CONCEPTION D'UN NOYAU TEMPS RÉEL PRÉEMPTIF

Lanteri Isabelle

CNAM 1993/94

Table des matières

Index des illustrations

1 Présentation

On présente ici un noyau temps réel développé en 1993/94 au CNAM dans le cadre des travaux pratiques de Génie Logiciel. Ce noyau conçu en assembleur 8086 Intel tourne sur PC.
On définit ici les principaux objectifs à atteindre et les contraintes à respecter lors du développement.
On décrit d'abord les caractéristiques générales des noyaux temps réel conformément aux normes fixées par le rapport SCEPTRE.
On définit ensuite les exigences propres auxquelles devra répondre notre noyau vu son contexte d' implantation et de développement.
La dernière partie fixe les impératifs à respecter sur le plan de la qualité du logiciel, notamment en ce qui concerne la documentation et les méthodologies utilisées pour le développement.

2 Noyau temps réel préemptif multitâche

On se propose de développer un noyau temps réel avec préemption permettant de gérer un système temps réel multitâche sur un micro-ordinateur de type PC.
Nous allons d'abord définir les termes employés.

- **Temps réel**

Les systèmes temps réel concernent des systèmes informatiques où les contraintes de temps sont fortes, un programme gérant les processus par scrutation séquentielle ne pourrait pas opérer de façon efficace.
Un programme temps réel gère l' occupation du processeur en fonction des échéances de temps liées au système, il active ou désactive des morceaux de programme appelés tâches.

- **Multitâche**

Une tâche consiste en une suite d' actions, elle opère de façon asynchrone et en temps réel. Plusieurs tâches peuvent opérer ensemble, elles semblent être accomplies simultanément alors qu' en faît le processeur commute d' une tâche à l' autre : c' est le noyau qui attribue le processeur aux tâches de façon intermittente.
Il existe également des systèmes multi-processeurs où chaque tâche peut posséder son propre processeur.
Les tâches soumises à de fortes contraintes de temps sont liées à des interruptions matérielles (déclenchées par des circuits périphériques extérieurs) et sont appelées tâches matérielles ou immédiates : elles sont immédiatement activées lors d' une demande. Ce peut être, par exemple, un pilote activant une commande de tir.
Les tâches logicielles ou différées sont exécutées par morceaux selon un ordre de priorité fixé par le programme. Le noyau gère l' occupation du processeur suivant ces niveaux de priorité.

- **Préemption**

En première approche, les noyaux temps réel peuvent être classés en deux catégories :

- les noyaux sans réquisition du processeur ou non préemptifs
- les noyaux avec réquisition du processeur ou préemptifs

Dans le premier cas, une tâche est exécutée jusqu'à ce qu'elle fasse appel à un service du noyau. Celui-ci décide alors de la poursuite ou de l'arrêt de la tâche.
Dans le cas d'un noyau préemptif, une tâche peut à tout instant perdre le processeur au bénéfice d'une autre tâche de plus haute priorité. Pour ceci, une horloge génère une interruption à intervalles de temps fixes. Le programme associé scrute alors l'état des différentes tâches et attribue le processeur à la tâche de plus forte priorité.

3 Caractéristiques principales des exécutifs temps réel

Le rapport SCEPTRE (Proposed Standard For a Real Time Executive Kernel) dont une première mouture fut fournie vers 1980, définit les concepts généraux auxquels doit répondre un exécutif temps réel et propose une norme de noyau.
L 'ensemble des services que doit fournir un exécutif temps réel est résumé dans le tableau suivant :

- Communication
- Synchronisation
- Gestion et ordonnancement des tâches
- Gestion de la mémoire
- Gestion des interruptions et E/S physiques
- E/S logiques et gestion des périphériques
- Gestion des fichiers (logique et physique)
- Gestion des programmes
- Gestion des travaux et des transactions
- Traitement des erreurs et des exceptions
- Gestion du temps

En ce qui concerne le noyau lui-même, SCEPTRE se limite aux éléments suivants :

- La gestion des tâches
- La signalisation
- L'exclusion mutuelle
- La communication entre tâches
- La communication avec l' extérieur
- Le traitement des situations d' exception

Ces services sont fournis par un ensemble d' opérations élémentaires et de types d' objets dont la réunion définit le noyau SCEPTRE.
En ce qui nous concerne, nous nous limiterons dans un premier temps à ces dernières opérations, la conception modulaire de notre noyau permettant l' ajout de nouveaux services.
Nous allons maintenant détailler les opérations élémentaires et les entités intervenant dans notre noyau.

