Temps réel sous Linux

Ordonnancement - Temps partagé

Christophe Blaess

christophe.blaess@logilin.fr

https://www.blaess.fr/christophe/
twitter: @chrisblaess

Ingénierie et formations sur Linux et les logiciels libres
https://www.logilin.fr

Ce cours a été rédigé en utilisant des logiciels libres sur système d'exploitation Linux :

- *LibreOffice Writer* pour le support et la mise en page
- *LibreOffice Draw* pour les dessins vectoriels
- *Gimp* pour les images bitmap

Temps réel sous Linux

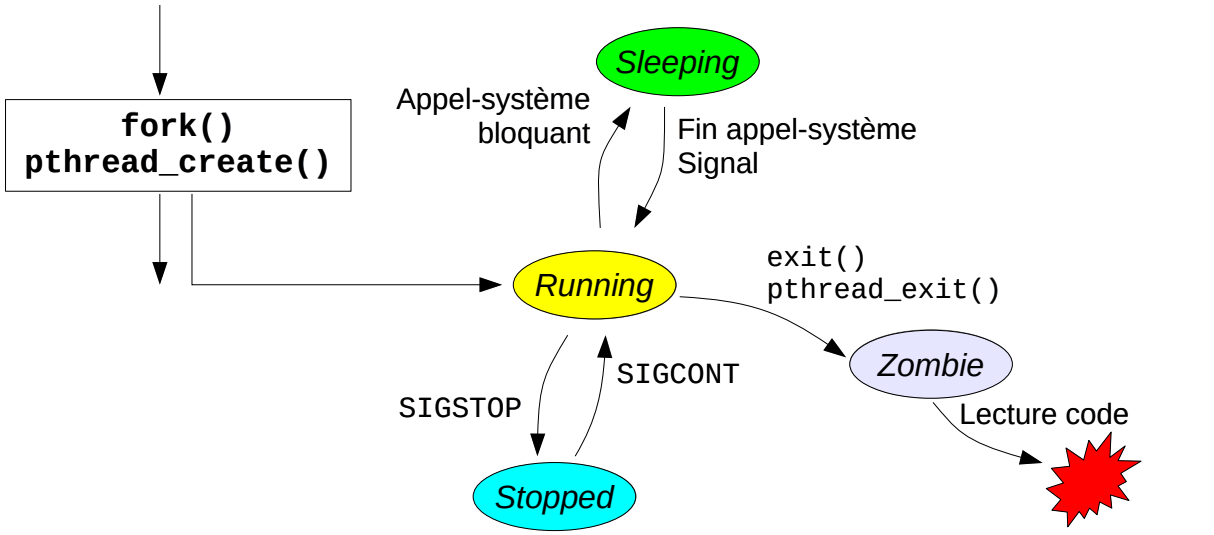
v. 5.3

https://www.blaess.fr/christophe/

https://www.logilin.fr

Ordonnancement

État des tâches




```
ps aux :
R – running
S – sleeping
T – stopped (traced)
Z – zombie
```

De sa création à sa disparition, une tâche peut prendre les états suivants :

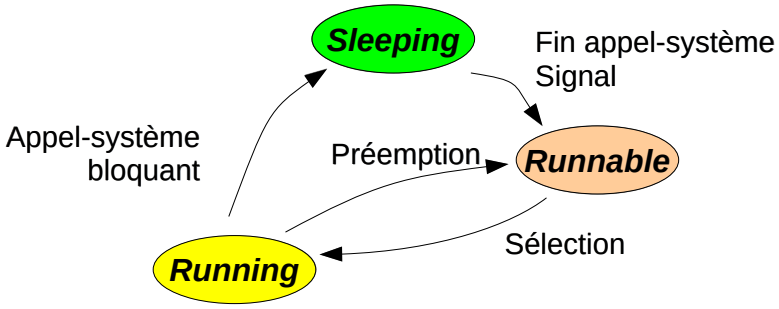
- *Running* : en cours d'exécution
- *Sleeping* : endormie en attente d'un événement externe
- *Stopped* : gelée temporairement (débogage par exemple)
- *Zombie* : terminée mais il reste son code d'état de terminaison.

Les transitions entre les états sont gérés par le noyau lors d'un appel-système ou à la réception d'une interruption.

Commutation de tâches

Pour pouvoir supporter plusieurs tâches simultanément, le noyau va partager entre elles la puissance CPU disponible. Il fera commuter les tâches régulièrement par l'intermédiaire d'un mécanisme appelée ordonnanceur (*scheduler*).

