TD ordonnancement temps réel monoprocesseur

F. Singhoff, D. Massé
Université de Bretagne Occidentale
Lab-STICC UMR CNRS 6285

9 janvier 2018

Table des matières

1 Rappels de cours

1.1 Modèles de tâches

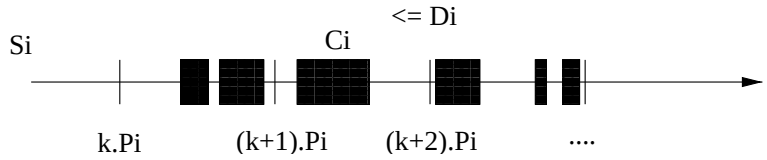

FIGURE 1 – Une tâche périodique

1. Périodiques. Une tâche périodique fait l'objet de plusieurs activations successives. Elle est définie par les paramètres S_i (date d'arrivée de la tâche dans le système), C_i (capacité de la tâche), P_i (période d'activation de la tâche), et D_i (délai critique exprimé relativement à P_i). Les tâches périodiques modélisent généralement des fonctions critiques du système.
2. Apériodiques. Une tâche apériodique i est activée une seule fois et est définie par les paramètres S_i (date d'arrivée de la tâche dans le système), C_i (capacité de la tâche), R_i (date d'exécution au plus tôt de la tâche) et D_i (délai critique). Les tâches apériodiques modélisent généralement des fonctions non critiques du système.
3. Sporadique : idem périodique mais P_i constitue un délai minimum inter-activations.

1.2 Ordonnancement à priorité fixe avec affection de priorité RM

Fonctionnement

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).

Test 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 $\forall i : S_i = 0$).
3. Test sur le temps de réponse de la tâche i (noté r_i) [JOS 86]) avec $D_i = P_i$; $r_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{r_i}{P_j} \right\rceil C_j$, à calculer par [AUD 93] : $w_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{P_j} \right\rceil C_j$. Conditions d'arrêt : échec si $w_i^n > P_i$, réussite si $w_i^{n+1} = w_i^n$. On commence avec $w_i^0 = C_i$.

1.3 L'algorithmes EDF

Fonctionnement

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 courte échéance.

Test de l'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 seulement, $\sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$ condition nécessaire uniquement.
2. Utilisation de la période d'étude (propriété du modèle périodique).

2 Exercices avec tâches indépendantes

2.1 Exercice 1 : ordonnancement à priorité fixe + Rate Monotonic

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 ?
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 ?
3. Nous modifions la tâche T1 par $P_1 = 30$ et $C_1 = 6$ et la tâche T2 par $C_2 = 3$. Le jeu de tâches est dit "harmonique". En effet, chaque période du jeu de tâches et multiple avec les autres périodes. Refaite 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 ?
4. Confirmez ce dernier résultat en calculant le temps de réponse de chaque tâche.

2.2 Exercice 2 : EDF

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.
2. Déterminer le nombre d'unité de temps libre sur la période d'étude.
3. Confirmez les points précédants en dessinant, sur la période d'étude, l'ordonnancement généré par EDF, d'abord avec la version préemptive, puis, avec la version non préemptive.
4. On considère maintenant le même jeu de tâches mais cette fois ci, 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 $D_i(t)$ 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.

2.3 Exercice 3 : serveurs de 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.

3 Exercices sur POSIX 1003.1b

3.1 Exercice 4 : politiques de 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.

3.2 Exercice 5 : POSIX 1003.1b et tâches périodiques

Soient deux tâches périodiques T1 et T2 définies par les paramètres suivants : C1=1, C2=2, P1=8, P2=7. Les tâches arrivent à l'instant 0. Les délais critiques sont égaux aux périodes

On considère un système POSIX 1003.1b offrant 32 niveaux de priorité (de 0 à 31). Les tâches de priorité 31 sont les tâches de plus forte priorité. Ce système propose les trois politiques POSIX 1003.1b *SCHED_FIFO*, *SCHED_RR* et *SCHED_OTHER*. Le quantum utilisé pour la politique *SCHED_RR* est d'une unité de temps. La politique *SCHED_OTHER* implante un algorithme d'ordonnancement pour les applications temps partagé. L'ordonnancement est préemptif.