4 Architecture générale du système

Le système final s' organise donc en fonction des éléments suivants :

- Des tâches logicielles adressent des requêtes au noyau qui gère leur ordonnancement et leur exécution.
- Le contexte matériel est tel que le noyau remplace un système d' exploitation existant, le DOS tel que l' on doit gérer son remplacement puis sa restitution.
- Des tâches matérielles déclenchées par des interruptions extérieures peuvent également adresser des requêtes à notre noyay.
- Un sous-ensemble des tâches matérielles est constitué par les tâches immédiates système qui sont des gestionnaires d' interruption système normalement gérés par le DOS. En première approche, nous ne sommes concernés que par les interruptions horloge et clavier.
- La communication avec l' extérieur s' effectue par l' intermédiaire du clavier (saisir) et de l' écran (afficher).
- Un noyau qui est équivalent à un système d' exploitation gère les tâches logicielles, immédiates et système et répond à leurs requêtes.

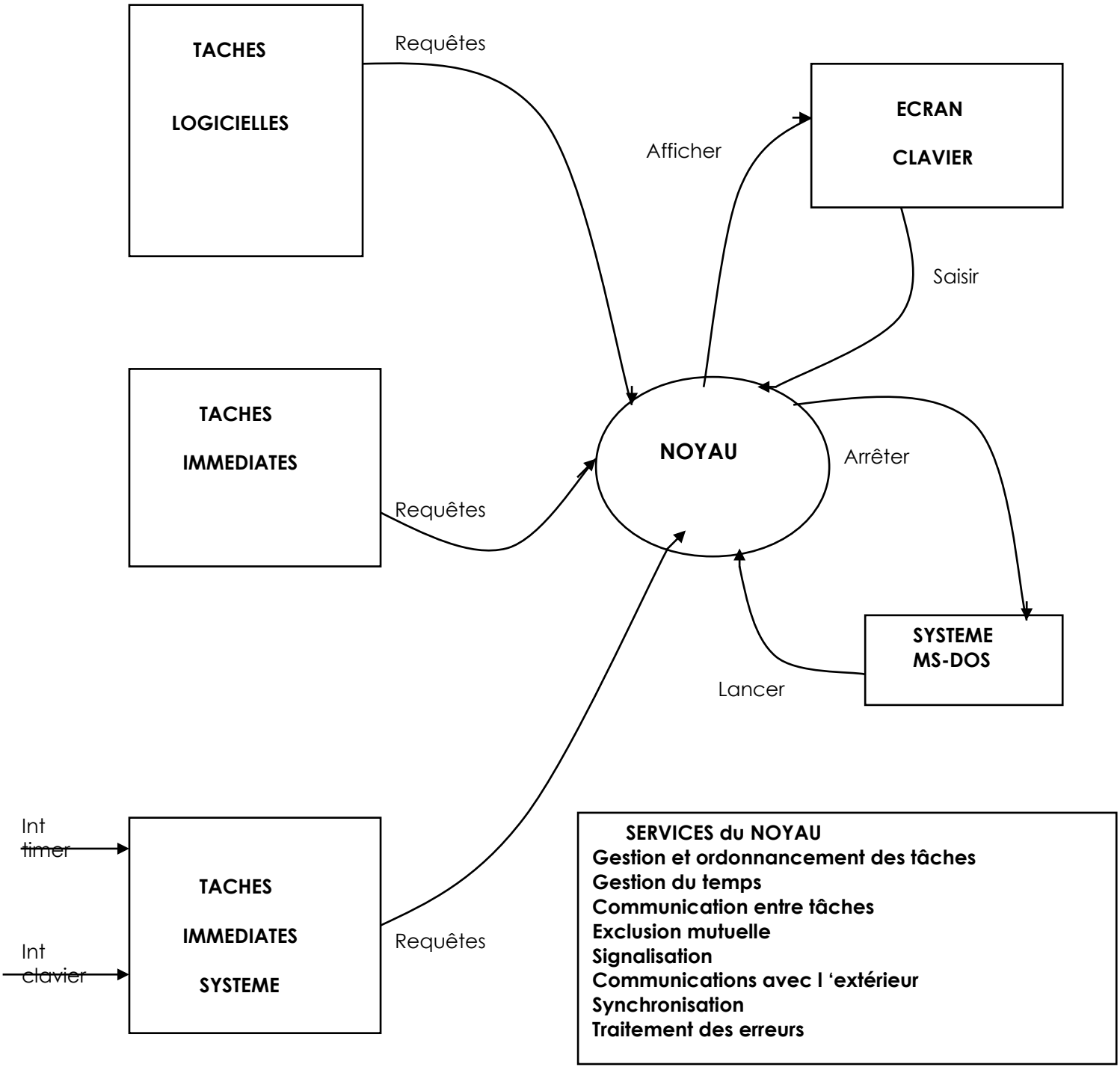

Figure 3-. Architecture générale du système

5 Etats et caractéristiques d'une tâche

Toute tâche logicielle possède ses caractéristiques propres et doit être clairement identifiable par le noyau qui traite ses requêtes.
Elle possède notamment un numéro d' identification et un niveau de priorité. Le noyau doit également connaître des éléments tels que adresse du code de la tâche lorsqu' elle commence son exécution et adresse de la pile de sauvegarde de la tâche.