Lorsqu'une tâche est suspendue en attente d'accès au processeur, elle va se trouver dans un état supplémentaire : *Runnable* (prête).

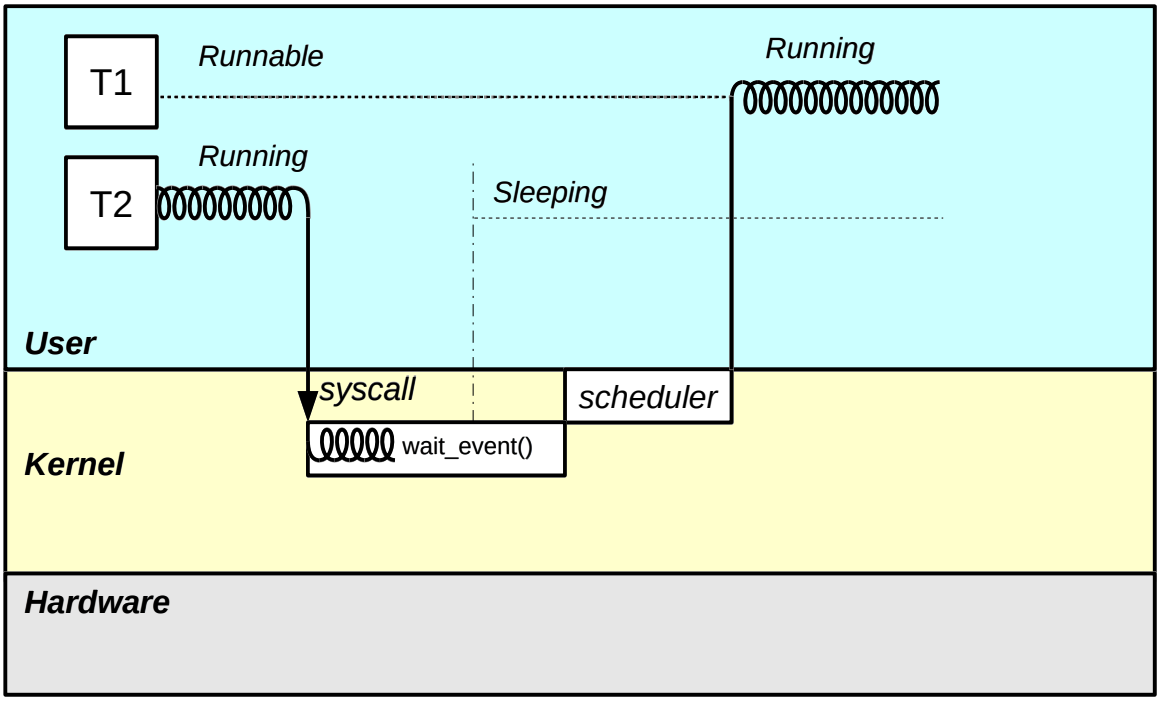


Si le noyau est capable d'interrompre de force l'exécution d'une tâche pour la passer dans l'état *Runnable*, on parle de multi-tâche *préemptif*.

Le nombre de tâches prêtes à un instant donné est nommé : *charge* du système. Celui peut être lu avec la commande shell `uptime`.