— Le jeu de tâches périodiques est il ordonnançable avec RM en mode préemptif ?
— En plus de T1 et de T2, on suppose que le système comprend les tâches apériodiques ci-dessous :

Nom	Capacité	Date d'arrivée	Priorité	Politique
fifo1	4	5	20	SCHED_FIFO
other1	5	0	0	SCHED_OTHER
fifo2	3	4	5	SCHED_FIFO
rr1	3	3	7	SCHED_RR
rr2	3	4	7	SCHED_RR

— Proposez pour T1 et T2 une priorité qui garantisse le respect de leurs contraintes temporelles conformément à la méthode RM et au système POSIX 1003.1b utilisé.
— Dessinez l'ordonnancement généré par l'ordonnanceur POSIX 1003.1b entre 0 et 30 unités de temps.

4 Exercices avec tâches dépendantes

4.1 Exercice 6 : ressources partagées, producteur/consommateur

Dans cet exercice, on souhaite montrer l'impact d'une inversion de priorité sur l'ordonnancement d'un jeu de tâches qui implantent un producteur-consommateur. Soient trois tâches définies par les paramètres suivants : $S_1 = S_2 = S_3 = 0$, $P_1 = 6$, $C_1 = 2$, $P_2 = 8$, $C_2 = 2$, $P_3 = 12$ et $C_3 = 5$. Les délais critiques sont égaux aux périodes (soient $\forall\ i\ :\ D_i = P_i$).

On utilise un algorithme à priorité fixe et RM pour ordonnancer les tâches (mode préemptif).

La tâche T1 produit des données que la tâche T3 consomme. Ces données sont stockées dans une mémoire commune que les tâches T1 et T3 accèdent en exclusion mutuelle. T1 accède à la ressource durant la deuxième unité de temps de sa capacité. T3 accède à la ressource durant la totalité de sa capacité.

1. Dessinez sur la période d'étude l'ordonnancement généré par un algorithme à priorité fixe et RM. Vous indiquerez les moments d'accès exclusif à la ressource ainsi que le moment où l'inversion de priorité intervient.
2. On suppose que l'on utilise la méthode d'héritage simple : une tâche qui bloque une autre plus prioritaire qu'elle, exécute la section critique avec la priorité de la tâche bloquée. Cette méthode d'héritage s'appele aussi PIP pour Priority Inheritance Protocol. Redessinez l'ordonnancement sur la période d'étude et indiquez le moment sur le graphe où l'inversion de priorité est évitée.
3. Donnez la valeur de B_1, B_2 et B_3.

4.2 Exercice 7 : comparaison PIP et ICPP

L'exercice précedent présentait le protocole d'héritage simple (aussi appelé PIP pour Priority Inheritance Protocol). Dans cet exercice, nous montrons un des défauts de ce protocole et une de ses évolutions très souvent implantée dans les systèmes d'exploitation temps réel : le protocole ICPP (pour Immediate Ceiling Priority Protocol).

Soient deux tâches définies par les paramètres suivants : $S_1 = 0, S_2 = 2, P_1 = 31, C_1 = 8, P_2 = 30$ et $C_2 = 8$. Les délais critiques sont égaux aux périodes (soient $\forall\ i\ :\ D_i = P_i$). On utilise un algorithme à priorité fixe et Rate Monotonic pour ordonnancer les tâches (mode préemptif).

Les tâches T1 et T2 accèdent à 2 ressources partagées : R1 et R2. T1 accèdent à R1 de la 2ème unité de temps de sa capacité à la 8ème incluse. T1 accèdent à R2 de la 4ème unité de temps de sa capacité à la 8ème incluse. T2 accèdent à R1 de la 6ème unité de temps de sa capacité à la 8ème incluse. T2 accèdent à R2 de la 2ème unité de temps de sa capacité à la 8ème incluse.

1. Dessinez sur le 30 premières unités de temps l'ordonnancement généré par RM en appliquant le protocole PIP. Que se passe t il de remarquable ?

