Chapitre 8

Introduction aux Systèmes Temps Réel

8.1 Introduction

La gestion du temps est l'un des problèmes majeurs des systèmes d'exploitation. La raison est simple: les systèmes d'exploitation modernes sont tous multitâche, or il utilisent du matériel basé sur des processeurs qui ne le sont pas, ce qui oblige le système à partager le temps du processeur entre les différentes tâches. Cette notion de partage implique une gestion du passage d'une tâche à l'autre qui effectuée par un ensemble d'algorithmes appelé ordonnanceur(*scheduler* en anglais).

Un système d'exploitation conventionnel comme Unix, Linux ou Windows utilise la notion de temps partagé, par opposition au temps réel. Dans ce type de système, le but de l'ordonnanceur est de donner à l'utilisateur une impression de confort toit en s'assurant que toutes les tâches demandées sont finalement exécutées. Ce type d'approche entraîne une grande complexité dans la structure même de l'ordonnanceur qui doit tenir compte de notions comme la régulation de la charge du système ou la date depuis laquelle une tâche donnée est en cours d'exécution. De ce fair, on peut noter plusieurs limitations par rapport à la gestion du temps.

Tout d'abord, la notion de priorité entre les tâches est peu prise en compte, car l'ordonnanceur a pour but premier le partage équitable du temps entre les différentes tâches du système (on parle de quantum de temps ou tick).

Ensuite, les différentes tâches doivent accéder à des ressources dites partagées, ce qui entraîne des incertitudes temporelles. Si une des tâches effectue une écriture sur le disque dur, celui-ci n'est plus disponible aux autres tâches à u instant donné et le délai de disponibilité du périphérique n'est donc pas prévisible.

En outre, la gestion des entrées/sorties peut générer des temps morts car une tâche peut être bloquée en attente d'accès à un élément d'entrée/sortie. La gestion des interruptions reçues par une tâche n'est pas optimisée. Le temps de latence - soit le temps écoulé

entre la réception de l'interruption et son traitement - n'est pas garanti par le système.

Enfin, l'utilisation du mécanisme de *mémoire virtuelle* peut entraîner des fluctuations importante dans les temps d'exécution des tâches.

8.2 Limitation des systèmes d'exploitation généralistes

Lorsqu'une application temps réel est soumise à de fortes contraintes de temps, l'utilisation d'un système d'exploitation généraliste est inadaptée, car les objectifs des systèmes d'exploitation conventionnels sont les suivants:

- garantir l'indépendance des processus et les protéger les uns vis-à-vis des autres notamment grâce à la MMU (Memory Management Unit), ce qui implique une mise en œuvre lourde des communications entre les processus;
- augmenter la vitesse moyenne de traitement à l'aide d'optimisations locales: utilisation d'un cache disque permettant un accès asynchrone, mais non prédictible, aux mémoires de masse; utilisation des optimisations des processeurs tels les pipelines et la mémoire cache nuisant à la prédictibilité du temps d'exécution (les optimisations processeur peuvent être utilisées par les noyaux temps réel aussi);
- limiter le surcoût processeur dû au système d'exploitation, ce qui conduit souvent à une gestion grossière du temps : l'unité de temps typique est la milli-seconde ou la dizaine de milli-secondes;
- favoriser l'équité dans l'ordonnancement au détriment du temps de réponse, utilisant des quanta de temps de l'ordre d'une dizaine de milli-secondes;
- faciliter la maintenance du système en offrant de nombreux processus de maintenance en concurrence avec les processus des utilisateurs, ce qui augmente la charge processeur et mémoire du système;
- rendre transparente l'utilisation de la mémoire, en utilisant la mémoire virtuelle, ce qui nuit à la prédictibilité.

De plus, pour arriver à ces buts, les systèmes d'exploitation généralistes sont le plus souvent monolithiques. En d'autres termes, le système d'exploitation est muni d'un noyau de base permettant la gestion des processus, leur ordonnancement et la gestion de la mémoire et de la mémoire virtuelle.

8.3 Notion de temp réel

Le qualificatif de temps réel est lié à la capacité du système à réagir en ligne et dans le respect de contraintes temporelles, à l'occurrence d'événements asynchrones issus de l'environnement extérieur. La terminologie temps réel est directement liée à l'aptitude du

système à présenter des temps de réponse(appelés aussi temps de réaction) en adéquation avec les dynamiques de l'environnement contrôlé. Ceux-ci doivent donc être suffisamment petits vis-à-vis de la vitesse d'évolution des sollicitations émises par l'environnement extérieur. Ce qui signifie que, dans le cas d'une infirmation arrivant de façon régulière (sous forme d'une interruption périodique du système), les temps d'acquisition et de traitement doivent rester inférieurs à la période de rafraîchissement de cette information.

Il existe de nombreuses définition du système temps réel mais une définition simple d'un tel système est la suivante:

> Un système temps réel est une association logiciel/matériel où le logiciel permettant, entre autre, une gestion adéquate des ressources matérielles en vue de remplir certaines tâches ou fonctions dans des limites temporelles bien précises.

Une autre définition pourrait être

> Un système est dit temps réel lorsque l'information après acquisition et traitement reste encore pertinente.

La définition du temps réel largement adoptée par les communautés scientifique et industrielle vient de Stankovic: la correction du système ne dépend pas seulement des résultats logiques des traitement, mais dépend en plus de la date à laquelle ces résultats sont produits. Le système doit ainsi être capable de réagir suffisamment rapidement pour que la réaction ait un sens. Par conséquent, une application temps réel implique généralement des activités auxquelles sont associées des contraintes temporelles. Parmi les plus courantes, on peut citer les échéances temporelles strictes des traitements.

Un système temps réel peut être embarqué ou non: un système embarqué utilise des moyens de calcul dont l'énergie est fournie pour le système lui-même. Les sources d'énergies peuvent être des batteries électriques, des moteurs alimentés eux-mêmes par du carburant, l'énergie ambiante comme l'énergie solaire, ou encore un mélange de plusieurs sources. Les systèmes embarqués sont caractérisés par des moyens de calcul consommant peu d'énergie pour favoriser l'autonomie, de petite taille comparativement aux moyens de calcul non embarqués pour diminuer l'encombrement et le poids du système de contrôle.

8.4 Caractéristiques et propriétés des systèmes temps réel

Un système temps réel se définit comme celui qui d'une part, interagit avec un environnement externe en évolution dans le temps, et qui d'autre part, réalise certaines fonctionnalités en relation avec cet environnement, et ceci en exploitant le plus souvent

des ressources limitées. Ces ressources concernent le plus souvent des entités matérielles comme de la mémoire physique, le processeur, le canal de communication, etc.

Un système temps réel se caractérise par sa nature intégrée: il constitue une composante d'un système plus imposant, doté d'interactions avec le monde physique. C'est souvent là que réside la principale source de complexité d'un système temps réel.

