# TD 1

# Ordonnancement

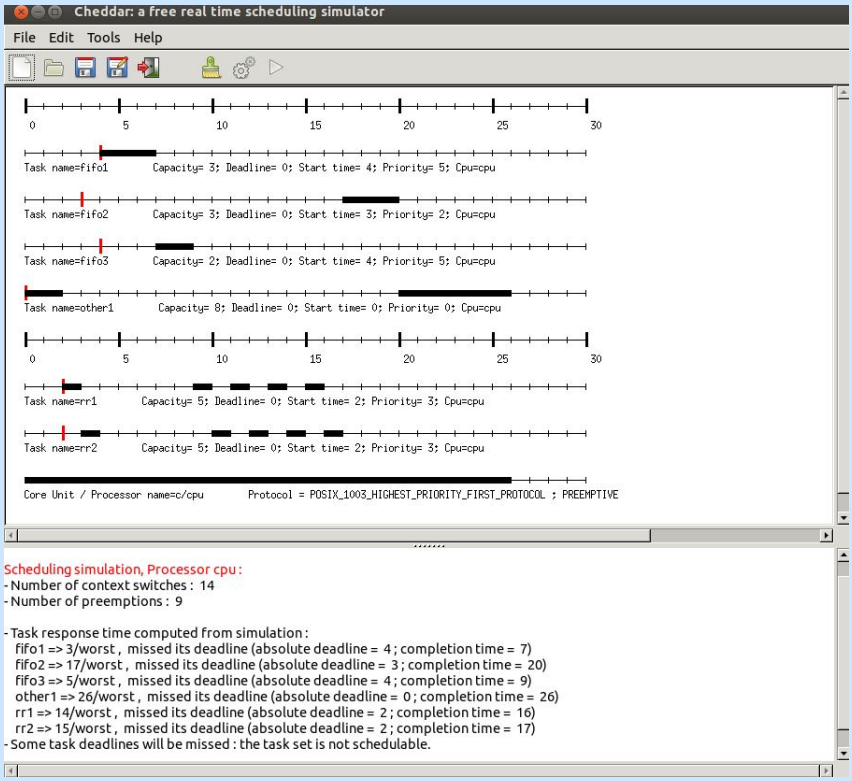
## 1.1 Ordonnancement avec Posix 1003.1b

EXERCICE 1 ▸ **Gestion des files d'attente**

Dans cet exercice, nous utilisons l'implantation des spécifications POSIX 1003.1b sur Linux. Sur Linux, il existe 100 niveaux de priorité : le niveau de priorité 0 est réservé à SCHED_OTHER et les niveaux de priorité 1 à 99 aux politiques SCHED_FIFO et SCHED_RR. Les tâches de priorité 99 sont les tâches de plus forte priorité. SCHED_OTHER est dédiée à l'ordonnanceur temps partagé. Le quantum utilisé par la politique SCHED_RR est d'une unité de temps. Pour simplifier, nous supposons que la politique SCHED_OTHER alloue le processeur de façon inversement proportionnel au temps processeur consommé par les tâches. L'ordonnancement est préemptif.

| Nom | Capacité | Date d'arrivée | Priorité | Politique |
|---|---|---|---|---|
| other1 | 8 | 0 | 0 | SCHED_OTHER |
| rr1 | 5 | 2 | 3 | SCHED_RR |
| rr2 | 5 | 2 | 3 | SCHED_RR |
| fifo1 | 3 | 4 | 5 | SCHED_FIFO |
| fifo2 | 3 | 3 | 2 | SCHED_FIFO |
| fifo3 | 2 | 4 | 5 | SCHED_FIFO |

Soit le jeu de tâches apériodiques ci-dessus. On suppose qu'une fois arrivée, les tâches sont toujours prêtes. Dessinez de l'instant 0 à l'instant 25, l'ordonnancement généré par l'ordonnanceur POSIX 1003.1b.




## 1.2 Ordonnancement périodique temps-réel

*Jolis dessins par C. Bompard, avril 2020.*

### 1.2.1 Résumé

On rappelle que pour une tâche $T_i$ :

— $S_i$ correspond au moment où la tâche est disponible

— $P_i$ correspond à la période de la tâche (la durée au bout de laquelle la tâche recommence)

— $C_i$ correspond à la capacité de la tâche, c'est sa durée d'exécution

**Ordonnancement à priorité fixe avec affection de priorité RM** Le calcul de priorité consiste à associer à chaque tâche une priorité fixe inversement proportionnelle à sa périodicité (Rate Monotonic).