La pile de sauvegarde d' une tâche contient le contexte d' exécution d' une tâche soit la valeur de tous les registres du processeur lorsque la tâche a été suspendue dans son exécution au profit d' une autre et qui devra être restituée lorsque la tâche reprendra son exécution.
Les paramètres utiles d' une tâche seront donc stockés dans une structure interne au noyau appelée BCT pour Bloc de Contrôle de Tâche.
Du point de vue du noyau, une tâche peut être :

- *__Non créée__ : la tâche n' existe pas pour le noyau et n' a donc pas de BCT.

- *__Créée__ : la tâche possède un BCT avec un numéro d' identification, un niveau de priorité et toutes les autres données utiles pour son traitement par le noyau. Elle peut alors évoluer vers l' état suivant qui est :

- **__Prête__ : la tâche est prête à être exécutée et peut occuper le processeur à tout moment. Les tâches prêtes sont rangées dans une file par ordre de priorité et la tâche en tête de cette file est :

- **__En exécution__ : la tâche occupe le processeur et est en cours d' exécution.

- **__En attente__ : l' exécution de la tâche a été suspendue. Le BCT de la tâche a été retiré de la file des tâches prêtes et placé dans une autre file d' attente sur un événement quelconque : libération d' un sémaphore, fin d' un délai de suspension programmé, demande de réveil, etc…

- **__Détruite__ : la tâche a été détruite et n' a plus de BCT. Elle ne pourra plus reprendre son exécution. La seule évolution possible est vers l' état non créée.