Dans ce contexte, deux contraintes sont pourtant à vérifier pour qu'un système temps réel soit fonctionnel:

— l'exactitude logique(logical correctness);
— l'exactitude temporelle(timeliness).

Le système doit non seulement fournir des sorties adéquates en fonction des entrées mais aussi produire les résultats sur les sorties au bon moment. Le temps de réponse(appelé aussi temps de réaction) d'un système en temps réel correspond à la durée entre la présentation des entrées à un système et l'apparition des sorties suite aux traitements effectués par ce système sur ces entrées. La spécification du temps de réponse est ainsi liée à l'échelle de temps d'évolution de l'environnement.

Un système temps réel est caractérisé, aussi, par:

— **Grande diversité des dispositifs d'entrées/sorties** : les données à acquérir qui sont fournies par les capteurs et les données à fournir aux actionneurs sont de types très variés (continu, discret, tout ou rien ou analogique). Il est aussi nécessaire de piloter un bus de terrain pour les communications.
— **Prise en compte des comportements concurrents**: l'ensemble de ces données physiques qui arrivent de l'extérieur et le réseau qui permet de recevoir des messages ne sont pas synchronisés au niveau de leurs évolutions, par conséquent, le système informatique doit être capable d'accepter ces variations simultanées des paramètres.
— **Respect des contraintes temporelles** : la caractéristique précédente impose de la part du système informatique d'avoir une réactivité suffisante pour prendre en compte tous ces comportements concurrents et en réponse à ceux-ci, de faire une commande en respectant un délai compatible avec la dynamique du système.
— **Sûreté de fonctionnement** : les systèmes temps réel mettent souvent en jeu des applications qui demandent un niveau important de sécurité pour raisons de coût ou de vies humaines. Pour répondre à cette demande, il est nécessaire de mettre en œuvre toutes les réponses de la sûreté de fonctionnement (développements sûrs, tests, méthodes formelles, prévisibilité, déterminisme, continuité de service, tolérance aux fautes, redondance, ségrégation des moyens de calcul et de communication, etc.).

8.4.1 Caractéristiques temporelles des systèmes temps-réel

Les contraintes temporelles qui sont classiquement présentées sont des contraintes de bout en bout, appelées aussi contraintes de latence. Ces contraintes représentent le délai maximal entre lecture de l'état du système (lecture des capteurs) et commande de celui-ci (commande des actionneurs). Par exemple, la commande de vol du drone, puisqu'elle doit s'exécuter à 50 Hz, c'est-à-dire avec une période de 20 millisecondes, se verra vraisemblablement munie de contraintes temporelles de bout en bout sur la commande de vol, par exemple de 20 millisecondes, afin d'obliger le système à s'exécuter une fois par période.

Le respect du protocole de communication avec les capteurs, actionneurs, ou bus de communication est une autre source, très importante, de contraintes temporelles. Ainsi, à chaque fois qu'une trame est disponible sur un réseau, le système doit la lire et soit la traiter, soit la stocker pour traitement ultérieur, sous peine de la voir être remplacée (ou « écrasée ») par la trame suivante.

Il est nécessaire de préciser et de formaliser les caractéristiques temporelles d'un système. Cette caractérisation peut prendre de nombreuses formulations. Ainsi, nous pouvons définir de manière non exhaustive:

- Durée d'exécution d'une activité : l'activité d'une application, qui peut être l'enchaînement de plusieurs activités élémentaires (acquisition, traitement, commande, affichage...), possède une durée d'exécution qui peut être mesurée de diverses manières. Cette durée n'est pas constante à chaque occurrence de cette activité puisque les programmes et les enchaînements de programmes ne sont pas toujours identiques (branchement conditionnel, itération, synchronisation...).
- Cadence de répétition ou périodicité d'une activité : l'acquisition d'une donnée ou la commande d'un actionneur peuvent nécessiter une régularité liée par exemple à la fréquence d'échantillonnage.
- Date au plus tôt ou date de réveil : dans certains cas, un traitement doit être déclenché à une date précise relative par rapport au début de l'exécution de l'application ou absolue (plus rarement). Cette date de réveil n'implique pas obligatoirement l'exécution ; il peut y avoir un délai dû à l'indisponibilité du processeur.
- Date de démarrage : cet instant correspond à l'exécution effective de l'activité.
- Date de fin : instant correspondant à la fin de l'exécution de l'activité.
- Date au plus tard ou échéance : le traitement ou la commande d'un actionneur doit être terminé à un instant fixé par rapport au début de l'exécution de l'application. Dans le cas d'applications à contraintes temporelles strictes, cette échéance doit être respectée de façon impérative, sinon il y a faute temporelle et l'application est déclarée non valide.

— Temps de réponse : cette caractéristique peut s'appliquer à une activité de régulation ou à un ensemble d'activités de régulation ; elle est directement liée à la dynamique du système. Ce paramètre correspond à la différence entre la date de réveil et la date de fin de l'activité.

— Gigue temporelle : ce paramètre caractérise la répétabilité d'une activité au fur et mesure de ses occurrences. En effet, entre deux exécutions successives d'une même activité, ses caractéristiques temporelles peuvent changer : date d'activation, durée d'exécution, temps de réponse, etc.

8.5 Quelque exemples d'application

Nous pouvons citer quelques exemples d'applications temps réel:

Système de transport: que cela soit pour le transport terrestre (véhicule de tourisme, utilitaire, poids lourd, etc.), ferroviaire, aérien, ou spatial, les systèmes de transport sont caractérisés par une forte criticité et une forte complexité due à l'expansion du nombre de fonctions et la nécessité de tolérance aux fautes. Ils sont typiquement décomposés en sous-systèmes interconnectés par un ensemble de bus de terrains.

Drone volant, roulant, navigant, ou sous-marin : ce type d'applications est de plus en plus répandu, et de plus en plus d'autonomie est donnée aux drones. Qu'ils soient d'observation ou tactiques, ce type d'application trouve une large place aussi bien dans les applications militaires que civiles (exploration de zone radioactive après un accident, planétaire, sous-marine, cinéma et télévision, etc.).

Robot de production : un robot, réalisant une activité spécifique (peinture, assemblage, tri) sur une chaîne de production, doit effectuer son travail en des temps fixés par la cadence de fabrication. S'il agit trop tôt ou trop tard, l'objet manufacturier traité sera détruit ou endommagé conduisant à des conséquences financières ou humaines graves (oubli d'un ou plusieurs rivets sur un avion).

Téléphone mobile: le système de contrôle-commande doit remplir plusieurs fonctions dont certaines ont des contraintes temporelles fortes pour avoir une bonne qualité de service (QoS : quality of service). Ainsi, la première fonction est de transmettre et de recevoir les signaux de la parole. En parallèle, il est nécessaire de localiser en permanence le relais le plus proche et donc de synchroniser les envois par rapport à cette distance (plus tôt si la distance augmente et plus tard si la distance diminue). Des messages de comptes rendus de la communication sont aussi émis avec une périodicité de plusieurs secondes. Les contraintes temporelles imposées au système doivent être imperceptibles à l'utilisateur.