La phase d'élection consiste à élire la tâche de plus forte priorité (ordonnancement à priorité fixe).

Tests de l'ordonnançabilité (**Hypothèses : tâches indépendantes et périodiques. Algorithmes préemptifs.**) :

1. Test sur le taux d'utilisation avec $\forall i : D_i = P_i : \sum_{i=1}^{n} \frac{C_i}{P_i} \leq n(2^{(1/n)} - 1)$ condition suffisante mais non nécessaire.
2. Utilisation de la période d'étude : ordonnancement à comportement cyclique (propriété du modèle périodique). Période d'étude = $[0, PPCM(P_i)]$ (si $\forall i : S_i = 0$).

**L'algorithme EDF** Le calcul de priorité consiste à déterminer une échéance. L'échéance $D_i(t)$ d'une tâche $i$ à l'instant $t$ est égale à la somme de la date de début de l'activation courante à l'instant $t$ et du délai critique $D_i$.

La phase d'élection consiste à élire la tâche de plus *proche* échéance.
Tests d'ordonnançabilité (**Hypothèses : tâches indépendantes et périodiques. Algorithmes préemptifs.**) :

1. Test sur le taux d'utilisation :
   — Cas $\forall i : D_i = P_i : \sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$ condition nécessaire et suffisante.
   — Cas $\exists i : D_i < P_i : \sum_{i=1}^{n} \frac{C_i}{D_i} \leq 1$ condition suffisante, et $\sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$ condition nécessaire uniquement.
2. Utilisation de la période d'étude (propriété du modèle périodique).

### 1.2.2 Exercices

*Les exercices de cette section sont tirés d'un TD de F. Singhoff, univ Brest, avec l'aimable autorisation de l'auteur. Une bonne partie des corrections aussi.*

EXERCICE 2 ▶ **Ordonnancement à priorité fixe + Rate Monotonic (RM)**

Soient trois tâches périodiques T1, T2 et T3 définies par les paramètres suivants : $S_1 = S_2 = S_3 = 0, P_1 = 29, C_1 = 7, P_2 = 5, C_2 = 1, P_3 = 10, C_3 = 2$. On suppose un ordonnancement à priorité fixe avec une affection Rate Monotonic des priorités. Les délais critiques sont égaux aux périodes (soient $\forall\ i\ :\ D_i = P_i$).

1. Calculez le taux d'utilisation pour RM. Le jeu de tâches est il ordonnançable ?

   Le jeu de tâches est ordonnançable car le taux d'utilisation est inférieur à la borne de Liu et Layland : $n \times (2^{1/n} - 1)$. Ici, $n = 3$ et le taux d'utilisation est égal à $U = \frac{C_1}{P_1} + \frac{C_2}{P_2} + \frac{C_3}{P_3} = \frac{7}{29} + \frac{1}{5} + \frac{2}{10}$. Un petit tour par exemple ici `https://www.xcasenligne.fr/giac_online/demoGiacPhp.php` en tapant :

   ```
   evalf(7/29+1/5+2/10)
   0.641379310345
   ```

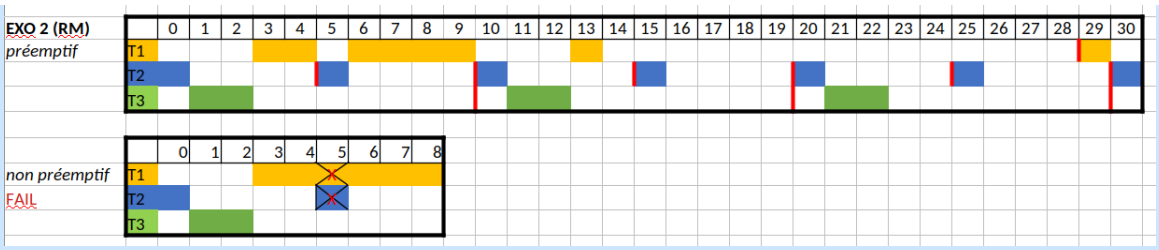
   La borne de Liu et Layland pour $n = 3$ est quant à elle égale à :

   ```
   evalf(3*(2^(1/3)-1)
   0.779763149685
   ```

   La condition suffisante est vérifiée, donc le jeu de tâche est ordonnançable.

