Ordonnancement sous contraintes temps réel *(module TRS3 - DEA informatique UHP Nancy I)*

Nicolas NAVET

http ://www.loria.fr/~nnavet

Table des matières

Résumé

Ordonnancer c'est définir un ordre sur l'exécution des différentes activités d'un système. Lorsqu'il s'agit d'un système informatique, ces différentes activités se décomposent en l'exécution de tâches sur un ou plusieurs processeurs et éventuellement en la transmission de messages sur un réseau de communication. Un système informatique temps réel a cette particularité que son bon fonctionnement dépend non seulement de la correction logique de ses résultats mais également des instants de production des résultats ; les instants de production doivent être prévisibles d'un point de vue temporel. Cette nécessaire prévisibilité s'exprime sous la forme de contraintes de temps issues du processus physique controlé. Pour respecter ses contraintes, il s'agira de trouver un ordre adéquat sur les différentes activités du système, c'est à dire les *ordonnancer*. Dans le contexte de l'ordonnancement, un système est dit *faisable* ou *ordonnançable* si toutes les invocations successives des activités qui le composent sont assurées de respecter leurs contraintes temporelles qui seront généralement des contraintes d'échéances.

De façon informelle, *une politique d'ordonnancement* est un algorithme qui va décider de la façon d'occuper la ressource en fonction des activités en concurrence. Cette décision peut-être prise *hors-ligne* en construisant avant la mise en marche de l'application une séquence d'ordonnancement qui est amenée à se répéter, ou *en-ligne* c'est à dire pendant l'exécution même du système. Dans ce cours, nous étudierons exclusivement l'ordonnancement en-ligne qui offre plus de flexibilité et de puissance dès que le système envisagé est un tant soit peu complexe. Le lecteur interessé par un comparatif des deux techniques pourra se référer à [9].

Ce cours traite de l'ordonnancement sous contraintes de temps d'un ensemble *d'activités récurrentes*, des tâches ou des messages, sur une ressource unique qui pourra être un ordinateur mono-processeur ou

un médium de communication partagé. Les tâches récurrentes (ou cycliques) sont primordiales dans les systèmes informatique temps réel car ceux-ci sont généralement implantés de façon à interagir régulièrement avec leur environnement, le plus souvent de façon strictement périodique, par la lecture de capteurs ou l'application de consignes.

Nous présentons un cadre formel qui permet l'étude rigoureuse des politiques d'ordonnancement et notamment le calcul de bornes sur les temps de réponse ce qui permet de décider de la faisabilité du système. Notre premier objectif est de montrer comment ce modèle mathématique permet d'unifier les résultats d'ordonnancement existants en les présentant dans un cadre formel cohérent. Mais l'objectif principal de ce modèle est bien de nous donner les outils et les techniques pour appréhender efficacement des problèmes non-encore résolus en ordonnancement comme l'analyse de nouvelles politiques ou la prise en compte de nouveaux modèles de tâches.

Nous nous efforcerons de présenter et de redémontrer des résultats fondamentaux de l'ordonnancement de tâches, en particulier certains théorèmes énoncés par Lui et Layland dans [8] concernant les politiques *Fixed Priority Preemptive* (FPP) et *Earliest Deadline First* (EDF). Nous traiterons également de l'ordonnancement non-preemptif à priorités fixes (*Non Preemptive Fixed Priority* - NPFP) avec le réseau de multiplexage véhicule CAN (*Controller Area Network*) comme cadre d'application. Dans la pratique, la majorité des systèmes d'exploitatioDn temps réel, et notament ceux se conformant au standard temps réel *Posix1003.1b*, offrent plusieurs politiques d'ordonnancement et nous proposerons des techniques permettant l'analyse d'ordonnançabilité de tels systèmes. Nous nous intéresserons ensuite aux systèmes dans lesquels deux types de contraintes coexistent, des *contraintes strictes* qui doivent être nécessairement respectées et des contraintes dites "*souples*" qui peuvent être occasionnellement non-respectées et pour lequelles nous envisagerons des solutions d'ordonnancement de type "*best effort*".

Pour des questions, des suggestions ou pour signaler d'éventuelles erreurs et imprécisions, vous pouvez me contacter par mail (Nicolas.Navet@loria.fr). Les exercices de ce cours sont étiquettés en fonction de leur niveau de difficulté :

$*$	exercice ou problème simple dont la résolution est une application directe du cours,
$**$	problème de difficulté intermédiaire,

$* * *$	problème relativement difficile (la difficulté est d'ordre technique ou l'idée conduisant à la solution n'est pas évidente),
$* * **$	problème pour lequel aucune solution satisfaisante n'est connue par les auteurs à ce jour.

Chapitre 1

Conception des systèmes temps réel

Ce chapitre est une introduction à la conception des systèmes temps réel. A partir d'un exemple concret du domaine de l'automobile, nous verrons comment une contrainte de temps se décompose sur l'ensemble des constituants d'une architecture informatique. Nous présenterons les différentes techniques qui permettent de vérifier le bon respect des contraintes de temps, c'est à dire de *valider* l'architecture envisagée. Parmi ces techniques, l'analyse d'ordonnançabilité qui permet, sous certaines hypothèses de bon fonctionnement, de vérifier le respect des contraintes de temps d'un ensemble d'activités. Nous en étudierons les principes et les limites. Enfin nous présenterons de façon informelle le modèle mathématique qui sera utilisé dans la suite de ce cours.

1.1 Prise en compte des contraintes de temps dans le cycle de vie d'une application

Un système informatique dont le bon fonctionnement ne dépend pas uniquement de la justesse des calculs mais aussi des instants de production de ces calculs est désigné sous le terme générique de *système temps réel* [16]. Notons que si l'on s'accorde généralement sur la véracité de cette définition quelque peu minimaliste, la littérature contient une pléthore d'autres définitions (plus d'une dizaine de définitions différentes sont répertoriées dans [17]), qui insistent plus particulièrement sur certaines caractéristiques du temps réel comme la nécessaire *prévisibilité* du système ou l'aspect *réactif* de l'application vis-à-vis de son environnement.

Classiquement, on décompose une application temps réel en deux entités : *une partie opérative* qui est le processus physique effectivement contrôlé et *une partie commande* qui est le système informatique assurant le contrôle de la partie opérative. La partie commande prend régulièrement connaissance de l'état du processus physique par le biais de valeurs provenant de capteurs, calcule la prochaine action à effectuer sur la base des mesures antérieures et du modèle de comportement attendu et répercute ses consignes sur le système contrôlé par le biais d'actionneurs. La partie commande de l'application peut être répartie sur différents sites, par exemple pour être capable de répondre simultanément à plusieurs sollicitations de l'environnement, pour dupliquer des fonctions et ainsi pouvoir pallier d'éventuelles défaillances ou tout simplement parce que la topologie du processus contrôlé l'impose. La coopération est alors assurée par l'échange de données sur un réseau de communication. A chaque instant, les valeurs de ces données constituent une image de l'état du processus commandé ainsi que de l'application elle même.

Les contraintes de temps pesant sur les activités d'une application temps réel sont issues de la dynamique du processus physique contrôlé et ces contraintes temporelles devront être prises en compte tout au long du cycle de vie de l'application. Classiquement le cycle de vie d'une application se décompose en sept étapes qui sont la rédaction du cahier des charges, la spécification, la conception, l'implantation, l'intégration, l'exploitation et la maintenance. Les contraintes de temps sont présentes sous une forme plus ou moins formelle dès la phase d'analyse des besoins, c'est à dire lors de l'élaboration du cahier des charges. Par exemple, si l'on considère un système de contrôle aérien, une contrainte plausible est que "la position des avions sur l'écran radar devra être mise à jour toutes les 100ms".

La vérification des propriétés temporelles commence effectivement lors de la phase de spécification dont l'objectif est la définition d'une *architecture fonctionnelle* validée, c'est à dire sans ambiguïtés, sans incohérences et complète [2]. L'Architecture Fonctionnelle (ou AF) décrit formellement la structure d'une application et son comportement attendu, c'est à dire l'ensemble des services et traitements respectivement à fournir et effectuer par les fonctions ainsi que les échanges de données inter-fonctions. L'AF ne tient compte ni de la répartition des fonctions sur les sites ni de l'*Architecture Matérielle* (AM) support définie comme l'ensemble des composants qui assurent les ressources matérielles et logicielles nécessaires à l'exécution de l'application (e.g. machines, OS, réseaux, protocoles,...). L'indépendance de l'AF vis-à-vis du support exécutif permet d'une part de réduire la complexité du problème général et d'autre part de faciliter la réutilisation d'AFs existantes. Néanmoins, dans cette étape du cycle de vie, une validation ne peut être menée qu'en faisant des hypothèses fortes sur les durées de traitement et donc implicitement sur les performances. Les vérifications sur l'AF portent dont essentiellement sur la correction "logique" du système, comme par exemple, l'absence d'interblocage, la terminaison des traitements ou l'équité dans l'accès à une ressource.

