INSTITUT NATIONAL
DES SCIENCES
APPLIQUÉES
CENTRE VAL DE LOIRE

Système II : L'ordonnancement

INSA CVL 3ème Année SZPIEG

Département STI

Promotion 2020

Année 2017-2018

Ordonnancement Scheduling Noyau 2.6 et supérieur

- Objectifs :
  - ✓ Temps de réponse rapide
  - ✓ Bon débit pour les tâches d'arrière plan
  - ✓ Eviter la famine (processus sans temps d'exécution)
  - ✓ Gérer au mieux les besoins processus basse et haute priorité

Définition

- Politique d'ordonnancement

L'ensemble des règles qui détermine quand et comment un processus s'exécute dans le temps s'appelle « Politique d'ordonnancement »

Multiplexage temporel

- L'ordonnancement est basé sur la technique du temps partagé:
  - Le temps processeur est partagé en tranches, les quanta, et permettent à plusieurs processus de fonctionner en pseudo parallélisme.
  - Sur un système multi processeurs ou multi cœurs, le parallélisme est bien réel.
  - Le principe de temps partagé repose sur l'interruption d'horloge, il est transparent pour le processus lui même. Donc pas de code à ajouter dans le processus

Durée des quanta de temps

- Les quanta de temps ne doivent ni être trop long ce qui se ferait au détriment du pseudo-parallélisme.
  Ni trop court, car le microprocesseur passerait son temps en changement de contexte.
- Environ 800 à 5 ms

Algorithme d'ordonnancement

- Dans le noyau 2.6 chaque processeur possède sa propre file d'ordonnancement.
- Le choix du processus à exécuter se fait à temps constant quelque soit le nombre de processus.
- L'ordonnanceur à toujours un processus à exécuter c'est le processus « swapper » PID 0 qui s'exécute si aucun autre processus disponible.

Processus actifs et processus expirés

- Le noyau conserve deux listes de fichiers exécutables.
  Les processus actifs :
  qui n'ont pas totalement consommés leurs quanta de temps.
  Les processus expirés :
  qui ont consommé ces quanta.

- Le noyau privilégie les premiers par rapport aux seconds mais n'affame pas les seconds.

Les politiques d'ordonnancement

- SCHED_FIFO :
  Processus temps réel First In First Out.
  Si un processus Fifo tient le temps processeur alors le noyau le laisse en début de file si aucun processus temps réel de priorité supérieure n'apparaît.
- SCHED_RR Round Robin (Tourniquet)
  Si un processus RR tient le temps processeur alors, a la fin de son quantum, le noyau le mets à la fin de la liste des processus temps réel de même niveau de priorité.
- SCHED_OTHER ou SCHED_NORMAL
  Processus traditionnel à temps partagé. Politique par défaut pour un processus

Notion de priorité et SCHED_OTHER

- Chaque processus possède une valeur qui permet à l'ordonnanceur de faire le choix du processus à activer.
- Cette valeur est dynamique sous Linux, elle est appelée « priorité dynamique».
- Les processus qui n'ont pas depuis quelques temps de quanta de temps attribués voient cette valeur augmentée. Les autres voient cette valeur chutée.

Notion de priorité et SCHED_OTHER

- L'algorithme utilise aussi une valeur appelée « Priorité statique ».
  Cette valeur est dans l'intervalle [100,139].
  100 forte priorité, 139 faible priorité, on pourrait parler de poids.
  Un nouveau processus hérite de la priorité statique du parent.
  On peut manipuler en partie cette valeur grâce aux appels système nice() et setpriority() et en mettant des valeurs en paramètres dans l'intervalle [-20,+19]

Priorité statique et quantum de base

- Quantum de base (ms)= (140 – priorité statique)*k

  k = 20 si priorité statique <120
  k = 5 si priorité statique >=120

- Donc plus la priorité statique est basse (valeur grande) et plus le quantum de temps est court

Classification des processus

- On distingue deux types de processus :
  - ✓ Les consommateurs d'entrées/sorties.
    Ils passent beaucoup de temps à attendre des données mais ils se doivent de réagir vite lorsqu'elles arrivent (Shells, éditeurs et traitements de texte)
  - ✓ Les consommateurs de temps processeur.
    Ils font des calculs intensifs, on parle de traitement par lots.

