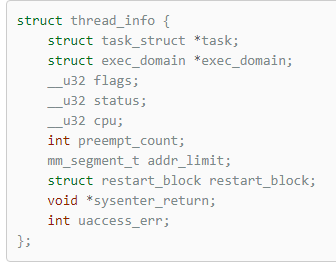
Les Descripteurs de processus sont dans une liste doublement chainée circulaire : la task list

Pour retrouver rapidement le process en exécution :

* La structure thread\_info (en bas de la stack kernel) pointe sur le task\_struct en cours d’exécution.
* Quelque fois, à la place du thread\_info, on met le task\_struct en exécution dans un registre



Current est une macro qui désigne le processus en exécution et Current\_thread\_info () retourne le thread\_info (en masquant les 13 LSB du SP), donc le current->task.



Set\_task\_state (task, state)  changer l’état d’un processus

Task->state = state ;  idem

Set\_current\_state (state) ; idem pour le current

List\_for\_each (pos, head)  Parcours d’une liste doublement chainée.

List\_entry () Retrouver un objet plus grand qui en contient un autre

Next\_task (task) Macro utilisant list\_entry ()

Prev\_task (task) Idem

For\_each\_process (task) Macro parcourant la liste complète.

Espace mémoire d’un processus :

* + Pile
  + Tas (Malloc…)
  + Biblios
  + Data
    - Initialisées
    - Non initialisées
  + Programme

Naissance d’un processus :

* Fork () (et vfork () et \_\_clone () ) implémentés via clone () qui lui-même va appeler do\_fork() qui va appeler copy\_process ().

Fork () va utiliser le COW (copy on write) : permet de différer la copie du père : au lieu de dupliquer l’espace d’adresse du père, on le partage au début.

Chaque data qui a besoin d’être changée par le fils va être marquée et le fils recevra une unique copie de la data.

Clone () va utiliser une série de flags pour savoir quelles ressources le parent et le fils vont partager.

Copy\_process () va :

1. Appeler dup\_task\_struct () qui va créer une kernel stack, un thread\_info et un task\_struct (Ici les task\_struct père et fils sont identiques).
2. Check si le nouveau fils ne va pas dépasser les ressources autorisées du père
3. Initialiser ou RST certains paramètres du descripteur de processus.
4. Etat mis à TASK\_UNINTERRUPTIBLE
5. Appeler copy\_flags ()
6. Appeler alloc\_pid ()
7. En fonction des flags passés à clone (), dupliquer certains trucs du père (si c’est un thread par exemple).
8. Finalement renvoi un pointeur vers le nouveau fils.
9. Do\_fork() va réveiller le fils.

Vfork () identique à fork () mais table des pages non copiée : parent bloqué jusqu’à ce que le fils appel exec () et fils ne peut pas écrire sur l’espace d’adressage.

Donc fork () peut créer des problèmes de refus de création de fils : si le père possède 2 Gb et que le système ne possède pas 2 Gb supplémentaire, fork () échouera.

Vfork () lui ne copiera pas la table des pages et attendra exec () donc vfork () plus rapide et pas de problème de mémoire mais problèmes de sécurité.

* Exec () : Crée un nouvel espace d’adresses et load un nouveau programme en mémoire

Mort d’un processus :

* Exit () qui va appeler do\_exit ()

Qui va entre autre libérer la mémoire, les IPCs, les fichiers, stocker le retour, envoyer un signal au parent, reparenter ses fils et appeler le scheduler. Il est dans l’état zombie.

Le parent meurt -> le fils devient enfant d’un des threads du groupe et sinon d’Init.

Le descripteur de processus est libéré si :

* Le parent a obtenu des infos sur son fils terminé
* Le parent notifie le kernel qu’il s’en bat les couilles du gamin

Release\_task () est appelé pour libérer le task\_struct.

Init appel périodiquement wait() sur ses fils afin de libérer les zombies.

Scheduler :

Distingue deux types de processus : les liés aux E/S et ceux liés au processeur. Le scheduler va plutôt favoriser les E/S.

Temps de CPU d’un processus dépend de son comportement (Utilisation du CPU de manière massive ou non) et de comment on la configurer (nice…)

Yielding : le fait qu’un processus décide de laisser la main de lui-même (real time processus)

Implémentation de runqueues au niveau de chaque CPU.

**Linux n’affecte pas de temps CPU aux processus mais une proportion de CPU.**

Chaque processus qui run voit sa proportion CPU diminuer.

Le scheduler est modulable : différents algorithmes peuvent être appliqués à différents processus.

Le CFS (completely fair scheduler) de linux est attribué aux processus normaux : SCHED\_NORMAL ou SCHED\_OTHER.

Scheduler RT : SCHED\_FIFO et SCHED\_RR

Si un autre scheduler à une plus grande priorité et qu’il a un processus prêt, il prend la priorité.

Un processus possède un minimum de temps d’utilisation CPU : la granularité minimum (1 ms) pour éviter des changements de contexte trop fréquents.

Lorsqu’un processus tourne, il possède un timeslice : décrémenté à chaque tick d’horloge.Quand ça arrive à 0, le processus est préempté.

Schedule () va chercher le scheduler de plus grande priorité et lui demander quel processus doit prendre la main (rappel : CFS est le scheduler des processus normaux).

Le scheduler est appelé lors d’un retour en user space d’un system call ou d’une interruption

Priorité d’un processus :

Priorité supérieure : le processus va runner avant les autres alors que si plusieurs processus ont la même prio : round robin.

2 types de priorités implémentés par Linux :

* Nice value : de -20 à 19 permet de modifier la proportion de temps CPU accordé.

Les processus normaux ont une prio de 100 à 139 donc.

* RT priority : de 0 à 99 : permet de faire passer des processus avant d’autres.

Sleeping et waking up :

On s’endort toujours sur une condition et on se réveil sur cette même condition.

Context switching :

Schedule () appel context\_switch () qui va :

* Appeler switch\_mm ()
* Appeler switch\_to () : sauvegarde des registres et pile et restauration pour le nouveau.

Préemption :

Tant que le kernel n’est pas en mode bloqué (région de code critique), il peut être préempté.