Une *Architecture Opérationnelle* (AO) validée est l'objectif attendu de la phase de conception du système. L'AO est le résultat de la projection de l'AF sur l'AM support, les composants de cette dernière étant choisis relativement à un ensemble de critères issus du cahier des charges parmi lesquels les performances évidemment, le coût, qui pour les productions de masse revêt une importance toute particulière, mais aussi par exemple, des garanties sur la pérennité de la production. La projection comprend le placement des fonctions sur les différents sites, leur éventuelle subdivision en différentes tâches (au sens informatique du terme) mais aussi la fixation des paramètres des mécanismes exécutifs associés à l'implantation, comme les ordonnanceurs de tâches et les protocoles de communication. Ainsi à chaque tâche, il faudra vraisemblablement allouer une priorité et éventuellement décider d'une politique d'ordonnancement si l'OS support permet un choix comme c'est le cas dans les exécutifs se conformant au standard *Posix1003.1b* (étudié dans le chapitre ?). De façon similaire, selon le protocole de transmission de données, il peut être nécessaire de fixer certains paramètres de la communication comme la priorité des trames échangées.

Il s'agit ensuite de s'assurer que l'AO proposée vérifie les contraintes de l'application, c'est la phase cruciale de validation de l'AO. Il est évident que la remise en cause éventuelle de choix effectués en termes d'AM, de répartition ou de fixation de paramètres sera plus facile et moins coûteuse lors de l'étape de conception que plus en aval dans le cycle de vie de l'application et particulièrement pendant l'exploitation du système.

1.2 Contraintes de temps

Comme cela a été déjà précèdemment évoqué, les contraintes de temps pesant sur les applications temps réel sont issues du processus physique sous-jacent. Plaçons nous par exemple dans le cadre de l'automobile en considérant les deux organes majeurs que sont une Boite de Vitesse automatique (BV) et le Contrôle Moteur (CM) reliés par un réseau de communication. Après un changement de vitesse, initié sur le site BV, le contrôle moteur doit effectuer un certain nombre d'ajustements comme par exemple procéder à une réduction de couple. Le délai entre l'occurrence de l'évenement "changement de vitesse" et la réduction effective du couple doit toujours rester inférieur à une certaine valeur (fixée par les concepteurs du véhicule en fonction du grand nombre d'impératifs dont des contraintes mécaniques et de confort). Il s'agit en l'occurrence d'une contrainte stricte de date au plus tard que nous qualifierons de contraintes sur un *temps de réaction* ($\tau_{reaction}$, cf. figure 1.1), qui se décompose en le *temps de réponse de bout-en-bout du système informatique* (R_I) plus le *temps de réponse de la partie opérative* (R_Ψ), c'est à dire le délai entre l'application des consignes (e.g. fermeture d'un clapet) sur le site CM et la baisse effective de couple au niveau requis. Nous nous intéresserons plus particulièrement au délai R_I, R_Ψ se situant en dehors de notre domaine de compétences. Néanmoins pour vérifier le respect de la contrainte associée à $\tau_{reaction}$, il faudra nécessairement avoir la connaissance d'une borne sur R_Ψ qui nous sera fournie par des spécialistes de la partie opérative.

D'un point de vue implémentation, imaginons qu'après un changement de vitesse une tâche devienne active sur le site BV ; elle aura, entre autres charges, d'émettre un message à la fin de son exécution à destination du CM pour l'informer de la situation courante. Sur le site CM, après réception du message, la tâche qui devra éventuellement agir sur les actionneurs, se déclenche. On peut décomposer cette contrainte de temps de réponse de bout-en-bout du système informatique en :

– le délai de production du message τ_G : c'est le délai entre le changement de vitesse et la fin d'exécution de la tâche productrice du message destiné au CM, il se décompose lui même en :

  1. le temps de prise en compte de l'événement, noté τ_{p_msg}. D'un point de vue implémentation, deux stratégies sont possibles : (1) la tâche est périodique et τ_{p_msg} est alors égal à la différence entre la prochaine activation de la tâche et la date du changement de vitesse, (2) la tâche est déclenchée sur

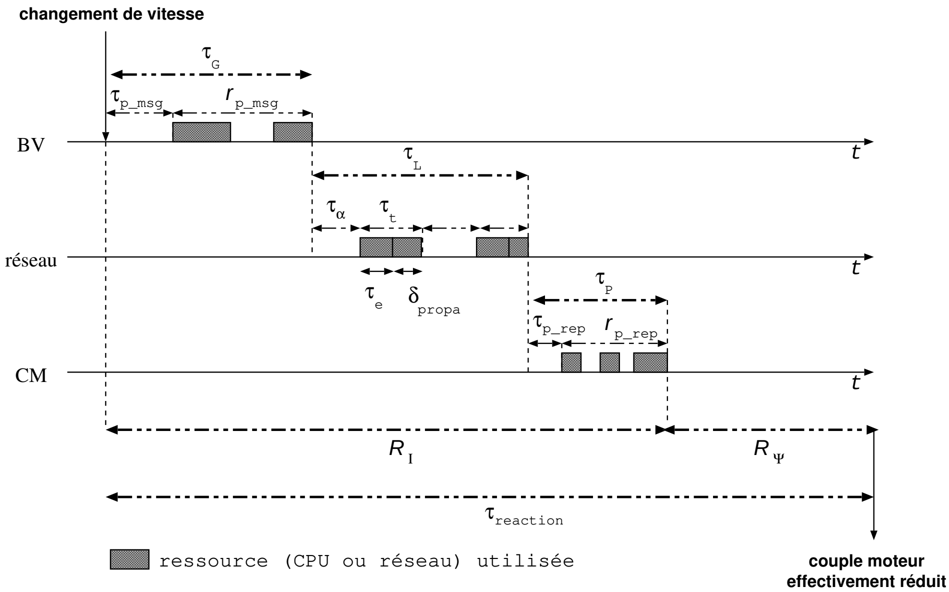

FIG. 1.1 – Décomposition du temps de réaction.

l'occurrence d'un événement, par exemple une interruption matérielle (signalant le changement de vitesse), et τ_{p_msg} sera donc le temps de prise en compte de cet événement,

2. le temps de réponse de la tâche productrice, noté r_{p_msg} fonction essentiellement de la durée d'exécution de la tâche, des autres tâches du site et de la, ou des, politique(s) d'ordonnancement(s).

– le temps de latence du message, noté τ_L qui est le délai entre la fin d'éxécution de la tâche productrice et la réception complète du message par le site consommateur, en l'occurrence le site CM. Le message sera segmenté en n trame(s), où n est fonction du protocole de communication et de la taille des données. Le temps de latence du message est dépendant du protocole de communication utilisé mais si l'on considère par exemple un bus à priorité tel que CAN[6], standard de fait dans l'industrie automobile, chaque trame i sera reçue par le site CM après un délai égal à la somme du temps requis pour gagner l'accès au bus (τ_i^α) et du temps de transmission effectif de la trame (τ_t^i). Ce temps de transmission étant classiquement composé du temps d'émission des bits composant la trame (τ_e^i) et du délai de propagation δ_{propa}. On a donc pour un bus à priorité $\tau_L = \sum_{i=1}^{n} \tau_\alpha^i + \tau_t^i$ avec $\tau_t^i = \tau_e^i + \delta_{propa}$.
– le délai de production de la réponse, noté τ_P, qui est le délai entre

l'arrivée complète du message et la fin d'exécution de la tâche consommatrice du message qui appliquera les consignes sur le moteur. Comme pour la tâche productrice, outre le temps de réponse de la tâche (r_{p_rep}), on considérera un délai (τ_{p_rep}) entre la fin d'arrivée du message et l'activation effective de cette tâche.

On identifie naturellement d'autres contraintes temporelles que celle de date au plus tard. En particulier, on peut citer les contraintes de date au plus tôt ("une tâche ne doit pas s'exécuter avant l'instant t"), de simultanéité sur des occurrences d'événements ou de gigues ("le délai entre l'affichage de deux images consécutives d'un flux vidéo doit être compris entre x et y millisecondes"). Néanmoins, la contrainte de date au plus tard est au coeur de la problématique du temps réel puisqu'elle apparaît même explicitement dans de nombreuses définitions des systèmes temps réel sous le terme d'échéances (*deadlines*). Cette contrainte de date au plus tard sera au centre de nos préoccupations dans la suite de ce document.

1.3 Validation de l'architecture

À l'issue de la phase de spécification de l'application, les fonctions, les flux de données entre fonctions ainsi que les contraintes de temps associées devront être clairement identifiés pour la partie temps réel stricte du système. En partant de cette spécification fonctionnelle, des choix liés à l'implémentation seront faits : comment implémenter d'un point de vue informatique les fonctions ? Combien de tâches (au sens de processus) et pour chacune quelles contraintes de temps ? Dans le cas d'une application distribuée, sur quel processeur exécuter chacune des tâches ? Enfin pour chaque tâche, il faudra se poser la question de la priorité, de la politique d'ordonnancement et du mécanismes d'activation (activation périodique ou activation sur l'occurrence d'un événement ?). Des réponses à ces questions doivent être apportées par les concepteurs de d'une application et ceux-ci devront s'assurer que leurs choix en matière de conception garantissent le respect des contraintes de l'application et notament de ses contraintes de temps.