2. Dessinez, sur les 30 premières unités de temps, l'ordonnancement généré par RM, d'abord avec la version préemptive, puis, avec la version non préemptive (vous commencerez à la date zéro). Que constatez vous ?

   Comme on est en RM, les priorités sont $P(T_2) > P(T_3) > P(T_1)$, soit inversement proportionnel à la périodicité des trois tâches. Les règles du jeu font qu'à chaque date multiple de la période $P_i$ la tâche $i$ est de nouveau tirable.




3. Calculez le temps de réaction de chaque tâche.

Le temps de réponse correspond à la différence entre la terminaison de la tâche et le moment où la tâche est disponible (0 pour les trois tâches ici). On obtient donc :

— pour la tâche 1 : 14

— pour la tâche 3 : 3

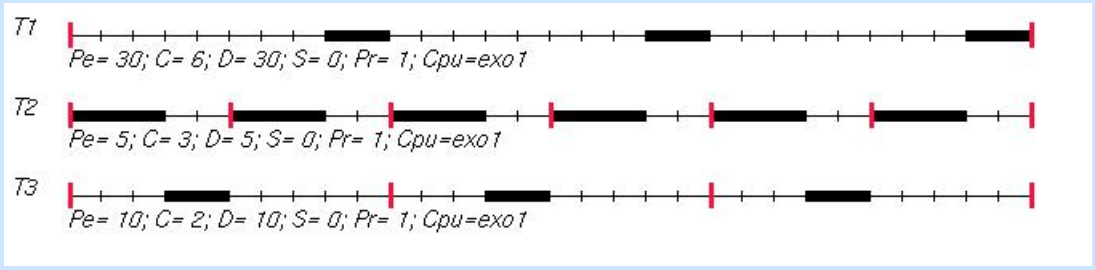
— pour la tâche 2 : 1

Le temps de réponse de chacune des tâches est bien inférieur à la périodicité, ouf.

4. Nous modifions la tâche T1 par $P_1 = 30$ et $C_1 = 6$ et la tâche T2 par $C_2 = 3 (P_2 = 5)$. On conserve la tâche T3 ($P_3 = 10$, $C_3 = 2$). Le jeu de tâches est dit "harmonique". En effet, chaque période du jeu de tâches est multiple des autres périodes. Refaites le test d'ordonnançabilité par le taux d'utilisation. Puis, dessinez de nouveau l'ordonnancement généré par RM sur sa période d'étude (mode préemptif). Que constatez vous ?

On commence par calculer le taux d'utilisation. $U = \frac{6}{30} + \frac{3}{5} + \frac{2}{10} = 1$. Le taux d'utilisation est égal à 1 mais les tâches sont harmoniques : le jeu de tâches est donc faisable dans ce cas

Attention : le test de Liu et Layland sur le taux d'utilisation avec RM est une condition suffisante mais non nécessaire.




## EXERCICE 3 ► EDF : Earliest deadline first

Soient trois tâches périodiques T1, T2 et T3 définies par les paramètres suivants : $S_1 = S_2 = S_3 = 0, P_1 = 12, C_1 = 5, P_2 = 6, C_2 = 2, P_3 = 24, C_3 = 5$. Les délais critiques sont égaux aux périodes (soient $\forall\ i\ :\ D_i = P_i$).

1. Calculez le taux d'utilisation du processeur. Concluez sur l'ordonnançabilité du jeu de tâches.

On calcule encore une fois le taux d'utilisation. $U = \frac{5}{12} + \frac{2}{6} + \frac{5}{24} \approx 0.96$.

Le taux d'utilisation est $\leq 1$, donc la CS pour l'ordonnançabilité par EDF est vérifiée, et donc le jeu de tests est ordonnançable.

2. Déterminer le nombre d'unités de temps libre sur la période d'étude (le ppcm des périodes, ie 24) (sans dessiner d'ordonnancement). On trouvera une formule liant ce nombre à la période d'étude et le taux d'utilisation.