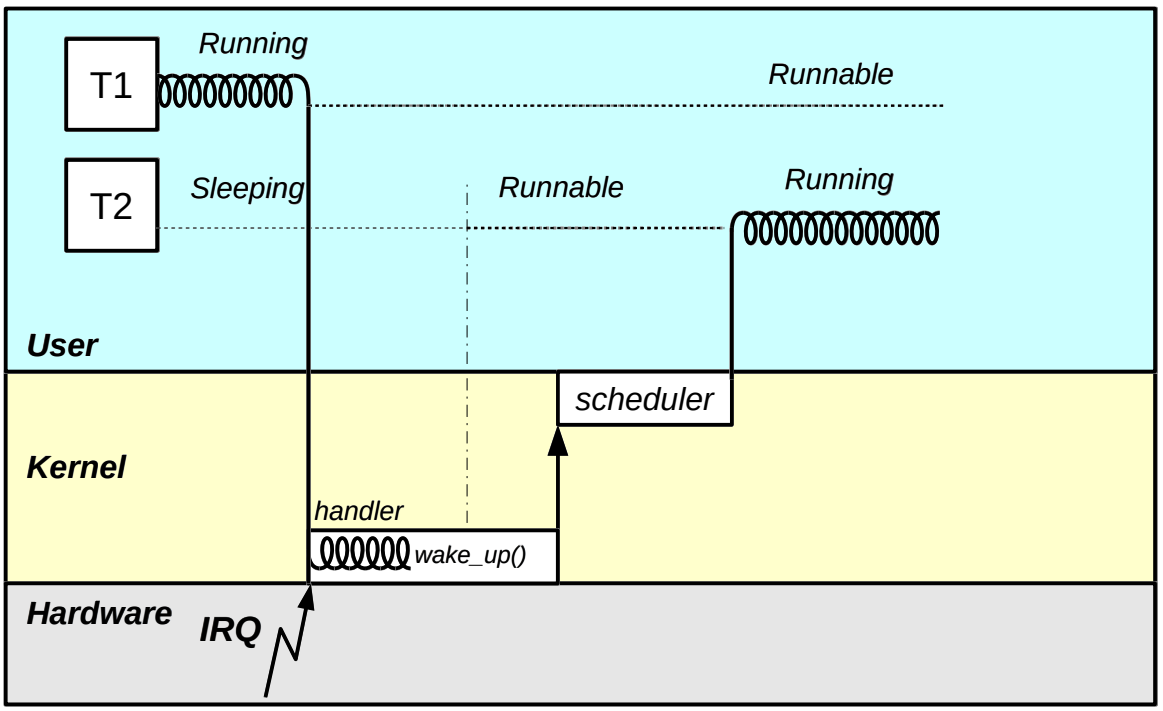
Il y a trois situations de commutation entre tâches actives :

Mise en sommeil d'une tâche



T2 invoque un appel-système bloquant – `open()`, `read()`, `write()`, `recvfrom()`, `sendto()`, `ioctl()`, `select()`, etc. – et laisse le CPU à une tâche de priorité moindre qui était en attente.

Réveil sur arrivée d'interruption

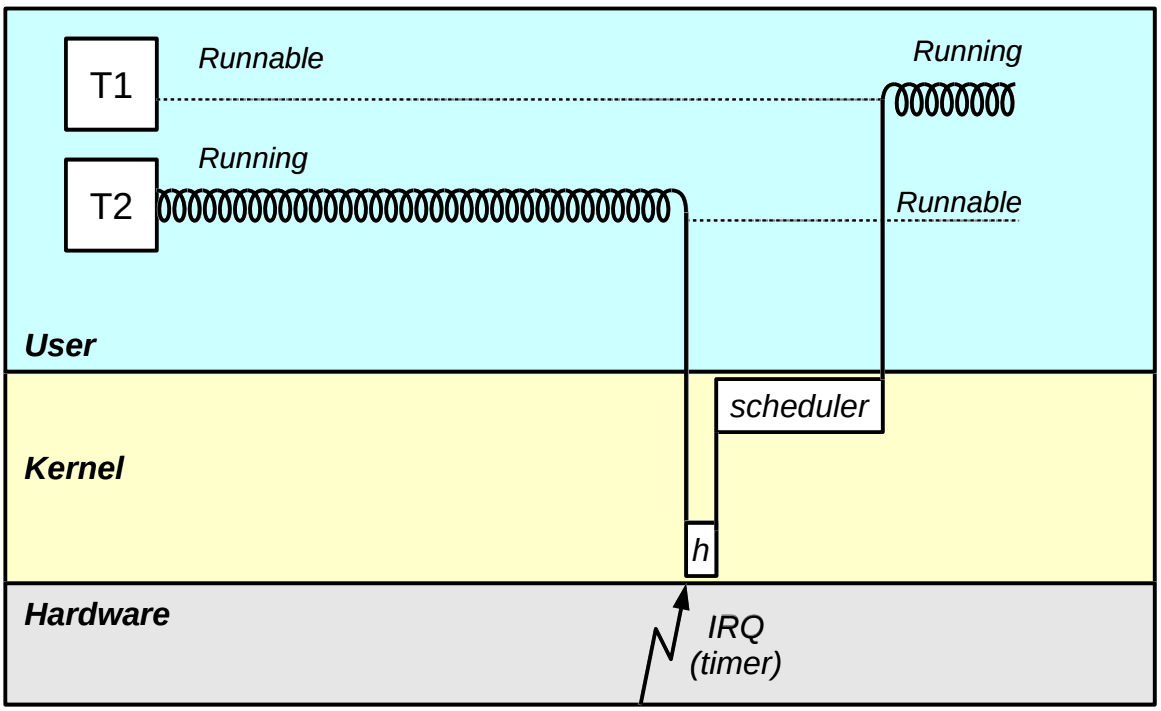


L'interruption signale que l'événement attendu par la tâche T2 s'est produit.