Système de vidéoconférence: ce système doit permettre l'émission et la réception

d'images numérisées à une cadence de 20 à 25 images par seconde pour avoir une bonne qualité de service. Afin de minimiser le débit du réseau, une compression des images est effectuée. D'autre part la parole doit aussi être transmise. Bien que correspondant à un débit d'information moindre, la régularité de la transmission, qualifiée par une gigue temporelle, est nécessaire pour une reproduction correcte. De plus ce signal doit être synchronisé avec le flux d'images. Ces traitements (numérisations images et parole, transmission, réception, synchronisation..) sont réalisés en cascade, mais avec une cohérence précise.

Pilotage d'un procédé de fabrication (fonderie, laminoir, four verrier...) : par exemple la fabrication d'une bobine d'aluminium (laminage à froid) exige un contrôle en temps réel de la qualité (épaisseur et planéité). Cette vérification en production de la planéité nécessite une analyse fréquentielle (FFT) qui induit un coût important de traitement. Le système doit donc réaliser l'acquisition d'un grand nombre de mesures (246 Ko/s) et traiter ces données (moyenne, FFT...) à la période de 4 ms. Ensuite, il affiche un compte-rendu sur l'écran de l'opérateur toutes les 200 ms et enfin imprime ces résultats détaillés toutes les 2 s. Un fonctionnement non correct de ce système de contrôle de la qualité peut avoir des conséquences financières importantes : production non conforme à la spécification demandée.

8.5.1 Classification des systèmes temps réel

Un système en temps réel peut être classé en fonction de l'acceptabilité de l'absence de ses contraintes de temps. S'il est absolument inacceptable de ne pas respecter une contrainte de temps, par exemple, si cela peut entraîner la perte de vies humaines, nous appelons cela un système en temps réel dur. Le stimulateur cardiaque mentionné ci-dessus en est un exemple.

Si l'absence d'une contrainte de temps est acceptable mais indésirable, nous l'appelons un système en temps réel doux. Les systèmes de courriel, les routeurs sans fil et les systèmes de commande ont tous des contraintes en temps réel qu'ils sont conçus pour respecter. Même ainsi, les conséquences du non-respect de ces délais sont souvent minimes ou insignifiantes, par exemple, vous risquez d'être ennuyé parce que votre émission de télévision préférée doit se tamponner pendant une seconde. Lorsqu'un système temps réel logiciel manque une date limite pour une opération donnée, l'opération ne perd pas immédiatement toute sa valeur. La valeur diminue plutôt avec le temps, c'est-à-dire qu'elle diminue vers zéro à mesure que le temps s'écoule au-delà de la date limite.

De nombreux systèmes existent sur un spectre allant du dur au mou, où il n'est pas inacceptable de manquer une échéance, mais ce faisant, l'opération en cours perd immédiatement toute sa valeur. Les systèmes qui se situent dans ce spectre sont souvent appelés systèmes temps réel fermes (voir Figure 8.1, au milieu).

Dans un système en temps réel dur, un dépassement de délai entraîne non seulement une perte totale de valeur, mais aussi une valeur négative, c'est-à-dire un préjudice (voir Figure 1, à droite).

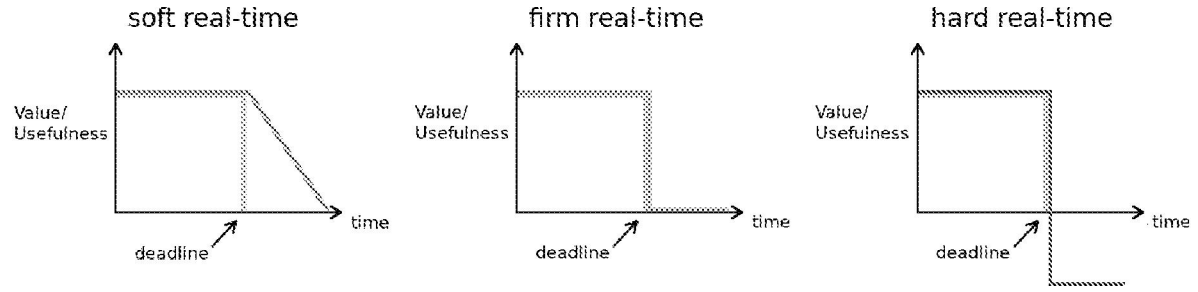

FIGURE 8.1 – Classification des systèmes temps réel

8.5.2 Les systèmes à contraintes souples

Les systèmes à contraintes souples ou *molles* (soft real time) acceptent des variations dans le traitement des données de l'ordre de la demi-seconde ou la seconde. On peut citer l'exemple des systèmes multimédia: si quelques images ne sont pas affichées, cela ne met pas en péril le fonctionnement correct de l'ensemble du système. Ces systèmes se rapprochent fortement des systèmes d'exploitation classiques à temps partagé. Ils garantissent un temps moyen d'exécution pour chaque tâche. On a ici une répartition égalitaire du temps CPU entre processus.

8.5.3 Les systèmes à contraintes dures

Les systèmes à contraintes souples ou *molles* (soft real time)pour lesquels une gestion stricte du temps est nécessaire pour conserver l'intégrité du service rendu. On citera comme exemples les contrôles de processus industriels sensibles comme la régulation des centrales nucléaires ou les systèmes embarqués utilisés dans l'aéronautique. Ces systèmes garantissent un temps maximum d'exécution pour chaque tâche. On a ici une répartition totalitaire du temps CPU entre tâches.

Les systèmes à contraintes dures doivent répondre à trois critères fondamentaux :

- La prévisibilité :
- Le déterminisme logique : les mêmes entrées appliquées au système doivent produire les mêmes effets.
- Le déterminisme temporel : une tâche donnée doit obligatoirement être exécutée dans les délais impartis, on parle d'échéance.

— La fiabilité : le système doit être disponible. Cette contrainte est très forte dans le cas d'un système embarqué car les interventions d'un opérateur sont très difficiles voire même impossibles. Cette contrainte est indépendante de la notion de temps réel mais la fiabilité du système sera d'autant plus mise à l'épreuve dans le cas de contraintes dures.

On peut résumer le comportement que doit adopter un système temps réel à travers trois caractéristiques fondamentales:

— prévisibilité;
— déterminisme;
— fiabilité.

En effet, les activités doivent être prévues et exécutées dans les contraintes de temps spécifiées. Pour garantir cela, le concepteur doit toujours se placer dans le *pire cas*. Le déterminisme consiste à écarter toute incertitude sur le comportement des activités considérées de manière individuelle ou en interaction. Parmi les sources de non-déterminisme, on peut citer la charge de calcul, les entrées-sorties, les interruptions, etc. Concernant la contrainte de fiabilité, les composants matériels et logiciels se doivent d'être fiables dans un contexte temps réel. C'est pourquoi, généralement, les systèmes temps réel sont conçus pour être tolérants aux défauts(fault-tolerant).