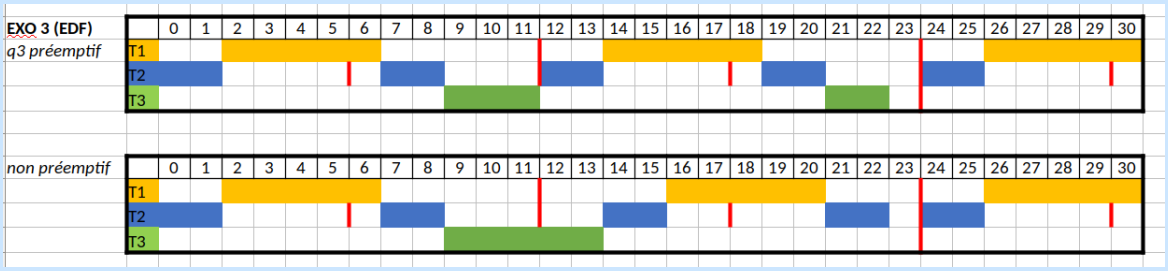
Cette partie n'a pas été vue en cours. La notion de taux d'utilisation correspond à la durée moyenne d'utilisation du processeur. On connaît donc le taux de "non-utilisation" via (1 - taux d'utilisation).

nb d'unités de temps libre = (1 - taux d'utilisation) × période d'étude

Ici, en reprenant la formule du taux d'utilisation précédente, on obtient un nombre d'unités de temps libre égal à 1 sur une période d'étude de 24.

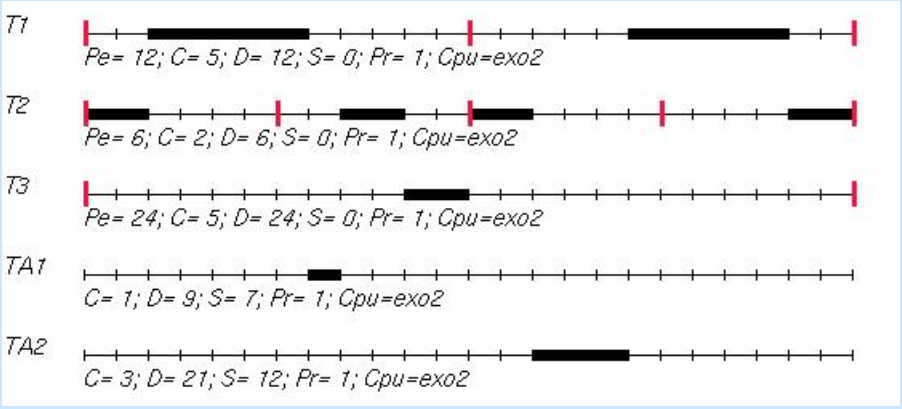
3. Confirmez les points précédents en dessinant, sur la période d'étude, l'ordonnancement généré par EDF, d'abord avec la version préemptive, puis, avec la version non préemptive.

Pas super difficile, en utilisant EDF (les priorités sont “la plus courte distance à échéance en premier”) :




4. On considère maintenant le même jeu de tâches mais cette fois ci, deux tâches apériodiques $TA_1$ et $T1_2$ arrivent respectivement aux instants 7 et 12. Leurs capacités sont de 1 et 3 unités de temps. Leurs échéances $D_i(t)$ interviennent aux instants 9 et 21. Le jeu de tâches est il ordonnançable par EDF (mode préemptif) ? Dessinez l'ordonnancement sur les 30 premières unités de temps.

Le cas de l'apériodique n'a pas été traité en cours. Sur le cas général on ne peut rien dire, mais là ça ne marche pas, car la tâche T3 ne respecte pas son échéance.




EDF est tres instable en surcharge.

## EXERCICE 4 ▶ RM avec tâches apériodiques

Soient deux tâches périodiques définies par les paramètres suivants : $S_1 = S_2 = 0, P_1 = 15, C_1 = 4, P_2 = 7, C_2 = 1$. Les délais critiques sont égaux aux périodes (soient $\forall\ i\ :\ D_i = P_i$). L'ordonnancement est effectué avec un algorithme à priorités fixes et RM (mode préemptif).

