NOM

sched_setscheduler, sched_getscheduler - Lire / fixer la politique d'ordonnancement et ses paramètres.

SYNOPSIS

```
#include <sched.h>

int sched_setscheduler(pid_t pid, int policy, const struct  sched_param
*p);

int sched_getscheduler(pid_t pid);

struct sched_param {
    ...
    int sched_priority;
    ...
};
```

DESCRIPTION

sched_setscheduler fixe à la fois la politique d'ordonnancement et ses paramètres pour le processus identifié par pid. Si pid vaut zéro, la politique du processus en cours sera fixée. L'interprétation du paramètre p dépend de la politique employée. Actuellement il y a trois politiques proposées par Linux : SCHED_FIFO, SCHED_RR, et SCHED_OTHER. Leurs sémantiques respectives sont décrites ci-dessous.

sched_getscheduler lit la politique d'ordonnancement et ses paramètres pour le processus identifié par pid. Si pid vaut zéro, la politique du processus en cours sera récupérée.

Politiques d'ordonnancement

L'ordonnanceur est la partie du noyau qui décide quel processus prêt va être exécuté ensuite. L'ordonnanceur de Linux propose trois politiques différentes, une pour les processus classiques, et deux pour les applications à vocation temps-réel.

Une valeur de priorité statique sched_priority est assignée à chaque processus, et ne peut être modifiée que par l'intermédiaire d'appels systèmes. Conceptuellement, l'ordonnanceur dispose d'une liste de tous les processus prêts pour chaque valeur possible de sched_priority (sched_priority est dans l'intervalle 0 à 99).

Afin de déterminer quel processus doit s'exécuter ensuite, l'ordonnanceur de Linux recherche la liste non-vide de plus haute priorité statique et prend le processus en tête de cette liste. La politique d'ordonnancement détermine pour chaque processus l'emplacement où il sera inséré dans la liste contenant les processus de même priorité statique, et comment il se déplacera dans cette liste.

SCHED_OTHER est l'ordonnancement universel temps-partagé par défaut, utilisé par la plupart des processus. SCHED_FIFO et SCHED_RR sont prévus pour des applications temps-réel qui nécessitent un contrôle précis de la sélection des processus prêts.

Les processus ordonnancés avec SCHED_OTHER doivent avoir une priorité statique de 0, ceux ordonnancés par SCHED_FIFO ou SCHED_RR peuvent avoir une priorité statique dans l'intervalle 1 à 99. Seuls les processus disposant de privilèges Super-User peuvent obtenir une priorité statique supérieure a 0 afin d'être ordonnancé par SCHED_FIFO ou SCHED_RR.

Les appels systèmes sched_get_priority_min et sched_get_priority_max permettent de déterminer l'intervalle de priorités valides de manière portable sur les systèmes conformes à la norme POSIX.1b.

Tout ordonnancement est préemptif : Si un processus avec une priorité statique plus élevée devient prêt, le processus en cours est interrompu et retourne dans sa liste d'attente. La politique d'ordonnancement détermine simplement l'ordre utilisé dans une liste de processus prêts avec des priorités statiques égales.

SCHED_FIFO: Ordonnancement First In-First out (premier arrivé, premier servi)
SCHED_FIFO ne peut être utilisé qu'avec des priorités statiques supérieures à 0, ce qui signifie que dès qu'un processus SCHED_FIFO devient prêt, un processus normal SCHED_OTHER en cours d'exécution sera interrompu. SCHED_FIFO est un ordonnancement simple à base de tranches de temps. Pour les processus ordonnancés par SCHED_FIFO les règles suivantes sont appliquées :

Un processus SCHED_FIFO qui a été préempté par un autre processus de priorité supérieure restera en tête de sa liste et reprendra son exécution dès que tous les processus de priorités supérieures sont à nouveau bloqués.

Quand un processus SCHED_FIFO devient prêt, il est inséré à la fin de sa liste.

Un appel système sched_setscheduler ou sched_setparam placera le processus SCHED_FIFO identifié par pid à la fin de sa liste s'il est prêt.

Un processus appelant sched_yield sera placé à la fin de sa liste.

Aucun autre évènement ne modifiera l'ordre des listes de priorités statiques égales avec SCHED_FIFO.

Un processus SCHED_FIFO s'exécute jusqu'à ce qu'il soit bloqué par une opération d'entrée/sortie, qu'il soit préempté par un processus de priorité supérieure, ou qu'il appelle sched_yield.