Classification des processus

- Pour que le système semble réactif à l'utilisateur, il faut que le temps de délai moyen soit entre 50 et 150 ms.
- D'où l'ordonnanceur pénalisera plutôt les applications de calcul (Compilateur, calcul scientifique) au profit des applications shell et traitement de texte.
- Pour tenir compte de ces deux catégories le noyau affecte au processus une valeur appelée bonus

Priorité statique et dynamique

- Donc un processus possède une priorité statique et une priorité dynamique, valeur variant de [100, 139]
- La priorité dynamique est liée à la priorité statique par la formule suivante :
  Priorité dynamique = MAX (100, min (priorité statique – bonus+5,139))
- Le bonus est une valeur comprise entre [0,10]
- Le bonus dépend de l'historique du processus et de sa durée moyenne de veille.
- La durée moyenne de veille est le nombre de millisecondes où le processus était en attente de quantum lors de l'exécution des autres processus.
- Cette durée moyenne ne peut excéder 1 seconde

Durée moyenne de veille

- Comment classer un processus dans la catégorie « interactif » ou « traitement par lots »
  - Si bonus-5 >= priorité statique/4-28 alors le processus est interactif.
- Le deuxième terme de l'inégalité est appelé « Delta d'activité ».
- Rq : Il est plus facile pour un processus de haute priorité (priorité statique faible) de devenir « interactif »

Exemples de valeurs

Description	PS	Valeur Nice	Quantum de base	Delta Interactif	Limite sommeil
Highest Static Priority	100	-20	800ms	-3	299ms
High Static Priority	110	-10	600ms	-1	499ms
Default Static Priority	120	0	100ms	+2	799ms
Low Static Priority	130	10	50ms	+4	999ms
Lowest Static Priority	139	19	5ms	+6	1199ms

Sched_other

Préemption de processus

- Les processus LINUX sont préemptibles.
- Lorsque qu'un processus passe à l'état « TASK_RUNNING », le noyau regarde sa priorité dynamique, si elle est supérieure au processus courant alors le premier est interrompu.
  Le processus courant est aussi interrompu lorsque son quantum de temps est expiré.
  Un processus préempté reste à l'état « TASK_RUNNING »

Les processus temps-réel

- Les processus temps réel ne doivent pas être bloqués par des processus de moindre priorité (SCHED_OTHER).
- On peut considérer que les processus Sched_other ont une priorité égale à 0 alors que FIFO et RR ont une priorité >0.
- Contrôle de système embarqués « robot »
  Collecteur de données physiques « capteur »
  Flux vidéo ...

Ordonnanceur et classification

- La catégorie temps réel est explicite à la différence des catégories « consommateur I/O » et « consommateur temps processeur ».
- Rappel deux politiques temps réel sous Linux
  - SCHED_FIFO :
    Processus temps réel First In First Out.
    Si un processus Fifo tient le temps processeur alors le noyau le laisse en début de file si aucun processus temps réel de priorité supérieure n'apparaît.
  - SCHED_RR Round Robin (Tourniquet)
    Si un processus RR tient le temps processeur alors, a la fin de son quantum, le noyau le mets à la fin de la liste des processus temps réel de même niveau de priorité.

Priorité et processus TR

- Chaque processus temps réel a une priorité dans l'intervalle [1, 99].
  99 priorité la plus basse.
- Un processus temps réel inhibe les autres processus TR de plus basse priorité.

Exemple de résultats

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1112	root	20	0	637m	27m	9844	S	1	1.4	0:07.09	Xorg
2446	martial	20	0	63084	8756	3068	S	1	0.4	0:00.79	beam.smp
2587	root	20	0	2472	1204	884	R	1	0.1	0:00.11	top
2486	martial	20	0	39180	13m	9732	S	0	0.7	0:01.52	gnome-terminal
2510	martial	20	0	30096	16m	3948	S	0	0.8	0:02.05	ubuntuone-syncd
1	root	20	0	2656	1540	1128	S	0	0.1	0:01.02	init
2	root	15	-5	0	0	0	S	0	0.0	0:00.00	kthreadd
3	root	RT	-5	0	0	0	S	0	0.0	0:00.00	migration/0
4	root	15	-5	0	0	0	S	0	0.0	0:00.04	ksoftirqd/0
5	root	RT	-5	0	0	0	S	0	0.0	0:00.00	watchdog/0
6	root	RT	-5	0	0	0	S	0	0.0	0:00.00	migration/1
7	root	15	-5	0	0	0	S	0	0.0	0:00.00	ksoftirqd/1

Les appels systèmes

nice() : change la priorité statique d'un processus
getpriority() : obtient la propriété statique setpriority() :
sched_getscheduler() : obtient la politique de scheduling
sched_setscheduler()
sched_getparam() : obtient la priorité temps réel
sched_setparam()
sched_yield() : Rend le processeur volontairement
sched_get_priority_min() : Obtient la priorité temps réel min
sched_get_priority_max()
sched_rr_get_interval() : valeur du quantum.

Nouvelle priorité SCHED_BATCH

Depuis « Linux 2.6.16. » planification des traitements par lots.

SCHED_BATCH ne peut être utilisé qu'à la priorité statique 0. Cette politique est similaire à SCHED_OTHER car elle planifie le thread par sa priorité dynamique (nice()).
La différence est que cette stratégie obligera le planificateur à toujours supposer que le thread est gourmand en ressources processeur. Par conséquent, l'ordonnanceur appliquera une petite pénalité d'ordonnancement en ce qui concerne le comportement de réveil, de sorte que ce thread est légèrement défavorisé dans les décisions d'ordonnancement.