Un système temps réel n'est pas forcément plus rapide qu'un système à temps partagé. Il devra par contre satisfaire à des contraintes temporelles strictes, prévues à l'avance et imposées par le processus extérieur à contrôler. Une confusion classique est de mélanger temps réel et rapidité de calcul du système donc puissance du processeur (microprocesseur, micro-contrôleur, DSP).

Un système d'exploitation conventionnel, c'est-à-dire orienté temps partagé, doit organiser et optimiser l'utilisation des ressources de façon à ce que l'accès à ces ressources soit équitable. Un système d'exploitation n'a donc pour seule contrainte de temps que celle d'un temps de réponse satisfaisant pour les utilisateurs avec lesquels il dialogue. Les traitements qu'on lui soumet sont effectués en parallèle au fil de l'allocation des quanta de temps aux diverses applications.

Le système d'exploitation est capable de dater les événements qui surviennent au cours du déroulement de l'application: mise à jour d'un fichier, envoi d'un message, etc. Il permet aussi de suspendre un traitement pendant une certain délai, d'en lancer un à une certaine date...

En aucun cas, un système d'exploitation classique ne garantit que les résultats seront obtenus pour une date précise, surtout si ces résultats sont le fruit d'un traitement déclenché par un événement extérieur (émission d'un signal par un périphérique quelconque,...)

8.6 Contraintes de temps

Chaque système en temps réel a un ensemble de contraintes de temps qu'il a été conçu pour respecter. Si un système n'a pas de contraintes de temps, il n'est pas en temps réel. Ces contraintes de temps peuvent être divisées en deux catégories : la réponse aux événements et la planification des tâches.

8.6.1 Response aux évenements

Un événement est un stimulus auquel le système doit réagir. Ils peuvent être déclenchés à la fois par le matériel et le logiciel, et ils indiquent que quelque chose s'est produit et qu'il faut s'en occuper. Un événement peut sembler plus familier lorsqu'il prend la forme d'une interruption interne ou externe. Par exemple, lorsqu'un bouton est pressé, le système peut le détecter et effectuer les opérations nécessaires. Des événements peuvent être générés à tout moment lorsque le système détecte une modification. Le temps écoulé entre le moment où un système détecte un événement et le moment où il réagit à cet événement est appelé latence. La latence est définie comme le temps de réponse moins le temps de détection.

$$L = T_r - T_d \tag{8.1}$$

8.6.2 Planification des tâches

Une tâche est un ensemble d'instructions qui doivent être exécutées par le processeur du système. Certains concepteurs de systèmes en temps réel préféreront planifier les tâches, surtout si elles sont exécutées périodiquement. De nombreux systèmes embarqués ont besoin de détecter à plusieurs reprises un certain nombre d'entrées, puis de modifier les sorties en fonction de ces nouvelles informations. Ces types de systèmes se prêtent à l'utilisation de tâches. Ces tâches sont généralement planifiées et exécutées à l'aide d'un logiciel appelé planificateur. Le temps entre le moment où une tâche planifiée est censée s'exécuter et le moment où elle s'exécute réellement est appelé gigue. La gigue est définie comme le temps réel moins le temps désiré.

$$J = T_a - T_d \tag{8.2}$$

Qu'un système en temps réel utilise la réponse aux événements, la planification des tâches ou les deux, l'objectif final est d'avoir le moins de latence et de gigue possible, tout en définissant une limite supérieure (scénario du pire des cas) qui est jugée acceptable.

8.7 Modèles de conception

Comment concevoir un système embarqué en temps réel ? Au fur et à mesure que les exigences du système augmentent et que les contraintes de temps se resserrent, il devient de plus en plus difficile de gérer tous les besoins d'un système embarqué tout en respectant les délais. Voici quelques principes bien établis utilisés dans l'industrie.

8.7.1 Round–Robin

L'ordonnancement Round–robin est l'un des algorithmes d'ordonnancement les plus connus et les plus utilisés pour gérer les contraintes d'un système temps réel. Il fonctionne exactement comme son nom l'indique : il donne à chaque composant du système un tour de rôle pour utiliser les ressources partagées et accomplir la tâche souhaitée. La figure 2, ci-dessous, montre une CPU donnant à chaque tâche 500 ms de temps de traitement avant de passer à la tâche suivante. Les tâches peuvent être terminées ou non dans les 500 ms, et ils reprennent souvent là où ils se sont arrêtés quand leur virage se termine.

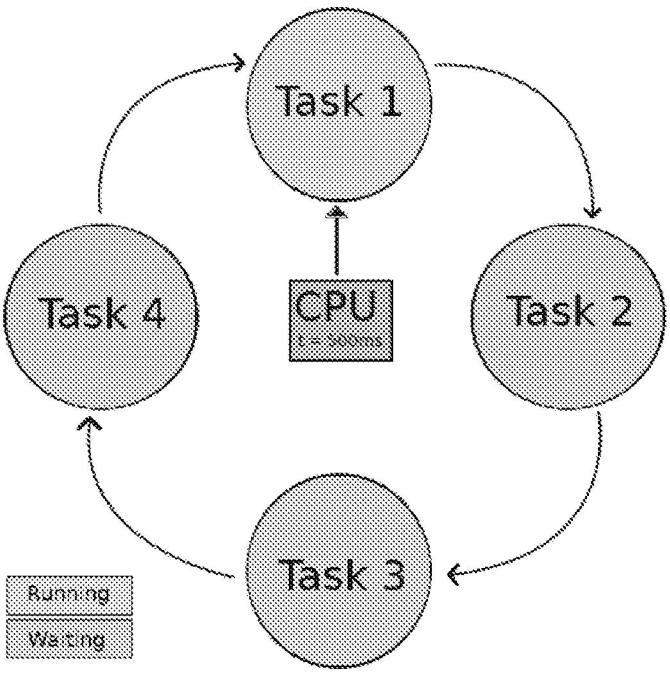

FIGURE 8.2 – Planificateur de tâches Round-robin

8.7.2 Mise en file d'attente

Une autre construction utile dans les systèmes temps réel est la file d'attente. Une file d'attente peut être considérée comme une ligne de production où les articles entrent et attendent d'être traités. Une fois prêt, le système supprime le poste suivant de la file d'attente et le traite. De cette façon, une nouvelle tâche qui doit être exécutée peut être placée dans la file d'attente et attendra pendant que le système s'occupe des tâches qui l'ont précédée. En fin de compte, la nouvelle tâche est prise en charge, même si d'autres tâches viennent s'y ajouter. La figure 3 montre une file d'attente FIFO (premier entré, premier sorti), où la tâche C reste dans la file d'attente jusqu'à ce que la tâche A et la tâche B aient été traitées.