On souhaite faire cohabiter des tâches apériodiques avec les tâches définies ci-dessus. Pour ce faire, on utilise la méthode du serveur par scrutation. Cette méthode consiste à dédier à une tâche périodique (tâche dite "serveur par scrutation") l'exécution des tâches apériodiques. A chaque activation du serveur par scrutation, celui-ci regarde si des tâches apériodiques sont arrivées **avant le réveil du serveur** (les tâches apériodiques doivent évidemment être prêtes). Le cas échéant, le serveur attribue du temps processor à concurrence de la capacité du serveur : l'exécution d'une tâche apériodique peut être donc répartie sur plusieurs activations du serveur. Le temps de traitement nécessaire au serveur pour vérifier si des tâches apériodiques sont présentes est supposé comme nul : si le serveur est réveillé alors qu'aucune tâche apériodique n'est présente, alors, le serveur ne consomme pas de ressource processeur. La capacité du serveur est d'une unité de temps. Cette unité de temps est donc uniquement consacrée à l'exécution des tâches apériodiques.

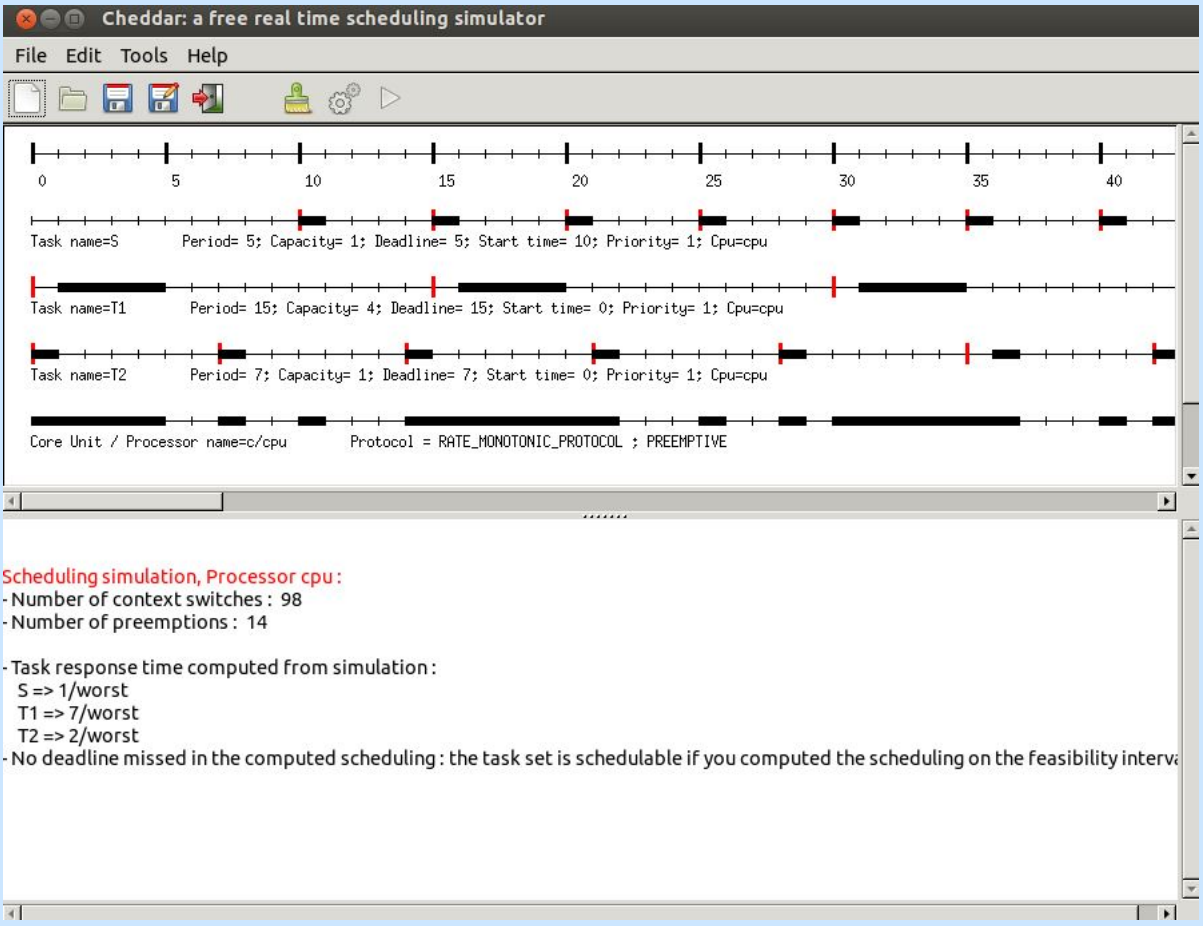
1. On suppose que le serveur par scrutation possède une période de 5 unités de temps. Le serveur arrive dans le système à l'instant zéro. Le jeu de tâches est il ordonnançable ?
2. On suppose maintenant que deux tâches apériodiques TA1 et TA2 arrivent respectivement aux instants 7 et 12. Leurs capacités sont de 1 et 3 unités de temps. Leurs échéances interviennent aux instants 9 et 21. Dessinez l'ordonnancement généré par le serveur par scrutation sur les 26 premières unités de temps.