La validation de l'architecture envisagée (l'Architecture Opérationnelle - AO) est le processus qui permet de s'assurer que le système a les performances requises pour garantir le respect des contraintes de l'application et notament les contraintes temporelles. Cette validation pourra être effectuée par l'observation sur prototypes ou maquettes et/ou à l'aide de modèles. Généralement, les contraintes de coûts et de temps, imposent de travailler sur des modèles du système réel dès que celui-ci est un tant soit peu complexe. En recoupant les définitions de [12]et de [15], nous définissons *un modèle comme une idéalisation du système réel nous permettant d'apprendre quelque chose d'utile sur son fonctionnement*. En l'occurrence, nous nous attacherons à vérifier, en exploitant le modèle, que l'application respecte bien ses contraintes et le cas échéant modifierons l'AO envisagée. Ce processus itératif de modélisation - exploitation du modèle - modification de l'AO se poursuit jusqu'à ce que l'on obtienne une AO validée ou, dans le cas défavorable, jusqu'à ce que l'on dresse un constat d'échec qui nous obligera à revenir en amont dans le cycle de vie, à la phase de spécification ou même au cahier des charges pour éventuellement relaxer certaines contraintes. Nous distinguerons deux grandes classes de modèles, les modèles de simulation, c'est à dire les modèles qui se "résolvent" par simulation, et les modèles analytiques dont la résolution est purement calculatoire comme les analyses d'ordonnançabilité.

1.3.1 Validation par simulation

L'utilisation de la simulation dans l'objectif d'évaluer les performances d'une AO, et donc sa capacité à respecter les contraintes de l'application, est une pratique extrêmement répandue qui s'impose généralement faute d'alternatives :

– des mesures sur un système réel nécessite sa réalisation et son instrumentation ce qui est long et coûteux et qui ne peut intervenir que loin en avant dans la phase de conception,
– les modèles analytiques ne sont, en règle générale, pas capables de refléter toute la complexité d'une AO ni de donner des renseignements sur toutes les métriques de performances imaginables : ils ne peuvent modéliser que des systèmes relativement peu complexes ou des sous-ensembles de systèmes complexes et sont donc généralement intrinsèquement insuffisants.

Néanmoins, nous pensons que la simulation seule n'est pas, pour les applications temps réel, une technique de vérification suffisante, et ce à plusieurs titres :

– il pèse toujours une incertitude forte sur la correction de l'implantation d'un modèle et sur sa pertinence et donc sur la validité même des résultats de simulation,
– la simulation d'une AO fait nécessairement des hypothèses sim-

plificatrices sur le système physique contrôlé. Ainsi, le modèle de l'AO du système de contrôle-commande embarqué dans un véhicule intègre un modèle, généralement simpliste, du comportement des différents organes du véhicule,
– les résultats dans le pire des cas (comme les temps de réponse maximaux) n'ont qu'une valeur indicative puisque simuler consiste à dérouler un certain nombre de trajectoires qui dans le cas général, est infiniment inférieur au nombre de trajectoires possibles.

Dans une optique temps réel et sûreté de fonctionnement, le dernier argument en défaveur de la simulation est le plus fort puisque, avant de penser aux performances dans le cas probable, c'est d'abord le pire cas, même si il ne survient qu'exceptionnellement, qui nous intéresse.Le concepteur aura donc tout intérêt à coupler les techniques de validation, coupler les modèles de simulation avec des modèles analytiques donnant des bornes sur les métriques de performance considérées et par observation sur prototypes, fixer les valeurs de certains paramètres et surtout vérifier que les hypothèses faites lors de la modélisation sont réalistes.

1.3.2 Analyse d'ordonnançabilité : principes et limites

L'analyse d'ordonnançabilité consiste à étudier la faisabilité du système. Il s'agit de vérifier que toutes les instances des activités respectent leurs contraintes sous une politique d'ordonnancement donnée avec des paramètres fixés (priorités des tâches sous FPP, durée d'un quantum de temps sous une politique de type Round-Robin ..). En pratique, une analyse d'ordonnançabilité est soit un calcul explicite de bornes sur les temps de réponse, soit un test de faisabilité parfois simplement basé sur la charge globale du système comme c'est le cas pour les politiques FPP et EDF. L'intérêt majeur de ces tests de faisabilité tient en leur complexité algorithmique restreinte : ils peuvent être linéaires en le nombre de tâches là où un calcul explicite de temps de réponse a une complexité polynomiale. Par contre ils ne sont parfois que suffisants ; certains ensembles de tâches parfaitement faisables peuvent ne pas passer le test.

L'analyse d'ordonnançabilité ne résout pas à elle seule tous les problèmes de conception liés à l'ordonnancement, en particulier
– elle ne répond pas à la question cruciale du choix de la ou des politiques d'ordonnancement et de leurs paramètres. Une analyse d'ordonnançabilité ne propose pas de solutions faisables, elle se

prononce simplement sur la faisabilité d'une allocation de politiques et de priorités qui lui aura été soumise,
- elle est d'une aide limitée pour l'optimisation du système à cause des hypothèses nécessairement pessimistes sur les pires temps d'exécution et puisque la seule trajectoire du système qui est étudiée est la plus pessimiste en termes de flux d'arrivée de travail.

Si par exemple le travail d'une tâche doit être rendu le plus régulièrement possible, il est souhaitable de minimiser la variabilité sur ses dates de fins d'exécution (la *gigue*). L'allocation de priorités et de politique influera naturellement sur la qualité du système vis-à-vis de ce critère. Pour décider des bonnes politiques d'ordonnancement et de leurs paramètres, il est parfois nécessaire de faire appel à des techniques d'optimisation comme les algorithmes génétiques compte tenu de la complexité algorithmique des problèmes. Néanmoins il faudra toujours faire appel à l'analyse d'ordonnançabilité pour distinguer les configurations faisables des configurations non-faisables et utiliser la simulation ou des mesures sur plate-formes pour évaluer la qualité de chaque configuration faisable. Le lecteur pourra par exemple consulter [4] et [14].

1.4 Présentation du modèle

Les premiers travaux en ordonnancement temps réel de tâches [8] ont porté sur des tâches périodiques à échéances sur requêtes ordonnancées sous FPP et EDF. Ces travaux ont éte ultérieurement adaptés et étendus pour prendre en compte des politiques et surtout des modèles de tâches répondant à d'autres besoins. Par exemple, dans [7] est traité le cas de tâches périodiques à échéance arbitraires (pas nécessairement égales à la période). Nous présentons dans ce cours un modèle mathématique d'une ressource qui exécute selon une certaine politique les instances recurrentes d'un ensemble de tâches. L'objectif est d'unifier les résultats existants et surtout de faciliter l'analyse de nouvelles politiques et de nouveaux types de tâches.

Inspiré du concept de *processus ponctuel marqué* [1] qui est un modèle stochastique pour les systèmes à événement discrets, le modèle que nous utiliserons a été initialement développé dans [3] puis étendu dans [11]. Il a été ensuite utilisé pour démontrer certains résultats nouveaux en ordonnancement par exemple dans [13; 5; 10]. L'espace des trajectoires du système va modéliser toutes les évolutions possibles d'un système. Les éléments de base du modèle sont les instances des tâches avec divers attributs comme la *date d'activation* qui détermine à partir de quel moment la tâche est prête à être exécutée, le *temps d'exécution* qui représente le temps de calcul nécessaire à l'instance pour terminer lorsqu'il n'y a pas de préemption et la *date d'échéance*, relative ou absolue, avant laquelle l'instance doit terminer son exécution.

Pour définir les politiques d'ordonnancement, nous associons à toute instance de tâche une *fonction de priorité* qui indique à chaque instant la priorité courante de l'instance. En ligne, l'ordonnancement sélectionne à tout instant *l'instance la plus prioritaire en premier*. L'ordonnancement effectivement réalisé par une politique est exprimé sous la forme d'une *fonction d'ordonnancement* qui spécifie à chaque instant la quantité de processeur utilisée par toute instance du système. La fonction d'ordonnancement d'une instance ne peut prendre que deux valeurs : 1 lorsque l'instance est en cours d'exécution et 0 sinon.

Entre la date d'activation et la fin d'exécution d'une instance, il y a forcément toujours du travail en attente. L'évolution de ce travail en attente est indiqué par la *fonction de charge* comme dans la théorie des files d'attente. Au moment de l'activation, sa valeur devient égale au temps d'exécution de l'instance puis elle décroît ensuite d'une façon qui dépend de la politique et des autres tâches jusqu'à atteindre zéro à la fin d'exécution de l'instance.