FIGURE 8.3 – File d'attente FIFO

8.7.3 RTOS

Il arrive un moment dans la conception et la mise en œuvre d'un système en temps réel où la gestion des contraintes de temps est si lourde que l'utilisation d'un modèle ou d'un principe de conception unique ne devient plus possible. C'est à ce stade qu'un système d'exploitation en temps réel devient la meilleure solution. Un système d'exploitation en

temps réel, ou RTOS (Real-Time Operating System), utilise les modèles de conception de l'ordonnancement et des files d'attente, mais il ajoute d'autres fonctionnalités, notamment la priorité des tâches, le traitement des interruptions, les communications entre tâches, les systèmes de fichiers, le multithreading et bien plus. Il en résulte la méthode la plus efficace pour atteindre et dépasser les objectifs en matière de contraintes de temps.

8.8 Architecture des applications temps réel

8.8.1 Architecture logicielle des applications temps réel

Architecture multitâche

Le comportement concurrent des événements et grandeurs physiques externes amène à décrire l'environnement comme un système fortement parallèle. Cela conduit naturellement à adapter les méthodes de conception et de réalisation du système de contrôle-commande d'un tel environnement à ce parallélisme. Aussi, l'architecture la mieux adaptée pour répondre à ce comportement parallèle du procédé externe est une architecture multitâche. Ainsi au parallélisme de l'environnement, la réponse est le parallélisme de conception. Nous pouvons définir la tâche ou activité ou processus comme « une entité d'exécution et de structuration de l'application ». Cette architecture logicielle multitâche facilite la conception et la mise en œuvre et surtout augmente l'évolutivité de l'application réalisée.

D'une manière très générique, la fgure xxx donne l'architecture logicielle d'une application de contrôle-commande multitâche. Nous pouvons ainsi découper cet ensemble de tâches ou activités selon les groupes suivants :

- Tâches d'entrées/sorties : ces tâches permettent d'accéder aux données externes par l'intermédiaire de cartes d'entrées/sorties et ensuite de capteurs et d'actionneurs directement liés au procédé géré. Ces tâches peuvent être activées de façon régulière ou par interruption.
- Tâches de traitement : ces tâches constituent le cœur de l'application. Elles intègrent des traitements de signaux (analyse spectrale, corrélation, traitement d'images, etc.) ou des lois de commande (régulation tout ou rien, régulation du premier ordre, régulation PID, etc.). Dans le cadre de cet ouvrage, nous considérerons ces tâches comme des boîtes noires, c'est-à-dire que le traitement effectué par ces tâches relève des domaines comme le traitement du signal, le traitement d'images ou l'automatique, disciplines qui débordent largement le contexte de ce cours.
- Tâches de gestion de l'interface utilisateur : ces tâches permettent de présenter l'état du procédé ou de sa gestion à l'utilisateur. En réponse, l'opérateur peut

modifier les consignes données ou changer les commandes. Ces tâches peuvent être très complexes et coûteuses en temps de calcul si l'interface gérée est de taille importante (tableau de bord) ou de type graphique (représentation 3D).

— Tâches de communications : ces tâches sont destinées à gérer les messages envoyés ou reçus à travers un ou plusieurs réseaux ou bus de terrain. Si ce type de tâches existe, l'application est dite distribuée ou répartie.

— Tâches de sauvegarde : ces tâches permettent de stocker l'état du système à des instants fixés. Cette sauvegarde peut être utilisée a posteriori pour analyser le fonctionnement de l'application ou lors d'une reprise d'exécution à une étape précédente

Après l'analyse et la conception de l'application, nous obtenons un ensemble de tâches ou activités qui coopèrent afin de réaliser le contrôle-commande du procédé géré. Ces tâches appartiennent aux différents groupes listés précédemment : tâches d'entrées/sortie, tâches de traitement, tâches de gestion de l'interface utilisateur, tâches de communications et tâches de sauvegarde. Ce découpage purement fonctionnel peut être modifié dans certains cas en utilisant une conception tournée vers les entités ou « objets » à contrôler.

Les tâches obtenues, qui constituent l'application de contrôle-commande, ne sont pas des entités d'exécution indépendantes. En effet certaines tâches sont connectées vers l'extérieur pour les entrées/sorties. De plus elles peuvent être liées par des relations de type (voir fgure xx) :

— **Synchronisation** : cela se traduit par une relation de précédence d'exécution entre les tâches;

— **Communications** : à la notion de précédence, traduite par la synchronisation, s'ajoute le transfert de données entre les tâches ;

— **Partage de ressources** : les tâches utilisent des éléments mis en commun au niveau du système comme des zones mémoire, des cartes d'entrées/sorties, cartes réseau, etc. Certaines de ces ressources, comme par exemple les zones mémoire, ne sont pas ou ne doivent pas être accessibles, pour avoir un fonctionnement correct, par plus d'une tâche à la fois, elles sont dites ressources critiques.

Modèles d'exécution et ordonnancement

Cette architecture logicielle peut être vue comme un ensemble de tâches synchronisées, communicantes et partageant des ressources critiques. Le rôle essentiel du système informatique est donc de gérer l'enchaînement et la concurrence des tâches en optimisant l'occupation du processeur, cette fonction est appelée l'ordonnancement. L'ordonnancement est un point crucial des systèmes temps réel; en effet l'ordonnancement va déterminer le comportement temporel et être le garant du respect des contraintes de temps imposées

à l'exécution de l'application.

Nous pouvons distinguer deux modèles d'exécution des systèmes temps réel: l'exécution dite synchrone et l'exécution asynchrone. Nous allons présenter ces deux modèles d'exécution à l'aide d'un modèle d'application très simple. Cette application, constituée d'un ensemble de tâches pour gérer le procédé, intègre en particulier les deux tâches suivantes:

— Tâche de lecture des données entrées par l'opérateur à l'aide d'un clavier, appelée `Lecture_consigne`. L'intervention humaine fait que cette tâche peut être longue.

— Tâche d'alarme qui se déclenche sur un événement d'alerte correspondant au dépassement d'un paramètre critique, appelée `Alarme`. Celle-ci doit s'exécuter au plus vite pour éviter l'endommagement du procédé.

Pour mettre en avant les différences entre les deux modèles d'exécution nous allons étudier la situation dans laquelle la tâche `Lecture_consigne` s'exécute et la tâche `Alarme` demande son exécution alors que la tâche `Lecture_consigne` n'est pas terminée.

Dans le modèle d'exécution synchrone, la perception de l'occurrence de tout événement par le système est différée du temps d'exécution de la tâche en cours.

Dans l'exemple proposé, nous pouvons constater que la prise en compte d'un signal d'alerte n'est effective que lors de la fin de la tâche `Lecture_consigne`. D'un point de vue du procédé, la réaction est perçue comme différée, alors que du point de vue du système informatique, elle est perçue comme immédiate.

L'occurrence des événements externes a donc été artificiellement synchronisée avec le système informatique, d'où le nom d'exécution synchrone.