1. Oui, le jeu de tâches est ordonnançable : il suffit de faire le test sur le taux d'utilisation en sommant sur les tâches T1, T2 et S ; S est la tâche périodique qui fait office de serveur de tâches apériodiques. (cf. formule page 15 du cours).

   Dans le cas où uniquement les tâches T1 et T2 existent (sans tâche apériodique supplémentaire), la tâche S ne

consomme rien car il n'y a pas de tâche apériodique à ordonnancer. On a alors un taux d'utilisation $U = \frac{4}{15} + \frac{1}{7} \leq 1$.

2. Notez bien qu'aux instants 0 et 5, la tâche S ne consomme pas d'unité de temps (car aucune tâche apériodique n'est présente). A l'instant 10, la tâche S exécute la tâche apériodique TA1. Aux instants 15, 20 et 24, la tâche S exécute la tâche apériodique TA2. Les contraintes de tâches TA1 et TA2 ne sont pas respectées : ce n'est pas grave puisque les tâches apériodiques ne sont a priori pas critiques.



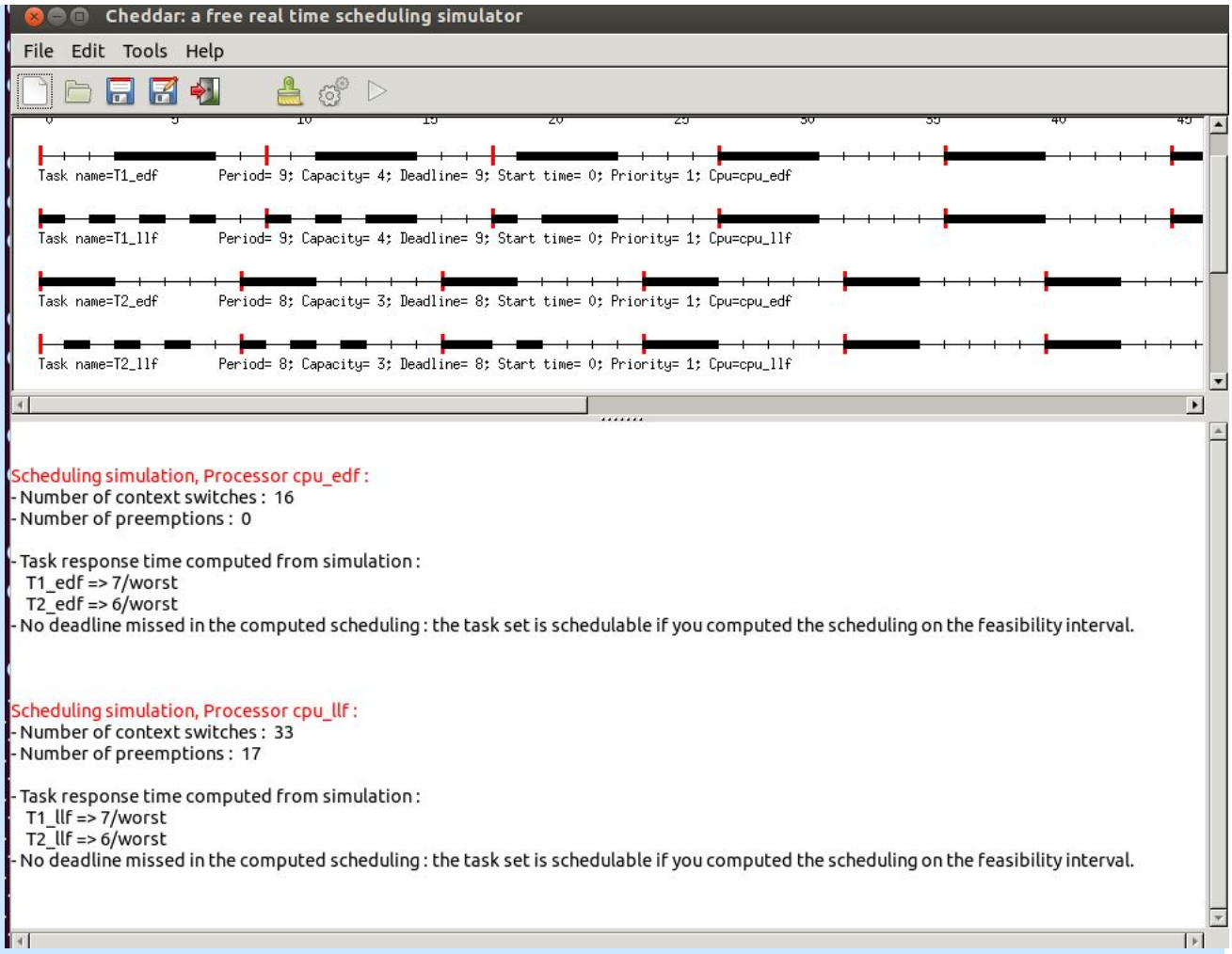

## EXERCICE 5 ▶ comparaison EDF/autre algo

Soient deux tâches T1 et T2 définies par les paramètres suivants : $S_1 = S_2 = 0, P_1 = 9, P_2 = 8, C_1 = 4, C_2 = 3$. Les délais critiques sont égaux aux périodes (soient $\forall\ i\ :\ D_i = P_i$).

On se propose d'étudier un nouvel algorithme d'ordonnancement dynamique : LLF (Least Laxity First). Ce dernier effectue l'élection des tâches prêtes grâce à leur laxité. La laxité, comme l'échéance, est une information dynamique qui évolue dans le temps. La laxité $L_i(t)$ d'une tâche $i$ à l'instant $t$ s'évalue par $L_i(t) = D_i(t) - reste(t)$ où $reste(t)$ est le reliquat de capacité à exécuter pour l'activation courante.

1. Dessinez l'ordonnancement généré par EDF sur les 20 premières unités de temps (en mode préemptif).
2. Dessinez l'ordonnancement généré par LLF sur les 20 premières unités de temps (en mode préemptif).