   On suppose maintenant que l'on applique le protocole ICPP qui fonctionne de la façon suivante :
   — ICPP est une variante du protocole PCP.
   — Une priorité (information désignée par le terme de **plafond de priorité**) est associée à chaque ressource. Cette priorité est égale à la plus grande priorité des tâches qui accèdent à la resource. Cette information est statique.
   — Chaque tâche a une priorité statique, affectée selon Rate Monotonic par exemple.
   — On ordonnance les tâches selon une priorité dynamique.
   — Priorité dynamique d'une tâche = maximum (priorité statique de la tâche, priorité plafond de toutes les ressources allouées).
   — La priorité des tâches évolue donc au fur et à mesure des allocations et libérations des ressources partagées.
   — On sait alors que B_i = plus grande section critique.
2. Re-dessinez le chronogramme sur les 30 premières unités de temps lorsque les ressources sont gérées grâce à ICPP.

5 Exercices complémentaires

5.1 Exercice 8 : exemple d'examen

On étudie dans cet exercice un système de visualisation intégré dans un véhicule automobile. Ce système est constitué d'un ensemble de tâches. Dans un premier temps, on étudie le respect des contraintes temporelles des tâches. Puis, on étudie les différentes synchronisations entre les tâches grâce aux réseaux de Petri.

Cette application est constituée de 5 traitements :

1. La tâche $CAPTEUR\_1$ qui lit toutes les 10 milli-secondes la vitesse du véhicule.
2. La tâche $CAPTEUR\_2$ qui lit toutes les 10 milli-secondes la température dans l'habitacle du véhicule.
3. La tâche $CAPTEUR\_3$ qui lit toutes les 40 milli-secondes la position GPS du véhicule.
4. La tâche $AFFICHAGE\_1$ qui produit toutes les 12 milli-secondes, un résumé des informations produites par les tâches $CAPTEUR\_1$, $CAPTEUR\_2$ et $CAPTEUR\_3$ sur un écran LCD.
5. La tâche $AFFICHAGE\_2$ qui affiche à la demande de l'utilisateur une carte routière. L'affichage doit être réactualisé toutes les 6 milli-secondes.

Première partie : étude des contraintes temporelles

Les tâches $CAPTEUR\_2$ et $AFFICHAGE\_1$ ont une durée d'exécution bornée par 2 milli-secondes. La tâche $CAPTEUR\_3$ requiert 4 milli-secondes pour exécuter une de ses activations. Enfin, les tâches $CAPTEUR\_1$ et $AFFICHAGE\_2$ ont une durée d'exécution inférieure à une milli-seconde.

On suppose que toutes les tâches démarrent à un instant identique. Enfin, on souhaite que les tâches terminent une activation donnée avant le début de leur activation suivante.

Le système utilisé propose un ordonnanceur préemptif à priorité fixe comprennant 32 niveaux de priorité (de 0 à 31). 0 est le niveau de priorité le plus fort. Le système ne permet pas l'affectation d'une priorité identique à plusieurs tâches.

Question 1.1

— En général, quels sont les critères utilisés pour fixer la priorité des tâches ?
— Déterminez les différents paramètres des tâches nécessaires à leur ordonnancement. Vous motiverez le choix de la priorité.

Question 1.2

A partir des priorités déterminées dans la question précédente, calculer les temps de réponse des tâches $CAPTEUR\_1$, $CAPTEUR\_2$, et $CAPTEUR\_3$.

Question 1.3

En fait, les tâches $CAPTEUR\_1$, $CAPTEUR\_2$, et $CAPTEUR\_3$ partagent une zone de mémoire accédée en exclusion mutuelle. $CAPTEUR\_1$ et $CAPTEUR\_2$ accèdent à cette ressource pendant toute leur capacité. $CAPTEUR\_3$ accède à la ressource pendant les deux premières milli-secondes de sa capacité. Le sémaphore utilisé applique le protocole à héritage de priorité simple.

Expliquer pourquoi les temps de réponse calculés dans les questions précédentes ne constituent plus des pires cas.

Question 1.4

Donnez le chronogramme décrivant l'ordonnancement des tâches de l'instant 0 à l'instant 25. Vous signalerez les instants où la ressource est allouée et libérée.

Question 1.5

Dans les questions 1.5 et 1.6, on considère que les tâches ne partagent plus de ressource.

Afin d'activer les tâches deux fois plus rapidement, on souhaite tester une configuration où les tâches sont réparties sur deux processeurs (les processeurs a et b).