Ce retard peut affecter la prise en compte de n'importe quel événement, quelle qu'en soit la gravité pour l'application. Il faut donc vérifier que l'architecture opérationnelle choisie permettra de prendre en compte les contraintes temporelles : hypothèse de la fenêtre de visibilité des événements ou d'instantanéité des actions. La capacité du système à appréhender un événement externe est caractérisée par la durée de la tâche la plus longue puisque les tâches sont non interruptibles ou non préemtibles.

Dans le cas du modèle synchrone d'exécution, nous avons un système d'ordonnancement complètement prévisible et, en conséquence, il est possible en faisant une analyse exhaustive de l'exécution de produire une séquence d'exécution qui est jouée de façon répétitive. Cette étude de la séquence est appelée analyse de l'ordonnancement hors-ligne. L'ordonnancement peut se réduire à un séquencement. Nous avons alors un environnement informatique très simple de l'application développée puisqu'il se réduit à une liste de tâches à exécuter. L'environnement informatique pour piloter cette liste de tâches se réduit à un système très simple : un séquenceur.

Dans le modèle d'exécution asynchrone, l'occurrence de tout événement est immédia-

tement prise en compte par le système pour tenir compte de l'urgence ou de l'importance. Dans l'exemple proposé, nous pouvons constater que la prise en compte d'un signal d'alerte est immédiate sans attendre la fin de la tâche `Lecture_consigne`. La prise en compte de l'événement `alerte` est identique pour le procédé et le système informatique. L'occurrence des événements externes n'est pas synchronisée avec le système informatique, d'où le nom d'exécution asynchrone.

Dans ce contexte, nous avons des tâches qui sont interruptibles ou préemtibles.

En conséquence, l'ordonnancement n'est pas totalement prévisible et l'analyse de l'exécution des tâches doit se faire en ligne par simulation ou par test. Cela nécessite l'utilisation d'un gestionnaire centralisé des événements et de la décision d'exécution : exécutif ou noyau temps réel.

Pour terminer cette section nous allons rappeler trois définitions importantes que nous avons utilisées et fixer le contexte de cet ouvrage. Nous avons ainsi défini :

- **Tâche non préemptible ou préemptible :**
  - Une tâche non préemptible ne peut être interrompue qu'à des endroits spécifiques et à la demande de la tâche elle-même : `fin_de_tâche`, `attente_signal`... Dans ce cas, la programmation est plus simple et aucun mécanisme de partage de ressources critiques n'est à prévoir. En revanche, des temps de réponse longs peuvent se produire.
  - Une tâche préemptible peut être interrompue à n'importe quel instant et le processeur être affecté à une autre tâche. Dans ce cas, les temps de réponse à un événement externe peuvent être très courts ; mais nous avons alors une programmation plus complexe avec un besoin de mécanisme de partage de ressources critiques.
- Analyse de l'ordonnancement hors-ligne ou en ligne :
  - Une analyse de l'ordonnancement hors-ligne correspond à la construction d'une séquence d'exécution complète sur la base des paramètres temporels des tâches en utilisant une modélisation (réseaux de Petri...) ou une simulation (animation ou énumération du modèle). L'ordonnanceur nécessaire est simple puisque la séquence d'exécution est prédéfnie, il se réduit à un séquenceur. En revanche, l'application ainsi figée est peu flexible.
  - Une analyse de l'ordonnancement en ligne correspond à un choix dynamique de la prochaine tâche à exécuter en fonction des paramètres de la tâche. Si le système a des contraintes temporelles, on doit utiliser préalablement à la mise en exploitation du système des tests permettant de vérifier qu'en toutes circonstances les contraintes temporelles seront respectées. On appelle ces tests des tests d'ordonnançabilité.

— Exécution synchrone ou asynchrone :
  — Une exécution est dite synchrone si les tâches sont non préemtibles et s'exécutent les unes après les autres dans un ordre qui peut être défini par une analyse hors-ligne de l'ordonnancement.
  — Une exécution est dite asynchrone si les tâches sont préemtibles et s'exécutent selon l'ordonnancement. Une analyse de la séquence doit se faire obligatoirement en ligne.

Exécutif ou noyau temps réel

Cet environnement particulier d'exécution, exécutif ou noyau temps réel, peut être assimilé à un système d'exploitation de petite taille dédié aux applications de contrôle-commande. La caractéristique fondamentale est son déterminisme d'exécution avec des paramètres temporels fixés (temps de prise en compte d'une interruption, changement de contexte entre deux tâches, etc.). Nous pouvons comparer les différences au niveau des objectifs fixés pour le noyau d'exécution d'un système informatique classique et d'un système informatique de contrôle-commande.

Un système classique n'a pas été conçu pour permettre de respecter des contraintes temporelles, mais il suit les règles suivantes :

— Politiques d'ordonnancement des activités basées sur le partage équitable du processeur : affectation identique du temps processeur à tous les processus en cours ;
— Gestion non optimisée des interruptions ;
— Mécanismes de gestion mémoire (cache...) et de micro-exécution engendrant des fluctuations temporelles (difficulté pour déterminer précisément les durées des tâches) ;
— Gestion des temporisateurs ou de l'horloge pas assez fine (plusieurs ms);
— Concurrence de l'application temps réel avec le système d'exploitation toujours actif;
— Gestion globale basée sur l'optimisation d'utilisation des ressources et du temps de réponse moyen des différents processus en cours.

Un système informatique temps réel s'attache aux caractéristiques suivantes :

— Efficacité de l'algorithme d'ordonnancement avec une complexité limitée ;
— Respect des contraintes de temps (échéances...). Ces contraintes temporelles se traduisent plus en termes de choix d'une activité à exécuter à un instant donné plutôt que de rapidité d'exécution de toutes les activités ;
— Prédictibilité (répétitivité des exécutions dans des contextes identiques) ;
— Capacité à supporter les surcharges ;
— Possibilité de certification pour les applications de certains domaines comme l'avio-

nique, l'automobile...

— Une application temps réel étant par définition un système multitâche, le rôle essentiel du noyau temps réel est donc de gérer l'enchaînement et la concurrence des tâches en optimisant l'occupation de l'unité centrale du système informatique. Les principales fonctions d'un noyau temps réel peuvent être scindées en trois groupes :

1. gestion des entrées/sorties (gestion des interruptions, gestion des interfaces d'entrées/sorties, gestion des réseaux de communications...)
2. ordonnancement des tâches (orchestration du fonctionnement normal, surveillance, changements de mode, traitement des surcharges...) ;
3. relations entre les tâches (synchronisation, communication, accès à une ressource critique en exclusion mutuelle, gestion du temps...).

Il est important de noter que les tâches sont les unités actives du système ; le noyau temps réel n'est actif que lors de son appel. Une tâche activée peut appeler le noyau temps réel par une requête. Les différentes requêtes sont servies par des modules du noyau temps réel appelées primitives. Ensuite le noyau temps réel réactive une tâche de l'application selon l'algorithme d'ordonnancement utilisé (voir fgure 1.9).