Nouvelle priorité SCHED_IDLE

Depuis « Linux 2.6.23. » planification de tâches à très faible priorité
SCHED_IDLE ne peut être utilisé qu'à la priorité statique 0. La valeur nice() du processus n'a aucune influence sur cette politique.

Cette stratégie est destinée à exécuter des travaux à une priorité extrêmement basse (inférieure même à une valeur nice() de +19 avec les stratégies SCHED_OTHER ou SCHED_BATCH).

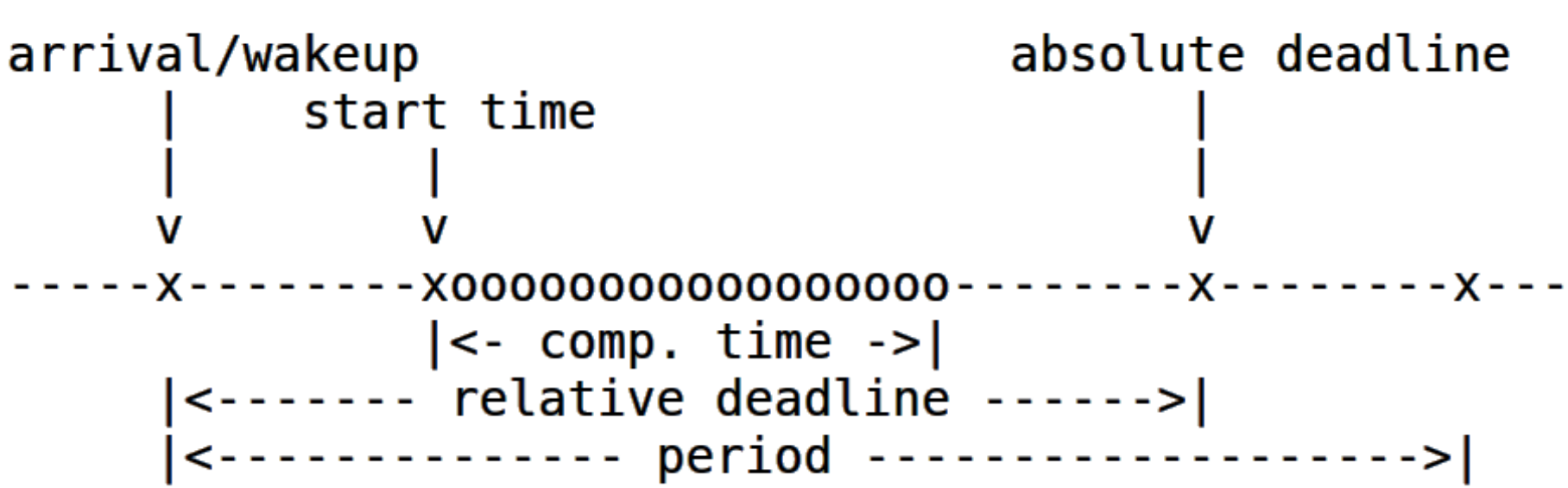
Nouvelle priorité SCHED_DEADLINE 1/3

Depuis « Linux 3.14 » planification des délais de tâche sporadique
Cette politique est actuellement implémentée en utilisant l'algorithme GEDF (Global Earliest Deadline First) conjointement avec CBS (Constant Bandwidth Server).
Pour définir et extraire cette stratégie et les attributs associés, vous devez utiliser les appels système sched_setattr (2) et sched_getattr (2) spécifiques à Linux.
Une tâche sporadique est une tâche qui comporte une séquence de tâches, chaque tâche étant activée au maximum une fois par période.
Chaque travail a également un délai relatif, devant lequel il devrait terminer l'exécution, et un temps de calcul, qui est le temps CPU nécessaire pour exécuter le travail.

Nouvelle priorité SCHED_DEADLINE 2/3

Le moment où une tâche se réveille parce qu'un nouveau travail doit être exécuté est appelé l'heure d'arrivée (également appelée heure de la demande ou heure de libération). L'heure de début est l'heure à laquelle une tâche commence son exécution. Le délai absolu est ainsi obtenu en ajoutant l'échéance relative à l'heure d'arrivée.
Algorithme GEDF : Global Earliest deadline first scheduling ("échéance proche = préparation en premier") est un algorithme d'ordonnancement préemptif, à priorité dynamique.
Il attribue une priorité à chaque requête en fonction de l'échéance de cette dernière selon la règle: Plus l'échéance d'une tâche est proche, plus sa priorité est grande.

Nouvelle priorité SCHED_DEADLINE 3/3



La commande « chrt »

La commande « chrt » permet de gérer les attributs temps réel d'un processus.

```
martial@acerfix ~ $ chrt -m
SCHED_OTHER priorité min/max      : 0/0
SCHED_FIFO priorité min/max       : 1/99
SCHED_RR priorité min/max         : 1/99
SCHED_BATCH priorité min/max      : 0/0
SCHED_IDLE priorité min/max       : 0/0
```