La fonction d'ordonnancement et la fonction de charge sont liées par la *fonction d'arrivé de travail* (FAT). La FAT d'une tâche est une fonction croissante en escalier avec des sauts en les instants d'activation des instances dont les hauteurs sont égales au temps d'exécution de l'instance activée. D'une façon générale, le travail en attente (fonction de charge) est la différence entre le travail arrivé (fonction d'arrivé de travail) et le travail effectué (fonction d'ordonnancement). Ces trois concepts permettent de décrire à tout instant l'état du système qui évolue en fonction des tâches en présence et de la politique d'ordonnancement définie par sa fonction de priorités. En particulier, il nous est possible de formuler des équations pour des instants particuliers comme le début et la fin d'exécution ce qui nous permettra de déterminer le temps de réponse des tâches.

Chapitre 2

Le modèle

Ce cours a pour objet l'ordonnancement sous contraintes de temps d'un ensemble d'activités récurrentes sur une ressource. Rentrent naturellement dans ce cadre d'étude l'ordonnancement d'un ensemble de tâches récurrentes sur une ordinateur mono-processeur mais également la transmission de messages périodiques sur un réseau de communication. Dans ce premier chapitre, nous présentons le modèle d'une ressource qui exécute, selon une politique donnée, les instances récurrentes d'un ensemble de tâches. Ce modèle est une structure mathématique, munie d'un certain nombre d'axiomes (indiqués par la notation $\stackrel{\text{def}}{=}$), qui nous fournit les outils pour décrire le système sans ambiguïté et l'analyser. Dans ce chapitre, nous déduirons de ce modèle certaines propriétés de base du système, puis, dans la suite du cours, nous nous attacherons à des caractérisation plus fines avec en particulier la détermination de bornes sur les temps de réponse qui est une problématique centrale dans la conception des systèmes temps réel.

2.1 Modèle de tâches récurrentes

Les activités récurrentes du système, que nous appelerons *des tâches*, constituent un ensemble fini noté $\mathcal{T} = \{\tau_1, \ldots, \tau_m\}$ où τ_k est la tâche d'indice k dont la $n^{\text{ième}}$ instance, notée $\tau_{k,n}$, possède certaines caractéristiques indépendantes de l'ordonnancement qui sera réalisé :

– son *instant de mise à disposition* $A_{k,n}$ (ou *date d'arrivée*) avec $0 \leq A_{k,1} \leq \ldots \leq A_{k,n-1} \leq A_{k,n} \leq \ldots$. C'est le premier instant à partir duquel l'instance $\tau_{k,n}$ est susceptible d'être exécutée,

– son *temps d'exécution* $C_{k,n}$ sur la ressource,

– son *échéance* $D_{k,n}$, c'est à dire l'instant auquel $\tau_{k,n}$ doit avoir fini de s'exécuter. L'échéance relativement à $A_{k,n}$, appelée *échéance relative*, est notée $\overline{D}_{k,n}$ et l'on a $D_{k,n} \stackrel{\text{def}}{=} A_{k,n} + \overline{D}_{k,n}$.
– Le temps entre la date d'arrivée de $\tau_{k,n}$ et la date d'arrivée de la prochaine instance $\tau_{k,n+1}$, appelé *temps interarrivée* ou *temps de cycle*, est noté $T_{k,n}$ avec $T_{k,n} \stackrel{\text{def}}{=} A_{k,n+1} - A_{k,n}$. Le plus souvent les temps interarrivées seront identiques $T_{k,n} = T_k \quad \forall n \in \mathbb{N}$ et la tâche τ_k est alors strictement périodique.

L'ensemble de ces caractéristiques qui ne dépendent pas de la politique d'ordonnancement constitue ce que nous appellerons l*e processus de tâches du système*. Ce seront les données d'entrée de l'algorithme d'ordonnancement et nous parlerons *d'ordonnancement d'un processus de tâches sous une certaine politique d'ordonnancement*.

Définition 1 *Processus de tâches.*

Un processus de tâches ω est défini par une séquence des dates de mise à disposition des instances des tâches (les $A_{k,n}$), par une séquence de temps d'exécution (les $C_{k,n}$) et par une séquence d'échéances (les $D_{k,n}$). L'ensemble des processus de tâches du système est noté Ω.

La $n^{\text{ième}}$ activation d'une tâche τ_k sous un processus de tâches ω devrait être notée $A_{k,n}(\omega)$ mais, pour simplifier les notations, ω n'apparaitra dans les notations que lorsque cela est indispensable.

2.2 Fonction d'ordonnancement

En fonction de la politique d'ordonnancement et des tâches en concurrence, l'allocation de la ressource dans le temps va différer ce qui, en particulier, va influer sur la date de fin d'exécution de toute instance $\tau_{k,n}$ que l'on notera $E_{k,n}$. Pour décrire comment la ressource est allouée, nous introduisons *la fonction d'ordonnancement* qui indique la quantité de ressource attribuée à une instance de tâche au cours du temps. $\Pi_{k,n}(t)$ est la quantité de processeur attribuée à $\tau_{k,n}$ à l'instant t. Nous nous intéresserons qu'au cas $\Pi_{k,n}(t) \in \{0,1\}$, le processeur est entièrement alloué à l'instance $\tau_{k,n}$ ou il ne l'est pas du tout :

$$\Pi_{k,n}(t) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{lorsque } \tau_{k,n} \text{ est exécuté} \\ 0 & \text{lorsque } \tau_{k,n} \text{ n'est pas executé} \end{cases} \tag{2.1}$$

La figure 2.1 représente sur un exemple l'évolution de l'intégrale de $\Pi_{k,n}(t)$ dans le temps et notament entre les instants $A_{k,n}$ et $E_{k,n}$ qui définissent l'intervalle de temps pendant lequel $\tau_{k,n}$ va se voir attribuer du temps processeur.

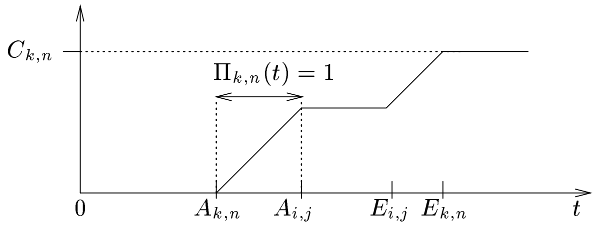

FIG. 2.1 – Evolution de $\int_0^t \Pi_{k,n}(x)dx$ au cours du temps.

On remarque la figure 2.1 que $\tau_{k,n}$ n'a la ressource à aucun instant entre $A_{i,j}$ et $E_{i,j}$ qui sont les dates de d'arrivée et de fin d'exécution d'une instance $\tau_{i,j}$. Cela signifie que $\tau_{i,j}$ est plus prioritaire que $\tau_{k,n}$ ce qui sera noté $\tau_{i,j} \succ \tau_{k,n}$. À l'arrivée de $\tau_{i,j}$ à la date $A_{i,j}$, $\tau_{k,n}$ est préemptée par $\tau_{i,j}$ et tant que cette instance aura du travail à exécuter, c'est à dire jusqu'en $E_{i,j}$, $\tau_{k,n}$ est en attente de la ressource. On voit sur cet exemple que l'instant de fin d'exécution d'une instance $\tau_{k,n}$, pour une politique donnée, dépend à la fois de sa quantité de travail mais également de la quantité de travail de toutes les instances plus prioritaires qui peuvent "*interférer*" avec $\tau_{k,n}$.

Par définition, la date de fin d'exécution $E_{k,n}$ est le premier instant tel que la somme du travail qui a été consacré à $\tau_{k,n}$ est égal à son temps d'exécution :

$$E_{k,n} \stackrel{\text{def}}{=} \min\{t \mid \int_0^t \Pi_{k,n}(x)dx = C_{k,n}\}$$

On note $B_{k,n}$ le premier instant auquel $\tau_{k,n}$ gagne le processeur, c'est le premier instant tel que $\Pi_{k,n}(t)$ soit positif :

$$B_{k,n} \stackrel{\text{def}}{=} \min\{t \geqslant A_{k,n} \mid \Pi_{k,n}(t) > 0\}$$

Il est toujours vrai que sur un intervalle $[t_1, t_2]$, une instance ne peut utiliser le processeur plus que son temps d'exécution :

$$\int_{t_1}^{t_2} \Pi_{k,n}(t)dt \leqslant C_{k,n} \qquad \forall\, t_1, t_2 \in \mathbb{R},$$

et qu'à chaque instant il ne peut y avoir plus d'une instance sur le processeur :

$$\forall\, t \in \mathbb{R}_+ \quad \sum_{k=1}^{m} \sum_{n \in \mathbb{N}} \Pi_{k,n}(t) \leqslant 1$$

De la même façon que $\Pi_{k,n}(t)$ mesure la quantité de processeur attribuée au niveau de l'instance, nous pouvons définir

$$\Pi_k(t) = \sum_{n \in \mathbb{N}} \Pi_{k,n}(t)$$

qui mesure la quantité de travail de toutes les instances de τ_k et

$$\Pi_{\mathcal{I}}(t) = \sum_{\tau_i \in \mathcal{I}} \Pi_i(t)$$

la quantité de travail des instances d'un sous-ensemble de tâches $\mathcal{I}$. Les fonctions $\Pi_{k,n}(t)$ comme $\Pi_k(t)$ et $\Pi_{\mathcal{I}}(t)$ sont définies comme étant continues à droite $\lim_{t \to t_2, t > t_2} \Pi_{k,n}(t) = \Pi_{k,n}(t_2)$.