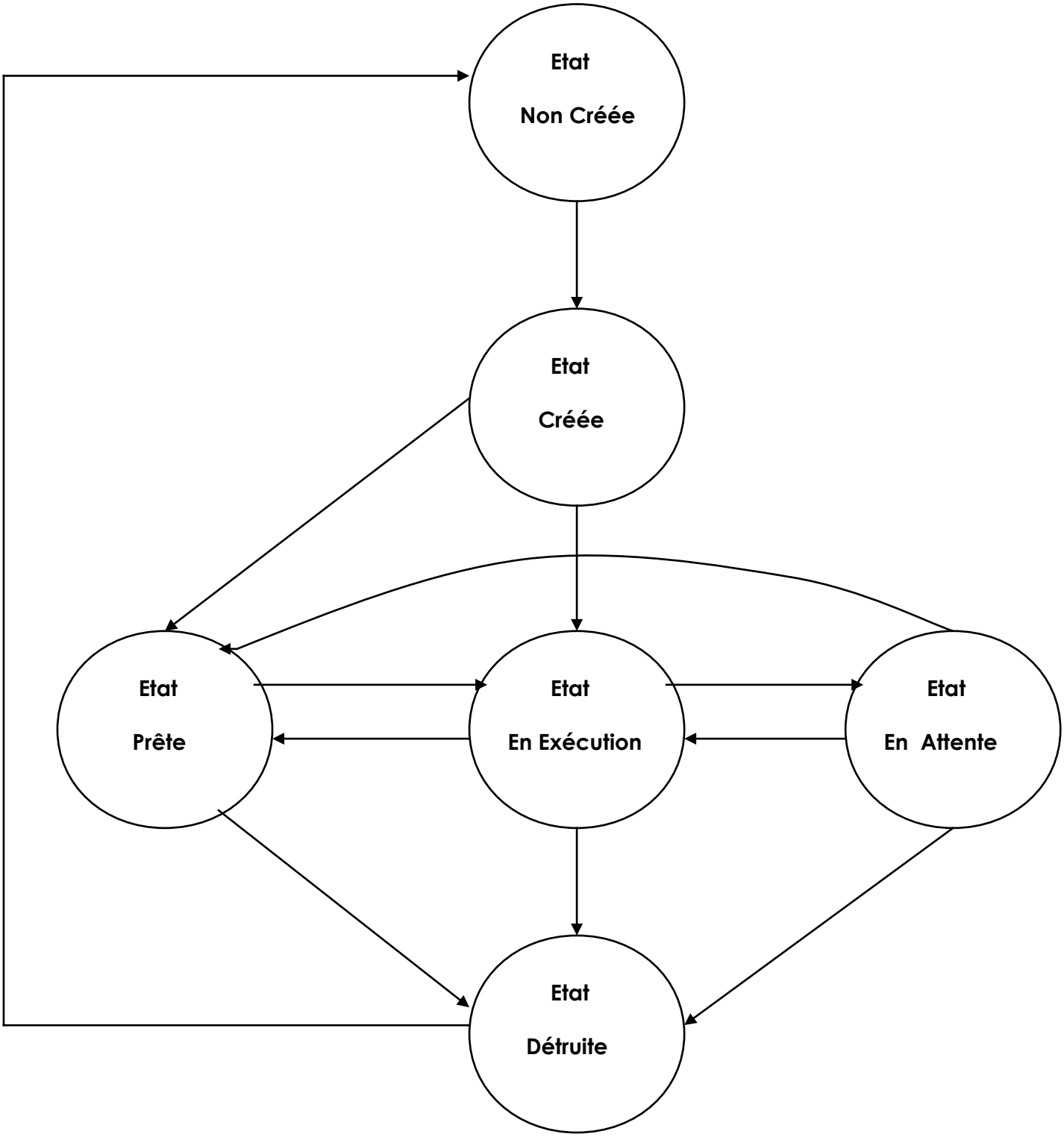

Figure 4- Etats d' une tâche et transitions possibles

6 Services

Nous allons maintenant détailler les services offerts par notre noyau. Ces services sont accessibles aux tâches applicatives par des requêtes.
Le tableau suivant récapitule l' ensemble de ces services avec les primitives d' appel.

Catégories	Mécanismes	Appels
Ordonnancement des tâches		CREER_TACHE METTRE_TACHE_EN_ATTENTE REVEILLER_TACHE CHANGER_PRIORITE DETRUIRE_TACHE ETAT_TACHE TEMPS_CYCLIQUE
Communication inter-tâches	Boites aux lettres	LIRE_MESSAGE ATTENTE_MESSAGE ENVOI_MESSAGE
Gestion du temps		INIT_HORLOGE LIRE_HORLOGE
Exclusion mutuelle	Sémaphores	ACTIVER_SEMAPHORE LIRE_SEMAPHORE ATTENTE_SEMAPHORE LIBERER_SEMAPHORE DESACTIVER_SEMAPHORE
Signalisation	Evénements	CREER_EVEN ETAT_EVEN INIT_EVEN SUP_EVEN ATTEND_EVEN ENVOI_EVEN
Gestion des tâches immédiates et systèmes		APPEL_INT FIN_INT INT_TIMER INT_CLAVIER
Communications avec l ' extérieur		ATTEND_CARACTERE AFFICHE_STATUS AFFICHE_CAR

7 Gestion et ordonnancement des tâches

Dans cette catégorie de services, notre noyau effectuera les opérations suivantes :

- CREER_TACHE (numéro d'identification, priorité)

Cette opération permet de créer une nouvelle tâche, son BCT est initialisé avec les valeurs données en paramètres et la tâche est candidate à l' exécution.

- METTRE_TACHE_EN_ATTENTE (numéro d'identification)

La tâche dont le numéro est fourni en paramètre est mise en attente. Son exécution est suspendue jusqu'à ce que son réveil soit demandé.

- METTRE_TACHE_EN_ATTENTE_DELAI (numéro d'identification, nombre de ticks)

La tâche est suspendue jusqu'à ce que le nombre de ticks spécifié soit écoulé.

- REVEILLER_TACHE (numéro d'identification)

La tâche suspendue est de nouveau candidate à l' exécution. Cet appel est obligatoirement fait par une autre tâche que celle en attente.

- CHANGER_PRIORITE_TACHE (numéro d'identification, priorité)

La tâche est candidate à l' exécution avec un nouveau niveau de priorité.