3. Que constatez vous? Conclure sur le choix entre ces deux algorithmes.

1. Avec EDF, toutes les échéances sont respectées avec une alternance des tâches T2 et T1.
2. Avec LLF, toutes les échéances sont respectées
3. On remarque que LLF introduit plus de commutations de contexte qu'EDF. L'efficacité d'LLF est la même que celle d'EDF. Toutefois, LLF utilise un critére de choix (la laxité) qui évolue à chaque fois qu'une tâche s'exécute durant une unité de temps ... d'où de plus nombreuses commutations de contexte. Cela peut être intéressant pour des questions de réactivité mais peu également induire un surcoût à cause du nombre de commutations de contextes, comme vu dans les cours précédents.


Cheddar: a free real time scheduling simulator
File Edit Tools Help
0 5 10 15 20 25 30 35 40 45
Task name=T1_edf Period= 9; Capacity= 4; Deadline= 9; Start time= 0; Priority= 1; Cpu=cpu_edf
Task name=T1_llf Period= 9; Capacity= 4; Deadline= 9; Start time= 0; Priority= 1; Cpu=cpu_llf
Task name=T2_edf Period= 8; Capacity= 3; Deadline= 8; Start time= 0; Priority= 1; Cpu=cpu_edf
Task name=T2_llf Period= 8; Capacity= 3; Deadline= 8; Start time= 0; Priority= 1; Cpu=cpu_llf
Scheduling simulation, Processor cpu_edf :
- Number of context switches : 16
- Number of preemptions : 0
- Task response time computed from simulation :
T1_edf => 7/worst
T2_edf => 6/worst
- No deadline missed in the computed scheduling : the task set is schedulable if you computed the scheduling on the feasibility interval.
Scheduling simulation, Processor cpu_llf :
- Number of context switches : 33
- Number of preemptions : 17
- Task response time computed from simulation :
T1_llf => 7/worst
T2_llf => 6/worst
- No deadline missed in the computed scheduling : the task set is schedulable if you computed the scheduling on the feasibility interval.