La séquence des valeurs de $\Pi_{k,n}(t)$ pour toute instance de tâches et pour tout instant t, notée $\{\Pi_{k,n}(t)\}_{k \in 1..m,\, n \in \mathbb{N},\, t \in \mathbb{R}}$, définit de façon compléte le résultat de l'ordonnancement du système, c'est à dire la trajectoire du système.

Définition 2 *Trajectoire du système.*

Etant donnée un processus de tâches ω, une trajectoire est le résultat de l'ordonnancement sous une certaine politique $\mathcal{A}$ (on parle "d'ordonnancement réalisé"). Une trajectoire est définie de façon compléte par la séquence des $\{\Pi_{k,n}(t)\}_{k \in 1..m,\, n \in \mathbb{N}, t \in \mathbb{R}}$.

Sous certaines politiques, il est possible qu'un même processus de tâches rejoué plusieurs fois donne lieu à des trajectoires différentes du système. C'est le cas lorsque le choix de la tâche à exécuter se fait de façon aléatoire par tirage au sort (*lottery scheduling*, consulter par exemple [18]). Dans une optique temps réel, où la prévisibilité du système est primordiale, nous ne nous intéresserons qu'à des politiques "*déterministes*".

Définition 3 *Politique déterministe.*

Soit ω un processus de tâches du système, une politique $\mathcal{A}$ est dite déterministe si il n'existe qu'une et une seule trajectoire du système sous la politique $\mathcal{A}$ avec le processus de tâches ω.

2.3 Fonction d'arrivée de travail

Pour mesurer la quantité de travail soumise à la ressource dans tout intervalle de temps, nous introduisons la *fonction d'arrivée de travail* (FAT), notée $S_{k,n}(t_1, t_2)$. Cette fonction mesure la quantité de travail demandée par une instance $\tau_{k,n}$ dans tout intervalle de temps $[t_1, t_2[$:

$$S_{k,n}(t_1, t_2) \stackrel{\text{def}}{=} C_{k,n} \cdot \mathbb{1}_{[t_1 \leqslant A_{k,n} < t_2]} \tag{2.2}$$

où $\mathbb{1}_{[a]}$ est la fonction indicatrice qui vaut 1 si le prédicat a est vérifié et 0 sinon. Remarquons que $S_{k,n}(t_1, t_2)$ est définie de façon à ne pas intégrer le travail arrivant exactement en t_2 ; $S_{k,n}(t_1, t_2)$ est continue à gauche ($\lim_{t \to t_2, t < t_2} S_k(t) = S_k(t_2)$). La figure 2.2 représente l'évolution de $S_{k,n}(0, t)$ dans le temps.

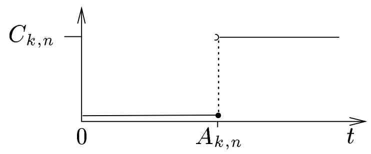

FIG. 2.2 – Evolution de $S_{k,n}(0, t)$ au cours du temps avec en particulier $S_{k,n}(0, A_{k,n}) = 0$.

Les FATs nous permettent de mesurer la quantité de travail plus prioritaire qu'une instance soumise sur un intervalle de temps et joue donc un rôle primordiale dans le calcul des temps de réponse. Intuitivement, le choix de la continuité à gauche est motivée par le fait qu'une instance $\tau_{i,j} \succ \tau_{k,n}$ ne retardera pas $\tau_{k,n}$ si sa date d'arrivée est exactement l'instant de fin d'exécution de $\tau_{k,n}$}(si $A_{i,j} = E_{k,n}$).

De la même façon que cela a été fait pour une instance, nous définissons une fonction d'arrivée de travail pour toutes les instances d'une même tâche τ_k :

$$S_k(t_1, t_2) \stackrel{\text{def}}{=} \sum_{n \in \mathbb{N}} S_{k,n}(t_1, t_2) = \sum_{n \in \mathbb{N}} C_{k,n} \cdot \mathbb{1}_{[t_1 \leqslant A_{k,n} < t_2]}. \tag{2.3}$$

ainsi qu'une FAT pour un sous-ensemble de tâches $\mathcal{I}$:

$$S_{\mathcal{I}}(t_1, t_2) \stackrel{\text{def}}{=} \sum_{\tau_i \in \mathcal{I}} S_i(t_1, t_2). \tag{2.4}$$

Les équations 2.3 et 2.4 ne font aucune hypothèse sur le modèle de tâches. Pour calculer les temps de réponse maximaux, il nous faudra par la suite caractériser plus précisement $S_k(t_1, t_2)$ et $S_{\mathcal{I}}(t_1, t_2)$ en fonction du modèle de tâche. Nous le ferons en particulier pour des tâches strictement périodiques à temps d'exécution constant.

2.4 Fonction de charge

La *fonction de charge*, notée $W_{k,n}(t)$, est la fonction qui représente la quantité de travail d'une instance $\tau_{k,n}$ arrivée avant l'instant t et qui reste encore à être effectuée à cet instant :

$$W_{k,n}(t) \stackrel{\text{def}}{=} S_{k,n}(0,t) - \int_0^t \Pi_{k,n}(u)du \tag{2.5}$$

Nous définissons également une fonction de charge au niveau de la tâche $W_k(t) \stackrel{\text{def}}{=} \sum_{n\in\mathbb{N}} W_{k,n}(t)$ et pour un sous-ensemble de tâches $W_{\mathcal{I}}(t) \stackrel{\text{def}}{=} \sum_{\tau_i\in\mathcal{I}} W_i(t)$. Comme $S_{k,n}(t_1,t_2)$ n'intégre pas la charge arrivant en t_2, $W_{k,n}(t)$ (et donc $W_k(t)$ et $W_{1..k}(t)$) est également continue à gauche ; ainsi par exemple $W_{k,n}(A_{k,n}) = 0$.

Une instance termine son exécution dès que sa quantité de travail en attente d'exécution est nulle. L'instant de fin d'exécution $E_{k,n}$ de $\tau_{k,n}$ peut donc être défini comme :

$$E_{k,n} \stackrel{\text{def}}{=} \min\{t > A_{k,n} \mid W_{k,n}(t) = 0\} \tag{2.6}$$

Comme cela est le cas sur les figures 2.1 et 2.3, il est possible qu'une instance de tâche soit retardée par des tâches plus prioritaires et son *temps de réponse* qui est par définition

$$R_{k,n} \stackrel{\text{def}}{=} E_{k,n} - A_{k,n} \tag{2.7}$$

est alors plus grand que son temps d'exécution $C_{k,n}$.

La figure 2.3 représente les évolutions des fonctions de charge pour deux instances successives d'une même tâche τ_k et illustre les différentes grandeurs introduites jusqu'à présent.

Exercice 1 $\star\star$

Quelles sont les valeurs des grandeurs suivantes :

1. $S_{k,n}(0,A_{k,n})$, $S_{k,n}(0,A^+_{k,n})$, $S_{k,n}(0,E_{k,n}+1)$ *et* $S_{k,n}(0,B_{k,n})$,
2. $W_{k,n}(A_{k,n})$, $W_{k,n}(A^+_{k,n})$, $W_{k,n}(B_{k,n})$ *et* $W_{k,n}(E_{k,n})$,
3. $\Pi_{k,n}(B_{k,n})$, $\Pi_{k,n}(A_{k,n})$, $\Pi_{k,n}(E_{k,n})$ *et* $\int_0^{E_{k,n}} \Pi_{k,n}(u)du$.

Justifier vos résultats si nécessaire.

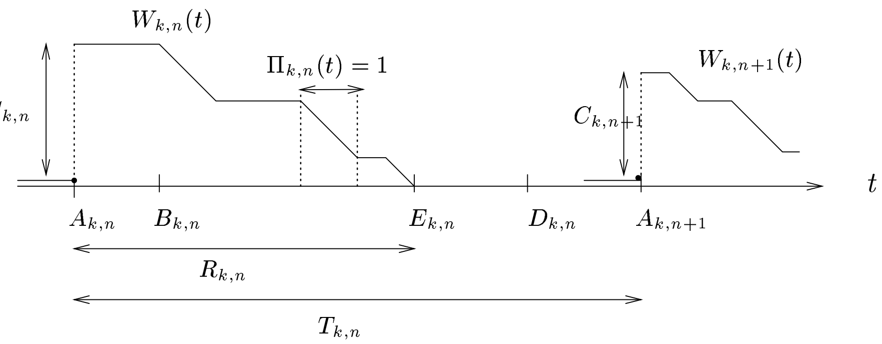

FIG. 2.3 – Dates et fonctions associées à une instance de τ_k.