Ainsi le noyau temps réel centralise toutes les demandes d'activation des tâches et gère des tables lui permettant de comparer les priorités (ou les urgences) et l'état de ces diverses tâches, ainsi que l'état d'occupation des ressources. La décision d'activation d'une tâche étant prise, le noyau temps réel lance les modules de programmes correspondant à cette tâche et lui alloue les ressources disponibles. La tâche activée occupe un processeur jusqu'à la fin de son exécution sous le respect des conditions suivantes :

— Elle ne réalise pas d'opérations d'entrées-sorties ;
— Les ressources utilisées sont disponibles ;
— Aucun événement extérieur ne revendique le déroulement d'une tâche plus prioritaire.

8.8.2 Architecture matérielle des applications temps réel

Comme nous l'avons vu, l'aspect matériel a une très grande importance dans les applications de contrôle-commande. Cette implication est liée d'une part à la connexion directe avec le monde physique réel à l'aide d'une grande diversité de systèmes d'entrées/sorties et d'autre part au matériel informatique parfois spécifique et développé pour une application donnée. Ce dernier point concerne les applications dites dédiées et embarquées ; le matériel a été conçu et créé spécifiquement pour une application, comme un téléphone portable, une caméra vidéo, etc. A titre d'exemple, les différents calculateurs embarqués dans un

véhicule automobile sont dédiés et spécialement développés pour cette application (figure 1.13). Dans ces matériels, nous trouvons un processeur de type microcontrôleur redondé pour avoir un haut niveau de sécurité, des composants spécifiques (ASIC : Application Specific Integrated Circuit), une alimentation électrique.. Tout ce matériel est ensuite encapsulé dans un boîtier résistant à l'environnement de fonctionnement usuel (chaleur, vibrations, ambiance corrosive, etc.).

Beaucoup d'applications de contrôle-commande s'exécutent sur des plateformes PC classiques. En revanche, il est toujours nécessaire de disposer de cartes d'entrées/ sorties qui doivent souvent être synchronisées (acquisition et commande effectuées à des instants précis). Ainsi, dans ce domaine d'applications informatiques, des matériels spécifiques existent et permettent ce fonctionnement temporel : c'est par exemple le cas de la plateforme PXI concaténation d'une plateforme classique PC et de cartes d'entrées/sorties synchronisées.

Enfin, dans de nombreux cas, les applications de contrôle-commande sont de type distribué, c'est-à-dire qu'elles sont composées de plusieurs sites ou processeurs, sur lesquels s'exécutent un environnement multitâche, reliés par un ou des réseaux informatiques (Ethernet, CAN, FIP...).

8.9 Les types de temps réel

- **Temps réel mou** : un retard dans la réponse n'est pas dramatique (distributeur de billets)
- **Temps réel dur**: un retard dans l'obtention du résultat le rend inutile (détection de missile)
- **Temps réel ferme**: un retard, s'il arrive très peu souvent, peut être toléré(téléphonie).

La plupart des systèmes temps-réel sont hybrides.

8.10 Les types de contraintes

- **Précision**: effectuer certains opérations à un moment précis,
- **Temps de réponse**: effectuer certains opérations en un temps maximum(système de freinage ABS) ou avec un temps moyen fixé (distributeur de billets)
- **Rendement**: nombre de requêtes traitées par unité de temps(robot de production dans une usine).

8.11 Exemples de contraintes temps réel

— Prélever la température toutes les 50 secondes.
— Déclencher l'airbag au plus tard 10 ms après l'impact.
— Envoyer 25 images par seconde.
— Lancer la sirène de l'usine à midi.
— Mettre à jour l'état du stock au plus tard 10 min après la vente de produit.

8.12 Tâches temps réel

Une tâche temps réel est une séquence d'instructions constituant l'unité de base d'un système temps réel. Les tâches réalisent des entrées/sorties et des calculs permettant de commander des procédés via un ensemble de capteurs et d'actionneurs, éventuellement tout ou rien, par exemple ensemble de tâches réalisant le contrôleur de vitesse d'une voiture ou le pilotage automatique d'un avion. Elles sont soumises à des contraintes temporelles qui doivent être respectées. Par exemple il faut désactiver avant un certain temps le contrôleur de vitesse d'une voiture lorsqu'on actionne les freins.

On suppose que les caractéristiques temporelles sont des entiers non négatifs multiples d'unité de temps en général égale à la période de l'horloge du processeur.

Les tâches temps réel peuvent être divisées en trois types:

8.12.1 Les tâches périodiques

Les tâches périodiques se répètent indéfiniment et leurs instances sont séparées par une période constante. Une tâche périodique est activée par le déclenchement d'une interruption produite par un timer externe, à une date appelée date d'activation. Chaque instance s'exécute pendant une durée d'exécution maximale et doit terminer son exécution avant une échéance relative à sa date d'activation. Une tâche périodique est représentée par quatre paramètres : une date d'activation, un temps d'exécution maximum (pire cas), une échéance relative et une période.

8.12.2 Les tâches apériodiques

Les tâches apériodiques doivent s'exécuter au moins une fois et ne se répètent pas nécessairement indéfiniment. Une tâche apériodique est représentée par trois paramètres : une date d'activation, un temps d'exécution maximum et une échéance relative à sa date d'activation.

8.12.3 Les tâches sporadiques

Les tâches sporadiques sont un cas particulier de tâches apériodiques. Ce sont des tâches qui se répètent indéfiniment avec une période minimum. En d'autres termes, deux instances peuvent être séparées d'une valeur plus grande que cette période minimum.

Une tâche sporadique est représentée par quatre paramètres : une date d'activation, un temps d'exécution maximum, une échéance relative à sa date d'activation et une durée minimum entre deux activations successives de la tâche.

8.13 Les contraintes temps réel

Dans un système temps réel, les tâches peuvent être soumises à plusieurs contraintes telles que des contraintes d'échéances, de périodicité stricte, de dépendances et de précédences, etc.

8.13.1 Échéances

Les contraintes d'échéances permettent d'exprimer une condition sur la date de fin d'exécution au plus tard d'une tâche. Considérons un système de tâches périodiques, suivant la relation entre la période T_i et l'échéance relative D_i de chaque tâche i nous distinguons trois types d'échéances:

8.13.2 Périodicité stricte

8.13.3 Dépendances entre tâches

8.13.4 Latence

8.14 L'ordonnanceur

Un ordonnanceur temps réel est un programme temps réel chargé d'allouer une ou des tâche(s) à ordonnancer au(x) processeur(s). Il est invoqué à certains instants. Et ceci suivant un algorithme ordonnancement donné de tel sorte que toutes les tâches respectent leurs contraintes temps réel.

