Ordonnancement et attente

M2 SETI B4 / MS SE SE758

Guillaume Duc
guillaume.duc@telecom-paris.fr
2020–2021

Plan

Tâches

- Une *tâche* est l'élément de base utilisé par l'ordonnanceur
- Elle est représentée par la structure `struct task_struct`
- Une tâche peut être dans l'un des états suivants
  - `TASK_RUNNING` : la tâche est soit actuellement en cours d'exécution, soit prête à être exécutée
  - En sommeil : elle n'est alors pas considérée par l'ordonnanceur
    - `TASK_INTERRUPTIBLE` : attente interruptible par l'événement attendu ou n'importe quel signal
    - `TASK_KILLABLE` : attente interruptible par l'événement attendu ou un signal `SIGKILL`
    - `TASK_UNINTERRUPTIBLE` : attente interruptible uniquement par l'événement attendu
  - `TASK_DEAD` : zombie

Tâches

- Lorsqu'une tâche est interrompue (interruption en provenance du matériel, d'une exception ou d'un appel système), le processeur bascule en espace noyau et donne la main au noyau
- Ce code du noyau s'exécute dans le contexte de la tâche qui vient d'être interrompue (l'espace d'adressage virtuel est celui de la tâche, `current` pointe vers la structure représentant cette tâche, etc.)
- À la fin du traitement, la tâche initiale reprend son exécution, sauf si elle a été mise en sommeil, arrêtée, ou qu'une tâche plus prioritaire est ordonnancée à sa place

Plan

Création d'une liste d'attente

- Pour mettre en attente un processus, il faut disposer d'une file d'attente `wait_queue_head_t` (définie dans `include/linux/wait.h`)
- Cette structure peut être initialisée
  - Statiquement
    ```
    wait_queue_head_t queue;
    DECLARE_WAIT_QUEUE_HEAD(queue);
    ```
  - Dynamiquement
    ```
    wait_queue_head_t *queue;
    // Allocation mémoire...
    init_waitqueue_head(queue);
    ```

Mise en sommeil

```
void wait_event(wait_queue_head_t wait_queue, bool condition)
int wait_event_interruptible(wait_queue_head_t wait_queue, bool condition)
```

- Ces deux fonctions permettent de mettre la tâche courante en sommeil sur la liste `wait_queue`
- Si `condition` est évaluée à vrai à l'entrée de ces deux fonctions, elles retournent tout de suite (la tâche n'est pas mise en sommeil)
- Dans le cas contraire, la tâche est mise en sommeil
  - Dans l'état `TASK_UNINTERRUPTIBLE` pour la première
  - Dans l'état `TASK_INTERRUPTIBLE` pour la seconde

Mise en sommeil (suite)

```
void wait_event(wait_queue_head_t wait_queue,
                bool condition)
int wait_event_interruptible(wait_queue_head_t wait_queue,
                             bool condition)
```

- Une fois mis en sommeil, ces fonctions retourneront que si
  - La file d'attente est explicitement réveillée (voir plus loin dans le cours) *et*, qu'au moment où la tâche redevient active, `condition` est évaluée à vrai
    - Dans le cas contraire, la tâche est remise immédiatement en sommeil (et la fonction ne retourne pas tout de suite)
  - Ou, dans le cas de la deuxième fonction, un signal est reçu par la tâche
    - Dans ce cas, la fonction retourne tout de suite (même si `condition` est faux ou si la liste n'a pas été réveillée) et renvoie `-ERESTARTSYS`

Mise en sommeil (suite)

```
void wait_event(wait_queue_head_t wait_queue,
                bool condition)
int wait_event_interruptible(wait_queue_head_t wait_queue,
                             bool condition)
```

- L'usage de la variante `interruptible` est fortement recommandé
- La plupart du temps, si vous êtes dans une fonction de rappel suite à un appel système (par exemple les fonctions `foo_read`, `foo_write`...), et que `wait_event_interruptible` renvoie `-ERESTARTSYS`, il suffit de renvoyer `-ERESTARTSYS` par votre fonction
  - Cela se traduira soit par un rappel automatique de votre fonction, soit par un échec de l'appel système avec l'erreur `-EINTR` vu de l'application

Mise en sommeil (variantes)

```
int wait_event_timeout(wait_queue_head_t wait_queue,
                       bool condition, timeout)
int wait_event_interruptible_timeout(wait_queue_head_t wq,
                                     bool condition, timeout)
```

- Ces deux variantes ajoutent une condition de sortie anticipée en cas d'expiration du `timeout` exprimé en *jiffies*
- Elles renvoient
  - `0` : *timeout* expiré et `condition` fausse
  - `1` : `condition` vraie à l'issue du *timeout* ou du réveil de la liste d'attente
  - `-ERESTARTSYS` : signal reçu (pour `wait_event_interruptible_timeout`)

Plan

Réveil

```
void wake_up(wait_queue_head_t *q);
```

- Toutes les tâches en attente sur cette listes sont réveillées
- Lorsque l'ordonnanceur leur donnera du temps d'exécution, elles évalueront chacune leur condition d'attente (`condition` passée aux fonctions `wait_event...`)
  - Pour les tâches dont la condition est vérifiée, l'attente est terminée, l'appel à la fonction `wait_event...` se termine
  - Pour les autres, elles sont rendormies
- L'appel à la fonction de réveil est typiquement fait dans un gestionnaire d'interruption pour réveiller la ou les tâches qui étaient en attente de cet événement

Réveil

```
void wake_up_interruptible(wait_queue_head_t *q);
```

- Cette fonction ne réveille que les tâches qui ont été mises en attente via `wait_event_interruptible*`
- Par convention, si vous utilisez `wait_event_interruptible*`, utilisez `wake_up_interruptible` (vous pouvez aussi utiliser `wake_up`)

Plan

Attente exclusive

- Dans certaines situations, plusieurs tâches peuvent être en attente d'un événement mais une seule de ces tâches sera capable de traiter cet événement
- Il est donc superflu de les réveiller toutes

Mise en attente

- Les fonctions vues précédemment mettent la tâche en attente *non exclusive*
- `int wait_event_interruptible_exclusive(wait_queue_head wait_queue, bool condition)` place la tâche en attente *exclusive*

Réveil

- `wake_up(wait_queue_head_t *q)` et `wake_up_interruptible(wait_queue_head_t *q)` réveillent toutes les tâches en attente *non exclusive* et une seule tâche en attente *exclusive* (s'il y en a)
- `wake_up_nr(wait_queue_head_t *q, int nr)` et `wake_up_interruptible_nr(wait_queue_head_t *q, int nr)` réveillent toutes les tâches en attente *non exclusive* et au plus `nr` tâches en attente *exclusive*
- `wake_up_all(wait_queue_head_t *q)` et `wake_up_interruptible_all(wait_queue_head_t *q)` réveillent toutes les tâches en attente *non exclusive* et toutes les tâches en attente *exclusive*

Plan

Exemple d'utilisation dans un pilote

- Une liste d'attente est déclarée dans la structure `foo_device` pour stocker les processus en attente de données en provenance du périphérique
- Dans la fonction de rappel `foo_read`, le pilote regarde si des données sont actuellement disponibles. Si ce n'est pas le cas, le processus est mis en attente
- Dans le gestionnaire d'interruption appelé lorsque des données sont disponibles dans le périphérique, le pilote réveille les processus en attente