2.5 Conservation de la charge entre politiques non-oisives

Une politique d'ordonnancement telle que le processeur est toujours occupé lorsqu'il y a du travail en attente est une *politique non-oisive*.

Définition 4 *Politique non-oisive*

Une politique est non-oisive si toute trajectoire sous cette politique vérifie

$$W_{1..m}(t^+) > 0 \quad \Rightarrow \quad \Pi_{1..m}(t) = 1.$$

Dans la suite de ce cours, nous restreindrons notre analyse à cette famille de politiques ; les politiques oisives n'ayant que très peu d'intérêt dans le cadre de l'ordonnancement préemptif de tâches sous contraintes d'échéances qui constitue l'objet principal de ce cours.

Théorème 1 *Conservation de charge entre politique non-oisive*

Soient $\mathcal{A}$ et $\mathcal{B}$ deux politiques d'ordonnancement non-oisives et ω un processus de tâches, la charge de travail en attente est à tout instant identique que l'ordonnancement de ω soit effectué avec $\mathcal{A}$ ou $\mathcal{B}$.

Preuve:

Toutes les grandeurs se rapportant à une politique $\mathcal{A}$ seront indicées par $\mathcal{A}$, par exemple $\Pi_k^{\mathcal{A}}(t)$. D'après la définition de la fonction de charge et comme le processus de tâches ω est identique pour $\mathcal{A}$ et $\mathcal{B}$, nous avons à tout instant t :

$$
\begin{aligned}
W_{\mathcal{T}}^{\mathcal{A}}(t) &= S_{\mathcal{T}}(0,t) - \int_0^t \Pi_{\mathcal{T}}^{\mathcal{A}}(x)dx \\
W_{\mathcal{T}}^{\mathcal{B}}(t) &= S_{\mathcal{T}}(0,t) - \int_0^t \Pi_{\mathcal{T}}^{\mathcal{B}}(x)dx
\end{aligned}
$$

Si $\Pi_{\mathcal{T}}^{\mathcal{A}}(t) = \Pi_{\mathcal{T}}^{\mathcal{B}}(t)$ pour tout t alors on a forcément $W_{\mathcal{T}}^{\mathcal{A}}(t) = W_{\mathcal{T}}^{\mathcal{B}}(t)$. Faisons l'hypothése que les fonctions d'utilisation de la ressource ne sont pas identiques sous les deux politiques. Par la continuité à droite de $\Pi_{\mathcal{T}}^{\mathcal{A}}(t)$ et $\Pi_{\mathcal{T}}^{\mathcal{B}}(t)$, alors il existe un temps $t_0 > 0$ t.q.

$$
\begin{aligned}
\Pi_{\mathcal{T}}^{\mathcal{A}}(t) &= \Pi_{\mathcal{T}}^{\mathcal{B}}(t), \quad \forall t\, t.q.\, 0 \leq t < t_0 \\
\Pi_{\mathcal{T}}^{\mathcal{A}}(t_0) &\neq \Pi_{\mathcal{T}}^{\mathcal{B}}(t_0)
\end{aligned}
$$

Si l'on a $\Pi_{\mathcal{T}}^{\mathcal{A}}(t_0) = 0$ et $\Pi_{\mathcal{T}}^{\mathcal{B}}(t_0) = 1$, comme les dates d'activation et les temps d'exécutions de toutes les instances de tâches sont identiques pour les deux politiques (on raisonne sur le même processus de tâches), et en utilisant le fait que les fonctions d'utilisation sont égales jusqu'à l'instant t_0[1]

$$W_{\mathcal{T}}^{\mathcal{B}}(t_0) = W_{\mathcal{T}}^{\mathcal{A}}(t_0).$$

En utilisant le fait que $\mathcal{A}$ est non-oisive, $\Pi_{\mathcal{T}}^{\mathcal{A}}(t_0) = 0$ signifie que $W_{\mathcal{T}}^{\mathcal{A}}(t_0) = 0$. Pour $\mathcal{B}$, $\Pi_{\mathcal{T}}^{\mathcal{B}}(t_0) = 1$ et $W_{\mathcal{T}}^{\mathcal{B}}(t_0) = 0$ est possible si il y a une arrivée exactement en t_0. Comme les instants des arrivées coïncident toujours sous $\mathcal{A}$ et $\mathcal{B}$, cela implique qu'il y ait également une arrivée pour $\mathcal{A}$ à l'instant t_0, ce qui contredit le fait que $\Pi_{\mathcal{T}}^{\mathcal{A}}(t_0) = 0$. Le même argument tient pour prouver que $\Pi_{\mathcal{T}}^{\mathcal{A}}(t_0) = 1$ et $\Pi_{\mathcal{T}}^{\mathcal{B}}(t_0) = 0$ est impossible.

■

Si la charge de travail globale en attente est identique sous toute politique non-oisive, les instances en attente, elles, diffèrent, et les politiques ne se valent généralement pas vis-à-vis du respect des échéances ou d'autres critères tels que la minimisation du nombre d'instances en attente.

[1]La valeur des fonctions $\Pi(\cdot)$ en t_0 n'influe pas sur la valeur de l'intégrale, il faudrait une infinité non-dénombrable de points pour cela.

2.6 Faisabilité du système

Le problème essentiel de l'ordonnancement temps réel est d'assurer que toutes les instances de tâches respecteront, quoi qu'il puisse arriver, leurs échéances, c'est à dire concevoir des systèmes composés de *tâches faisables*. De tels systèmes sont dits faisable ou *ordonnançable*.

Définition 5 *Faisabilité d'un ensemble de tâches.*

Un ensemble de tâches est faisable sous une certaine politique si toutes les instances de toutes les tâches finissent leur exécution avant leur échéance pour tout processus de tâches :

$$E_{k,n}(\omega) \leqslant D_{k,n}(\omega) \qquad \forall \omega \in \Omega, \forall k = 1..m, \forall n \in \mathbb{N}.$$

Le plus souvent les échéances relatives de toutes les instances d'une même tâche sont égales, $\overline{D}_{k,n} = \overline{D}_k \ \forall n \in \mathbb{N}$, et la tâche τ_k sera faisable si son temps de réponse maximal $\hat{R}_k$ sur toute trajectoire du système est inférieur à l'échéance relative :

$$\hat{R}_k = \max_{\omega \in \Omega, n \in \mathbb{N}} R_{k,n}(\omega) \leqslant \overline{D}_k$$

Comme nous l'étudierons par la suite, il n'est pas toujours nécessaire de calculer explicitement les temps de réponse pour juger de la faisabilité ; un test d'ordonnançabilité généralement basé sur la charge peut suffir. Néanmoins, en cas d'échec, la connaissance des temps de réponse maximaux pour chacune des tâches nous permet de localiser le problème et donc d'y remédier plus facilement.

D'une façon générale il n'est pas possible d'étudier l'ordonnancement de tous les processus de tâches possibles du système et pour évaluer les temps de réponse maximaux, il va falloir construire le ou les processus de tâches les plus défavorables qui puissent être envisagés et à prouver que sur les trajectoires résultantes les pire temps de réponse sont toujours inférieurs aux échéances relatives.

2.7 Définition des politiques d'ordonnancement

Il est primordial lorque l'on s'intéresse à l'ordonnancement de définir sans ambiguïté les politiques que l'on veut étudier ou implémenter. Il est naturellement possible d'utiliser le langage naturelle, par exemple nous définirions la politique *Fixed Priority Preemptive* (FPP) de la façon suivante :

> "Sous la politique d'ordonnancement préemptif à priorités fixes ou FPP, les priorités des instances de tâches sont fixes au cours du temps et à chaque instant le processeur est alloué à la tâche de plus forte priorité susceptible de s'exécuter, les tâches de priorité identique étant départagées selon le principe FIFO (*First In - First Out*)."

Pour FPP une telle définition est envisageable mais pour des politiques plus complexes, le langage naturel est souvent porteur de trop d'ambiguïtés. D'autre part, nous verrons qu'une définition plus formelle nous permettra de constituer des classes de politiques pour lesquelles des résultats d'ordonnancement génériques peuvent être trouvés. En particulier, dans le paragraphe ? nous donnerons des formules sur les temps de réponse valables pour la classe "*des politiques indépendantes du temps*".