Les tâches sont les entités de base de l'ordonnancement temps réel et elles sont périodiques ou apériodiques, à contraintes temporelles strictes ou relatives. Les stratégies d'ordonnancement sont des règles qui sélectionnent la prochaine tâche qui doit être exécutée dans un système multitâche. Autrement dit, elles décident de l'ordre dont les tâches vont accéder à la ressource la plus importante du système, le processeur. Au niveau du

noyau du système d'exploitation ces stratégies constituent l'ordonnanceur (scheduler en Anglaos). L'algorithme d'ordonnancement détermine, selon la politique (la stratégie d'ordonnancement) implantée, l'ordre effectif d'exécution des tâches : l'ordonnancement ou, autrement dit, le scénario des tâches.

.

Différentes méthodes d'implantation de l'ordonnanceur sont envisagées, selon le type de machine mono ou multiprocesseur. Pourtant, dans les deux cas, les politiques d'ordonnancement reposent sur les mêmes principes, et les informations dont elles ont besoin représentent les différents paramètres des tâches.

8.14.1 Algorithmes d'ordonnancement

Un système temps réel est constitué d'un ou plusieurs processeurs et d'un ensemble de tâches ayant des contraintes temps réel. Un algorithme d'ordonnancement détermine l'ordre total et les dates de démarrage des exécutions des tâches sur un ou plusieurs processeurs. Il existe plusieurs classes d'algorithmes d'ordonnancement temps réel

- statiques pilotés par table;
- statiques préemptifs basés sur des priorités;
- dynamiques basés sur une planification de l'exécution;
- dynamiques basés sur la notion du meilleur effort (best effort).

Les algorithmes dynamiques sont les plus flexibles car ils permettent la prise en compte d'événements « non prévus ». Parmi ceux-ci les principaux sont :

l'ordonnancement par priorités fixes

- premier arrivé, premier servi (First-Come, First-Served-FCFS) où toutes les tâches ont la même priorité;
- le tour de rôle ou tourniquet (Round Robin - RR);
- monotone par fréquences (Rate-Monotonic - RM);
- monotone par échéances (Invers Deadline - ID, ou Deadline Monotonic - DM)

l'ordonnancement par priorités dynamiques

- Earliest Deadline First - EDF ;
- Least Slack Scheduling - LLS, ou Least Laxity First - LLF, ou Shortest Slack Time - SST.

Dans un cadre plus général, les mêmes principes d'ordonnancement s'appliquent aux tâches sur une machine monoprocesseur comme sur une plate-forme multiprocesseur. Dans

ce dernier cas quelques hypothèses spécifiques à l'existence de plusieurs processeurs sont prises en compte :

— l'exécution d'une tâche n'est pas associée à un processeur déterminé;
— à tout instant une tâche ne peut s'exécuter que sur un seul processeur.

8.14.2 Invocation de l'ordonnanceur

Il existe plusieurs manières d'invoquer un ordonnanceur:

— Invocation guidée par les évènements(Event-Triggered): l'ordonnanceur est invoqué sur réception d'un évènement tel que l'activation d'une tâche, la libération d'une ressource (processeur, périphériques d'entrées et de sorties, etc.), la fin d'exécution d'une tâche. Ce mode d'invocation est utilisé dans l'ordonnancement en ligne qui prend les décisions d'ordonnancement en fonction des évènements qui se produisent lors de l'exécution des tâches;
— invocation guidée par le temps (Time-Triggered: les invocations sont indépendantes des évènements. Les dates d'invocation de l'ordonnanceur sont fixées avant l'exécution des tâches et sont généralement périodiques. Cela définit des tranches de temps (slot) pendant lesquelles on peut effectuer un traitement, alors que dans les approches dirigées par les évènements (Event-Trigger) le traitement est directement activé par un événement. Ce mode d'invocation de l'ordonnanceur est utilisé dans l'ordonnancement hors ligne où toutes les décisions d'ordonnancement ont été prises hors ligne.

8.15 États et gestion des tâches

Le partage du ou des processeur(s) et des ressources introduit plusieurs états pour une tâche :

— **Inactif** : la tâche n'est pas encore activée,
— **Prêt** : la tâche est activée et elle dispose de toutes les ressources dont elle à besoin pour s'exécuter,
— **Bloqué** : la tâche est en attente de ressources,
— **Exécution** : la tâche s'exécute,
— **Passif** : la tâche n'a pas de requête en cours, elle a fini son exécution.

L'ordonnanceur est chargé d'assurer la transition d'une tâche d'un état à un autre. À chaque invocation, l'ordonnanceur met à jour la liste des tâches prêtes en y ajoutant toutes les tâches activées et qui disposent de leurs ressources et en supprimant les tâches qui ont fini leurs exécutions ou qui sont bloquées par l'attente d'une ressource. Ensuite parmi les tâches prêtes, l'ordonnanceur sélectionne la tâche la plus prioritaire pour s'exécuter.

Ainsi, une tâche dans l'état inactif peut passer à l'état prêt. Une tâche à l'état prêt peut passer à l'état exécution ou à l'état bloqué. Une tâche à l'état exécution peut revenir à l'état prêt si elle est préemptée par une autre tâche plus prioritaire, elle peut passer à l'état bloqué si elle attend la libération d'une ressource ou si elle a fini son exécution, elle passe à l'état passif. Une tâche peut passer de l'état bloqué à l'état prêt. Et enfin une tâche peut passer de l'état passif à l'état prêt. La figure 8.4 donne une illustration des différents états et leurs transitions.

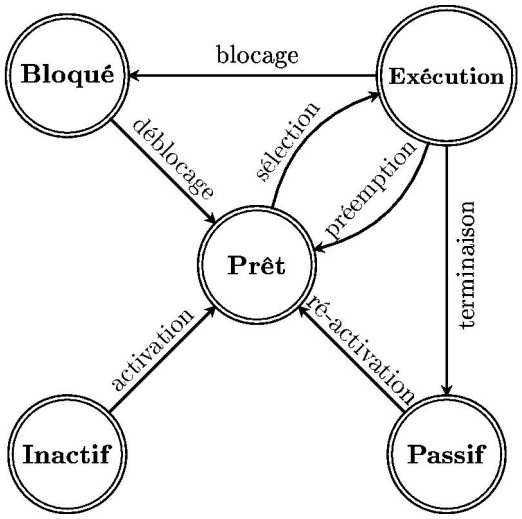

FIGURE 8.4 – États et transition d'une tâche

8.16 Ordonnancement Monoprocesseur

8.16.1 Priorités fixes

Une priorité fixe aux tâches est une priorité qui ne varie pas au cours de l'exécution de la tâche. Les algorithmes d'ordonnancement à priorités fixes aux tâches les plus utilisés sont Rate Monotonic et Deadline Monotonic.

8.16.2 Priorités dynamiques

8.17 Exemples de systèmes d'exploitation temps réel dans l'embarqué

8.17.1 FreeRTOS

Disponible sous licence GPL, cet RTOS open-source pour mictrocontrôleur et petits microprocesseur est l'un des systèmes d'exploitation leaders dans le monde de l'embarqué. Il se destine aux configurations matérielles de petite taille(applications à faible empreinte mémoire) mise

8.17.2 Keil RTX