Les tâches sont maintenant définies par les paramètres suivants :

Tâches	Processeur utilisé	Capacité	Période
$AFFICHAGE\_1$	b	4	6
$CAPTEUR\_3$	b	2	20
$AFFICHAGE\_2$	a	2	3
$CAPTEUR\_1$	a	2	5
$CAPTEUR\_2$	a	2	5

Les tâches restent ordonnancées selon un algorithme à priorité fixe tel que celui défini au début de cet exercice. De plus, on souhaite que les priorités soient affectées selon l'algorithme Rate Monotonic.

Démontrez qu'il existe au moins une des tâches qui ne respecte pas ses contraintes temporelles.

Question 1.6

En fait, l'affectation des tâches aux processeurs a et b a été réalisée au hasard. En effet, toutes les tâches de notre système peuvent à la fois s'exécuter sur le processeur a ou sur le processeur b. Les processeurs diffèrent uniquement par leur rapidité d'exécution : le processeur b est 2 fois plus rapide que le processeur a.

Affectez les tâches aux processeurs a et b de sorte que les contraintes temporelles de **toutes** les tâches soient respectées. **Justifiez vos choix.**

Deuxième partie : étude du comportement logique de l'application

Dans cette partie, on ne tient plus compte du caractère périodique des tâches de notre véhicule automobile : les tâches sont maintenant apériodiques.

On regarde les synchronisations des tâches $CAPTEUR\_1$, $CAPTEUR\_2$ et $CAPTEUR\_3$ lorsqu'elles accèdent à leurs ressources partagées.

Dans cet exercice, les tâches partagent 2 ressources $R1$ et $R2$. Pour son fonctionnement, $CAPTEUR\_1$ a besoin indifféremment de l'une des 2 ressources. $CAPTEUR\_2$ n'a besoin et ne peut utiliser que la ressource $R2$. Quant à $CAPTEUR\_3$, elle a besoin des deux ressources pour fonctionner.

1. Sachant que $CAPTEUR\_3$ ne prend pas les deux ressources simultanément, mais qu'elle les libère simultanément, proposer un modèle de fonctionnement du système à l'aide des RdP.
2. En vous appuyant sur le graphe des marquages accessibles, montrer que le RdP n'est pas sans blocage.
3. Proposer 2 solutions permettant d'éviter des situations de blocage dans le fonctionnement du système. On représentera les RdP correspondants.

5.2 Exercice 9 : comparaison des algorithmes EDF et à priorités fixes

Soient deux tâches T1 et T2 définies par les paramètres suivants : $S_1 = S_2 = 0, P_1 = 8, P_2 = 10, C_1 = 4$ et $C_2 = 5$. Les délais critiques sont égaux aux périodes (soient $\forall\ i\ :\ D_i = P_i$).

1. Dessinez sur les 24 premières unités de temps l'ordonnancement généré par EDF (en mode préemptif). Existe il des échéances manquées ?
2. Dessinez sur les 24 premières unités de temps l'ordonnancement généré par un algorithme à priorités fixes et RM (en mode préemptif). Existe il des échéances manquées ?
3. Que constatez vous ? est ce surprenant ?

5.3 Exercice 10 : comparaison des algorithmes EDF et LLF

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.

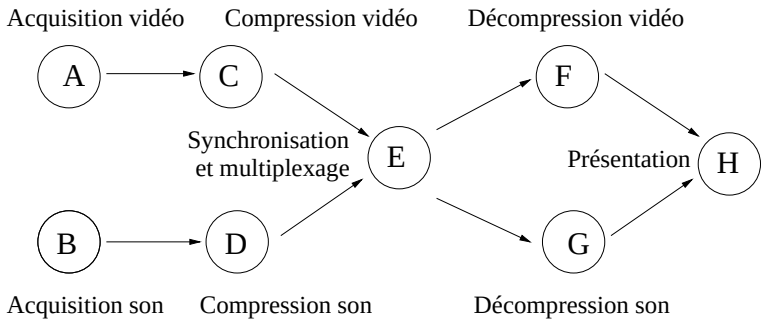

FIGURE 2 – Une application multimédia

5.4 Exercice 11 : contraintes de précédence

On s'intéresse à une application multimédia dont les traitements sont décrits par la figure 2. L'application est constituée de 8 tâches soumises à des contraintes de précédence. Les paramètres temporels de cette application sont donnés ci-dessous (exprimés en milli-secondes) :

Tâches	S_i	C_i	D_i
A	0	3	12
B	0	1	10
C	0	3	20
D	0	1	15
E	0	5	25
F	0	4	37
G	0	1	20
H	0	1	40

L'application étant par nature dynamique, nous utilisons un algorithme EDF préemptif. Le graphe de tâches est activé périodiquement toutes les 40ms.