2.7.1 Définition à l'aide de la fonction d'ordonnancement

Une première approche plus formelle est d'utiliser la fonction d'ordonnancement pour caractériser toutes les trajectoires sous une certaine politique. Par exemple sous FPP, avec la convention "*plus petit l'indice de la tâche, plus forte la priorité*" ($i < j \Rightarrow \tau_i \succ \tau_j$) qui sera toujours adoptée pour cette politique, nous pouvons écrire :

$$\Pi_{k,n}(t) = 1 \quad \Rightarrow \quad \begin{cases} \forall n, \forall i < k & W_{i,n}(t) = 0 \\ \forall j < n & W_{k,j}(t) = 0. \end{cases} \tag{2.8}$$

Ce qui signifie que toute instance $\tau_{k,n}$ ne pourra obtenir la ressource que si toutes les instances plus prioritaires n'ont pas de travail en attente d'exécution. L'ensemble des instances plus prioritaires que $\tau_{k,n}$ étant constitué des instances de tâches plus prioritaires mais aussi des instances de τ_k d'indice inférieur à n pour respecter la règle FIFO pour les instances d'une même tâche.

Exercice 2 $\star\star$

Soit une trajectoire du système vérifiant :

$$\Pi_{k,n}(t) = 1 \quad \Rightarrow \quad \forall i, j \vdash D_{i,j} < D_{k,n},\ W_{i,j}(t) = 0, \tag{2.9}$$

1. *Quelle est la politique d'ordonnancement sous-jacente ?*
2. *Dans quels cas la règle est-elle ambigue ?*
3. *Compléter la règle de façon à lever les ambiguïtés.*

Exercice 3 $\star$
*Définir à l'aide de la fonction d'ordonnancement les trajectoires sous les politiques FIFO (*First In - First Out*) et LIFO (*Last In - First Out*).*

Ces définitions qui utilisent la fonction d'ordonnancement $\Pi_{k,n}(t)$ ne s'intéressent pas à la politique d'ordonnancement à proprement parler mais aux caractéristiques communes à toutes les trajectoires sous cette politique. Il est possible de définir une politique plus directement que par ses effets à l'aide de la "*fonction de priorités*".

2.7.2 Définition à l'aide de la fonctions de priorités

Une seconde approche pour définir les politiques d'ordonnancement est d'associer à toute instance de tâche une fonction de priorité qui indique à chaque instant la priorité courante de l'instance. En ligne, l'ordonnanceur sélectionnera toujours *l'instance la plus prioritaire en premier*. La fonction de priorité $\Gamma_{k,n}(t)$, introduite dans [11], donne la priorité d'une instance $\tau_{k,n}$ au cours du temps. Pour être en mesure de décider de l'attribution de la ressource, il faut pouvoir comparer la priorité de toutes les instances du système entre elles :

$$\Gamma_{k,n}(t) \in \mathcal{P},$$

où $\mathcal{P}$ est un ensemble d'éléments muni d'un ordre total. Nous avons choisi d'exprimer les priorités comme des vecteurs de nombres $\mathcal{P} = \{(p_1, \ldots, p_n) \in \mathbb{R}^n \mid n \in \mathbb{N}\}$ munis d'un ordre lexicographique ; les coordonnées sont vues comme les lettres d'un mot et l'ordre entre deux priorités se décident à la première coordonnées différentes depuis la gauche. Deux vecteurs sont égaux ssi :

$$(p_1, \ldots, p_n) = (p'_1, \ldots, p'_{n'}) \quad \Longleftrightarrow \quad n = n' \text{ et } p_j = p'_j,\ j = 1, \ldots, n.$$

On note $p \preccurlyeq p'$ pour signifier que la priorité p est plus petite ou égale que p' alors que $p \prec p'$ est la relation strictement plus petit qui est définie par :

$$(p_1,\dots,p_n) \prec (p'_1,\dots,p'_{n'}) \iff \left|\begin{array}{ll} \text{(a)} & \exists\, i \leqslant \min(n,n') \vdash\ p_j = p'_j,\ j<i \text{ et } p_i > p'_i \\ \text{ou} & \\ \text{(b)} & n' < n \text{ et } p_j = p'_j,\ j = 1,\dots,n'. \end{array}\right.$$

Remarquons dans cette définition que plus petite la valeur numérique d'une coordonnée, plus grande est la priorité. Par exemple, on a $(1,2,3,4) \prec (1,2,2,4)$, $(1,2,3) \prec (1,2)$ et $(1,2,3,4,5) \prec (1,2,3,4,4)$. L'ordre sur les priorités étant défini, nous pouvons décider de la régle d'allocation de la ressource et nous adopterons logiquement la règle de l'instance la plus prioritaire en premier.

Définition 6 *Règle de l'instance la plus prioritaire en premier.*

La règle de l'instance la plus prioritaire en premier est respectée si pour tout $\tau_{k',n'} \neq \tau_{k,n}$ et $\forall\, \omega \in \Omega$

$$\Pi_{k,n}(t) = 1 \textit{ et } W_{k',n'}(t^+) > 0 \quad \Rightarrow \quad \Gamma_{k',n'}(t) \prec \Gamma_{k,n}(t)$$

Une définition possible de la fonction de priorités de la politique FPP est

$$\Gamma_{k,n}(t) = (k,n)$$

où la première composante est l'indice (et aussi la priorité cf. 2.7.1) de la tâche, la seconde composante étant le numéro de l'instance. Il faut remarquer qu'il existe généralement plusieurs fonctions de priorités pour définir une même politique, ainsi $\Gamma_{k,n}(t) = (k+8, n+1)$ est également une définition parfaitement valable de FPP.

Définition 7 *Fonction de priorités équivalentes.*

Deux fonctions de priorités Γ_1 et Γ_2 qui prennent respectivement leurs valeurs dans les ensembles $\mathcal{P}_1$ et $\mathcal{P}_2$ (tous deux munis d'un ordre total $\overset{1}{\succ}$ et $\overset{2}{\succ}$) sont équivalentes si pour tout $\tau_{k,n} \neq \tau_{i,j}$ l'ordre induit par les deux fonctions de priorités est identique :

$$\Gamma_{k,n}(t) \overset{1}{\succ} \Gamma_{i,j}(t) \Leftrightarrow \Gamma'_{k,n}(t) \overset{2}{\succ} \Gamma'_{i,j}(t).$$

Exercice 4 $\star\star$

Trouver des fonctions de priorités pour les politiques FIFO, LIFO, Shortest Processing Time First *(SPTF), et pour la politique dont toute trajectoire satisfait 2.9.*

Exercice 5 $\star\star$

Imaginer un système d'exploitation implémentant le concept de fonction de priorité, quels intérêts ?

L'intérêt premier des fonctions de priorité est de permettre de définir formellement les politiques d'ordonnancement de telle façon à lever toute ambiguïté sur leurs principes de fonctionnement. Néanmoins, toute fonction de priorité ne définit pas de façon valable une politique d'ordonnancement, en particulier l'allocation des priorités doit être nécessairement *décidable*.

Définition 8 *Allocation de priorités décidable.*

Une allocation de priorités est décidable si à tout instant deux instances actives[2] *ont des priorités différentes :* $\forall t \in \mathbb{R}_+, \forall \tau_{k,n} \neq \tau_{k',n'}$

$$W_{k,n}(t^+) > 0 \text{ et } W_{k',n'}(t^+) > 0 \quad \Rightarrow \quad \Gamma_{k,n}(t) \neq \Gamma_{k',n'}(t).$$

Considérons la politique définit par la fonction de priorité

$$\Gamma_{k,n}(t) = \int_0^t \Pi_{k,n}(x)dx. \tag{2.10}$$

Cette politique vise à partager le processeur de façon "équitable" entre instances actives. Une instance occupe la ressource à un instant t si le temps processeur qui lui a déjà été alloué jusqu'à cet instant est le plus petit parmi toutes les instances actives. En s'exécutant, la valeur de $\int_0^t \Pi_{k,n}(x)dx$ augmente et par la même la priorité de l'instance diminue. Imaginons deux instances $\tau_{k,n}$ et $\tau_{i,j}$ activées au même instant $t = A_{k,n} = A_{i,j}$, elles auront strictement la même priorité et la fonction de priorité définie par 2.10 n'est donc pas décidable. Il est possible de rendre l'allocation décidable en rajoutant comme coordonnées au vecteur de priorité l'indice de la tâche et l'indice de l'instance :

$$\Gamma_{k,n}(t) = \Big(\int_0^t \Pi_{k,n}(x)dx, k, n\Big). \tag{2.11}$$

Un problème subsiste néanmoins avec cette allocation. Reprenons l'exemple de nos deux instances activées à $t = A_{k,n} = A_{i,j}$, imaginons $k < i$, alors $\tau_{k,n}$ est exécutée et $\Pi_{k,n}(t) = 1$ alors que $\Pi_{i,j}(t) = 0$. Mais pour $\epsilon > 0$ arbitrairement petit on a :

$$\Gamma_{k,n}(t+\epsilon) = (\epsilon, k, n) \prec (0, i, j) = \Gamma_{i,j}(t+\epsilon),$$

et donc immédiatement après t, $\tau_{i,j}$ devient plus prioritaire que $\tau_{k,n}$. En pratique, une telle politique est impossible car aucune ressource

[2]Une instance $\tau_{k,n}$ est *active* à l'instant t si $t \geq A_{k,n}$ et $W_{k,n}(t^+) > 0$.

ne peut changer infiniment souvent de contexte dans un intervalle de temps fini. Il faut donc respecter une contrainte supplémentaire qui est que dans tout intervalle de temps de taille finie, l'ordre des priorités ne change pas infiniment souvent. Une solution à ce problème est d'imposer qu'une tâche devenue prioritaire le reste pendant un intervalle de temps minimum. La définition 2.11devient alors :

$$\Gamma_{k,n}(t) = \left(\left\lfloor \frac{\int_0^t \Pi_{k,n}(x)dx}{\Psi_{k,n}} \right\rfloor, k, n \right)$$

où le paramètre $\Psi_{k,n}$ et l'opérateur plancher font que la première coordonnée de la fonction de priorité reste constante un certain temps après le début d'exécution.