Elle est plus prioritaire que T1. Le noyau l'active donc, au détriment de T1.

Il s'agit de la commutation la plus fréquente sur un système interactif.

Expiration d'un timer (tranche de temps)



Lors de l'activation de T2, le noyau lui a attribué une tranche de temps (en armant un timer).

Si ce timer se déclenche avant que T2 s'endorme volontairement, elle sera préemptée pour laisser passer une autre tâche.

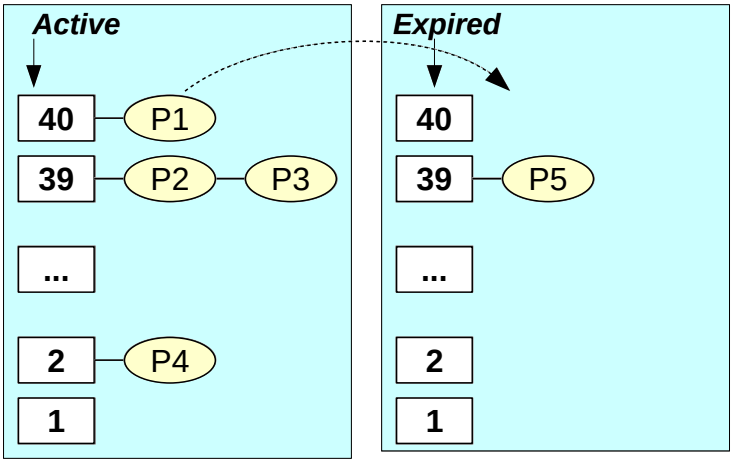
Dans un ordonnancement temps partagé, T1 peut être activée même si elle est moins prioritaire que T2. Ce n'est pas le cas avec un ordonnancement temps réel.

Ordonnanceurs *Goodness*, *O(1)* et *CFS*

L'ordonnanceur historique de Linux avant le noyau 2.6 s'appuyait sur une fonction `goodness()` qui calculait l'intérêt de chaque tâche prête et activait la plus élevée.

Inconvénient : la complexité de cet ordonnanceur est en *O(n)*.

Avec le noyau 2.6, est apparu un ordonnanceur *O(1)* s'exécutant en temps constant.



Les versions récentes du noyau s'appuient sur le CFS (*Complete Fair Scheduler*) pour un partage équitable du temps CPU.

Priorités et *nice*

L'utilisateur peut influer sur le choix de l'ordonnanceur lors de l'élection d'un processus en fixant une *priorité temps partagé* sur le processus :

```
int getpriority(int type, int ident);
int setpriority(int type, int ident, int priority);
```

La valeur de *type* dans `getpriority()` est `PRIO_PROCESS`, `PRIO_PGRP`, ou `PRIO_USER`.

La *priorite* est dans l'intervalle [-20, +19] mais elle est **inversée** : la valeur -20 correspond à la tâche la plus prioritaire, et +19 à la moins prioritaire.

```
int nice(int increment);
```

Les commandes shell `nice` ou `renice` permettent de fixer initialement et de modifier la priorité temps partagé d'un processus.

Le noyau est susceptible de favoriser des tâches interactives au détriment de tâches consommant du CPU.

Travaux pratiques : manipulation des priorités temps partagé

Le programme **`exemple-II-01.c`** prend en argument une valeur de *nice*, et modifie sa priorité. Ensuite il incrémente un compteur pendant cinq secondes et affiche le résultat obtenu.

Avant de lancer les expériences ci-dessous, exécutez `taskset -pc 0 $$` pour fixer le shell et les processus qu'il crée sur le CPU 0.

Comparez les résultats des exécutions suivantes :

```
$ ./exemple-II-01  0

$ ./exemple-II-01  10

$ ./exemple-II-01  20

$ ./exemple-II-01  0  &  ./exemple-II-01  10

$ ./exemple-II-01  0  &  ./exemple-II-01  5  &  ./exemple-II-01  10  &
   ./exemple-II-01  15  &   ./exemple-II-01  20
```