1. Calculez le taux d'utilisation. Le jeu de tâches est il ordonnançable si l'on ne tient pas compte des contraintes de précédence ? Rappel : un taux d'utilisation ne peut être exploité que sur des tâches indépendantes.
2. Confirmez en dessinant la séquence d'ordonnancement sur la période d'étude. L'ordonnancement généré par EDF est il conforme aux besoins de l'application ?
3. Utilisez la méthode de Blazewicz/Chetto pour supprimer les contraintes de précédence.
4. Calculez de nouveau le taux d'utilisation et redessinez l'ordonnancement ainsi généré sur sa période d'étude. Conclure sur l'ordonnançabilité de l'application.

5.5 Exercice 12 : les p'tits navions

On souhaite organiser le partage d'une piste de l'aéroport de Brest entre deux compagnies aériennes. La première compagnie dessert une ligne nationale vers Paris. La seconde dessert une ligne internationale.

La piste est allouée à un avion pendant son atterissage mais aussi pendant sa phase d'approche. Ces deux étapes durent respectivement 20 minutes et 10 minutes et ce quelque soit l'avion.

Pour la ligne nationale, un atterrissage s'effectue toutes les deux heures. La ligne internationale requiert un atterissage toutes les cinq heures. En outre, une fois par heure, la piste doit être inspectée par une équipe d'entretien au sol. Ce travail d'entretien dure 30 minutes.

— Proposez une solution pour partager la piste entre les différents intervenants pendant une journée complète. Vous donnerez la liste des tâches de ce système ainsi que le chronogramme décrivant le partage de la piste.

Vous utiliserez un des algorithmes d'ordonnancement étudié en cours. **Vous justifierez le choix de l'algorithme et la façon dont vous modélisez les tâches. Tout résultat non justifié sera compté comme faux.**

On souhaite modéliser l'entretien de la piste non pas par une tâche mais par quatre tâches. En effet, en pratique, l'entretien de la piste est réalisé en quatre phases (dans cet ordre) :

— Une phase préparatoire (distribution du matériel, etc) qui dure 5 minutes.
— L'entretien à proprement dit qui est effectué par deux équipes :

1. La première vérifie l'absence d'objet sur la piste. Elle doit terminer son travail 15 minutes après le début de l'entretien de la piste.
2. La deuxième s'occupe des systèmes de signalisation. Elle doit terminer son travail 30 minutes après le début de l'entretien.

Chaque équipe requiert 10 minutes pour effectuer sa tâche.
— Une phase de contrôle effectuée par une troisième équipe qui intervient lorsque les deux premières équipes ont terminé leur tâche. Le contrôle dure 5 minutes.

On suppose que ces 4 phases ne peuvent utiliser la piste en même temps.

Déterminez les paramètres de ces quatre tâches. **Justifiez vos réponses.**

6 Références bibliographiques

Les exercices de ce TD sont inspirés de ceux trouvés dans [COT 00, DEM 99, BUR 97, KRI 97]. A consulter pour des exercices supplémentaires.

[AUD 93] N. Audsley, A. Burns, M. Richarson, K. Tindel, and J. Wellings. « Applying New Scheduling Theory to static priority pre-emptive scheduling ». *Software Engineering Journal*, 8(5) :284–292, 1993.

[BUR 97] A. Burns and A. Wellings. *Real-time Systems and Programming Languages.* Addison Wesley, 1997.

[COT 00] F. Cottet, J. Delacroix, C. Kaiser, and Z. Mammeri. *Ordonnancement temps réel.* Hermès, 2000.

[DEM 99] I. Demeure and C. Bonnet. *Introduction aux systèmes temps réel.* Collection pédagogique de télécommunications, Hermès, septembre 1999.

[JOS 86] M. Joseph and P. Pandya. « Finding Response Time in a Real-Time System ». *Computer Journal*, 29(5) :390–395, 1986.

[KRI 97] C. M. Krishna and K. G. Shin. *Real-Time Systems.* Mc Graw-Hill International Editions, 1997.