Dans la suite du cours, nous verrons qu'il est possible de distinguer deux classes de fonctions de priorités, celles dont la définition est *indépendante* du temps et celles qui dépendent du temps.

Définition 9 *Fonction de priorité indépendante du temps.*
Une fonction de priorités est indépendante du temps si :

$$\Gamma_{k,n}(t) = \Gamma_{k,n}(0) \quad \forall t \in \mathbb{R}_+ .$$

Globalement les politiques définis par des fonctions de priorités indépendantes du temps sont plus faciles à analyser et nous établirons des formules génériques donnant en particulier les temps de réponses pour cette classe de politiques qui comprend FPP, FIFO, LIFO, ….

Exercice 6 $\star\star$

1. *Remarquer que la fonction de priorité de EDF est indépendante du temps. Comment expliquer alors que, classiquement, dans la littérature la politique EDF soit classée parmi les politiques à priorités dynamiques ?*
2. *La laxité est la différence entre le temps restant jusqu'à l'échéance et la quantité de travail encore à effectuer. Définir la politique* Least Laxity First *(LLF) à l'aide d'une fonction de priorité.*

Exercice 7 $\star\star$

Soit la politique d'ordonnancement définie par la fonction de priorité :

$$\Gamma_{k,\,n}(t) = (W_{k,n}(t^+),\, k,\, n)$$

où $W_{k,n}(t)$ est la fonction qui mesure dans le temps la charge de travail encore à effectuer pour l'instance $\tau_{k,n}$.

1. *Expliquer brièvement cette politique.*
2. *Représenter graphiquement l'ordonnancement sous la trajectoire définie par* $(A_{1,1} = 0,\ C_{1,1} = 4)$,$(A_{1,2} = 3,\ C_{1,2} = 3)$,$(A_{2,1} = 1,\ C_{2,1} = 2)$, $(A_{3,1} = 2,\ C_{3,1} = 3)$.
3. *Quelle hypothèse doit être satisfaite pour que cette politique soit implémentable ?*
4. *Soit* n*, le nombre d'instances de tâches activées dans un intervalle de temps, quel est le nombre maximum de changements de contexte (changement de la tâche active sur le processeur) ?*

2.8 Optimalité d'une politique

Dans ce cours, nous étudierons des résultats importants de l'ordonnancement et, en particulier, des résultats concernant *l'optimalité* de politiques d'ordonnancement dans certaines classes de problèmes. Le seul critère d'optimalité que nous retiendrons, car de loin le plus important dans un contexte temps réel, est celui de l'ordonnançabilité d'un ensemble d'activités récurrentes. Les classes de problèmes peuvent être distinguées en fonction de caractéristiques des tâches ou du système :
- préemptivité ou non-préemptivité de l'ordonnancement,
- modèle d'activation des tâches (ex : tâches strictement périodiques),
- caractéristiques des échéances (ex : échéances inférieures à la période),
- politique oisive ou non-oisive,
- ...

Définition 10 *Optimalité vis-à-vis de l'ordonnançabilité.*

Une politique d'ordonnancement est optimale vis-à-vis de l'ordonnançabilité si elle conduit à un ordonnancement faisable pour tout processus de tâches faisables.

La définition ci-dessus dit que s'il existe une politique telle qu'un ensemble de tâches soit faisable alors cet ensemble de tâches sera faisable sous la politique optimale. Nous verrons par exemple que l'allocation de priorités *Rate Monotonic* (plus petite la période, plus grande la priorité) est optimale pour la politique FPP dans le contexte de l'ordonnancement préemptif de tâches périodiques à échéances sur requête.

Il existe naturellement d'autres critères d'optimalité que l'ordonnançabilité du système qui puissent présenter de l'interêt pour des systèmes temps réel. Par exemple, dans des systèmes embarqués disposant de peu de mémoire vive comme les cartes à puce, le nombre maximum d'instances de tâches qui peuvent avoir été préemptées, et donc le nombre de contextes (registres du CPU, pointeur de code et de pile. . .) à sauvegarder, est important. Pour des équipements électroniques dont l'alimentation est assurée par des batteries (portables, assistants intelligents etc..), l'autonomie est un problème crucial et il est alors souhaitable d'implanter une politique qui fasse fonctionner le CPU à la fréquence la plus faible[3] compatible avec le respect des échéances.

[3] L'énergie consommée par un CPU varie en le carré de la fréquence de fonctionnement, le temps de calcul lui étant globalement linéaire en la fréquence.

Bibliographie

[1] F. Baccelli and P. Brémaud. *Elements of Queuing Theory*. Springer-Verlag, 1994. volume 26 of Applications of Mathematics.

[2] M. Bayart and F. Simonot-Lion. Impact de l'émergence des réseaux de terrain et de l'instrumentation intelligente dans la conception des architectures des systèmes d'automatisation de processus. Contrat MESR - 92-p-239, Centre de Recherche en Informatique de Nancy, Vandoeuvre-lès-Nancy, Février 1995.

[3] C. Chaouiya, S. Lefebvre-Barbaroux, and A. Jean-Marie. Real-time scheduling of periodic tasks. In P. Chrétienne, E.G. Coffman, J.K. Lenstra, and Z. Liu, editors, *Scheduling Theory and its Applications*, chapter 8, pages 167–191. John Wiley and Sons Ltd, 1995.

[4] M. DiNatale and J.A. Stankovic. Applicability of simulated annealing methods to real-time scheduling and jitter control. In *Proceedings of the 16th IEEE Real-Time Systems Symposium*, Décembre 1995.

[5] B. Gaujal, N. Navet, and J. Migge. Dual-priority versus background scheduling : a path-wise comparison. *Accepté pour publication dans Real-Time Systems*, 2002.

[6] ISO. *Road Vehicles - Low Speed serial data communication - Part 2 : Low Speed Controller Area Network*. 1994. International Standard Organization, ISO 11519-2.

[7] J.P. Lehoczky. Fixed Priority Scheduling Of Periodic Task Sets with Arbitrary Deadlines. In *Proceedings IEEE Real-Time Systems*, pages 201–209, Décembre 1990.

[8] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in hard real-time environment. *Journal of the ACM*, 20(1) :40–61, February 73.

[9] C.D. Locke. Software architecture for hard real-time applications : Cyclic executives vs fixed priority executives. *Journal of Real-Time Systems*, 4(1), 1992.

[10] J. Migge, Alain Jean-Marie, and N. Navet. Timing analysis of compound scheduling policies : Application to Posix1003.1b. *Accepted for publication in Journal of Scheduling*, to appear in 2002.

[11] J.M. Migge. *Scheduling of recurrent tasks on one processor : A trajectory based Model.* PhD thesis, Université Nice Sophia-Antipolis, 1999. disponible à l'adresse `http://www.migge.net/jorn/thesis/`.

[12] M.L. Minsky. Matters, minds and models. In *Proc. International Federation of Information Processing Congress*, pages 45–49, 1965.

[13] N Navet. *Évaluation de performances temporelles et optimisation de l'ordonnancement de tâches et de messages.* PhD thesis, Institut National Polytechnique de Lorraine (INPL), 1999. disponible à l'adresse `http://www.loria.fr/~nnavet`.

[14] N. Navet and J. Migge. Fine tuning the scheduling of tasks through a genetic algorithm : Application to Posix1003.1b compliant systems. *à paraître dans IEE Proceedings - Software*, 2002.

[15] J. Popper. *La dynamique des systèmes, principes et applications.* Éditions d'Organisation, 1973.

[16] J.A. Stankovic. Misconceptions about real-time computing. *IEEE Computer*, 21(10) :10–19, Octobre 1988.

[17] Luis Vega. *Modèles de coopération et de communication entre processus temps réel répartis : expression de contraintes de temps pour la vérification de propriétés temporelles dans la communication.* Thèse d'université, INPL, 1996.

[18] C Waldspurger and W Weihl. Lottery scheduling : Flexible proportional-share resource management. In *Operating Systems Design and Implementation*, pages 1–11, 1994.

Index