SCHED_RR: Ordonnancement Round Robin
SCHED_RR est une amélioration simple de la politique SCHED_FIFO. Tout ce qui est décrit pour SCHED_FIFO s'applique aussi à SCHED_RR, sauf que chaque processus ne dispose que d'une tranche temporelle limitée pour son exécution. Si un processus sous politique SCHED_RR s'est exécuté depuis une durée supérieure ou égale à la tranche temporelle (time quantum), il sera placé à la fin de la liste de sa priorité.

Un processus sous SCHED_RR qui a été préempté par un processus de priorité supérieure terminera sa tranche de temps lorsqu'il reprendra son exécution. la longueur du time quantum peut être lue avec sched_rr_get_interval.

SCHED_OTHER: Ordonnancement temps-partagé par défaut
La politique SCHED_OTHER ne peut être utilisée qu'avec des priorités statiques à 0. C'est la politique standard de l'ordonnanceur temps partagé de Linux, et est conçue pour tous les processus ne réclamant pas de fonctionnalités temps-réel.

Le processus à exécuter est choisi dans la liste des processus de priorités statiques nulles, en utilisant une priorité dynamique qui ne s'applique que dans cette liste.

La priorité dynamique est basée sur la valeur de "gentillesse" du processus (fixée avec les appels systèmes nice ou setpriority) et est incrémentée à chaque time quantum où le processus est prêt mais non sélectionné par l'ordonnanceur. Ceci garantit une progression équitable de tous les processus SCHED_OTHER.

Temps de réponse
Un processus de haute priorité bloqué en attente d'entrées/sorties est affecté d'un certain temps de réponse avant d'être sélectionné à nouveau. Le concepteur d'un gestionnaire de périphérique peut réduire grandement ce temps de réponse en utilisant un gestionnaire d'interruptions lentes comme décrit dans request_irq(9).

Divers
Les processus fils héritent de la politique d'ordonnancement et des paramètres associés lors d'un fork.

Le verrouillage de pages en mémoire est généralement nécessaire pour les processus temps réel afin d'éviter les délais de pagination. Ceci peut être effectué avec mlock(2) ou mlockall(2).

Comme une boucle sans fin non bloquante dans un processus ordonnancé sous une politique SCHED_FIFO ou SCHED_RR bloquera indéfiniment tous les processus avec une priorité plus faible, le développeur d'applications temps-réel devrait toujours conserver sur une console un shell ordonnancé avec une priorité supérieure à celle de l'application testée.

Ceci permettra un kill((1) d'urgence des applications testées qui ne se bloquent pas ou qui ne se terminent pas comme prévu. Comme les processus sous SCHED_FIFO et SCHED_RR peuvent préempter les autres processus normaux pour toujours, seuls les processus Super-User ont le droit d'activer ces politiques sous Linux.

Les systèmes POSIX sur lesquels sched_setscheduler et sched_getscheduler sont disponibles définissent _POSIX_PRIORITY_SCHEDULING dans <unistd.h>.

VALEUR RENVOYÉE

sched_setscheduler renvoie 0 s'il réussit sched_getscheduler renvoie la politique pour le processus s'il réussit.

En cas d'échec, -1 est renvoyé et errno contient le code d'erreur.

ERREURS

ESRCH Le processus numéro pid n'existe pas.

EPERM Le processus appelant n'a pas les privilèges nécessaires. Seul les processus Super-User peuvent activer les politiques SCHED_FIFO et SCHED_RR. Le processus appelant sched_setscheduler doit avoir un UID effectif égal à celui du processus pid, ou être Super-User.

EINVAL La valeur de politique d'ordonnancement policy n'existe pas, ou le paramètre p n'a pas de signification pour la politique policy.

CONFORMITÉ

POSIX.1b (POSIX.4)

BOGUES

Sous Linux 1.3.81,SCHED_RR n'a pas été testé totalement, et ne se comporte peut être pas exactement comme décrit ci-dessus.

VOIR AUSSI

sched_setparam(2), sched_getparam(2), sched_yield(2), sched_get_priority_max(2), sched_get_priority_min(2), nice(2), setpriority(2), getpriority(2), mlockall(2), munlockall(2), mlock(2), munlock(2).

Programming for the real world - POSIX.4 by Bill O. Gallmeister, O'Reilly & Associates, Inc., ISBN 1-56592-074-0
IEEE Std 1003.1b-1993 (POSIX.1b standard)
ISO/IEC 9945-1:1996 - C'est une nouvelle révision 1996 de POSIX.1 qui regroupe en un seul standard les normes POSIX.1(1990), POSIX.1b(1993), POSIX.1c(1995), et POSIX.1i(1995).

TRADUCTION

Christophe Blaess, 1997.