- DETRUIRE_TACHE (numéro d'identification)

La tâche n' existe plus pour le noyau et reprendre son exécution est impossible. Son BCT est libéré. Une tâche peut se détruire elle-même.

- ETAT_TACHE (numéro d'identification)

Cette requête permet de connaître l' état d' une tâche au moment de l' appel.

- TEMPS_CYCLIQUE_TACHE (nombre de ticks)

Cette primitive permet le fonctionnement en temps cyclique pour les tâches d' égale priorité. Chaque tâche s' exécute durant le nombre de ticks spécifié si elle fait partie du groupe de priorité le plus élevé. Dans le cas où ce mode n' est pas programmé, c' est la plus ancienne tâche lancée qui garde le contrôle du processeur.

8 Communication

La communication entre deux tâches consiste en un échange d' informations entre ces deux tâches selon un protocole. L' utilisation de la mémoire commune permet de réduire le protocole de communication à des opérations de synchronisation et à des manipulations de données partagées en mémoire commune.
Les stratégies les plus couramment utilisées pour la communication entre tâches utilisent la structure de file : files de tâches en attente, files de messages.
Nous avons retenu pour ce service la solution consistant à utiliser des boîtes aux lettres.

Les Boites aux Lettres

Les tâches communiquent entre elles par l' intermédiaire de boîtes aux lettres. Ces boîtes sont identifiables par leur numéro. On a prévu d' utiliser 16 boîtes numérotées de 0 à 15. A chaque boîte est associé un mot de deux octets destiné à contenir le message et une variable d' un octet destinée à contenir le numéro d' identification de la première tâche en attente d' un message sur cette boîte. Les tâches en attente à une même boîte sont insérées par ordre d' arrivée dans une file d' attente à structure FIFO.
Ces boîtes aux lettres sont initialisées avec la valeur zéro signalant qu' une boîte est vide.
Les primitives de communication permettant d' utiliser ces boîtes sont au nombre de trois :

- LIRE_MESSAGE (idf_boite)

Cette primitive permet à une tâche de prendre connaissance du message contenu dans la boîte donnée. Si la boîte est vide, elle contient le message 00 qui est transmis à la tâche appelante qui reprend son exécution

- ATTENTE_MESSAGE(idf_boite)

Cette primitive, comme la précédente, permet à une tâche de lire le contenu d' une boîte aux lettres mais si la boîte est vide, la tâche est suspendue jusqu » à réception du message.

- ENVOI_MESSAGE(idf_boite, message)

Cette primitive permet à une tâche d' envoyer un message à une boîte. Si une tâche est en attente à la boîte, le message lui est transmis et la tâche est réinsérée dans la file des tâches prêtes suivant son niveau de priorité.

9 Traitement des situations d' exception

On appelle exception une situation non attendue pouvant lorsqu'elle se produit, conduire la tâche concernée à abandonner le traitement en cours.
Une exception peut être du type matériel ou logiciel. Dans le cas d' un système implanté sur PC, il est improbable d ' avoir à traiter des cas de problèmes matériels. Au niveau mémoire, tout est alloué de façon statique.
Au niveau logiciel, la correction et la cohérence de toutes les requêtes seront vérifiées et en cas d' erreur ou d' incohérence, par exemple créer une tâche déjà existante, la requête sera refusée et un code d' erreur renvoyé à la tâche appelante.

10 Gestion du temps

Le système gère une horloge temps réel de deux mots (32 bits) en mémoire.
Cette horloge est initialisée à la valeur zéro au lancement du système. Elle contient le nombre de ticks en cours, le tick étant le plus petit intervalle de temps que peut mesurer le système. Sur un PC, le timer déclenche une interruption 18,2 fois par seconde et le tick ou unité de temps du système vaudra 55 ms.
Pour accéder à cette horloge, l' utilisateur disposera de deux primitives :

- INIT_HORLOGE
  Cet appel permet de réinitialiser l' horloge temps réel à la valeur 00.

- LIRE_HORLOGE
  Cette requête permet à la tâche appelante de lire la valeur courante de l' horloge temps réel.

11 Exclusion mutuelle

Le parallélisme des tâches induit un phénomène de compétition pour la manipulation des objets partagés. Un mécanisme d'exclusion mutuelle assure que des séquences d' instruction manipulant un même objet partagé soient disjointes dans le temps quel que soit l' ordre dans lequel ces séquences sont invoquées. On appelle section critique une telle séquence.
La solution retenue pour traiter ce problème consiste à utiliser des sémaphores à compte.

Les sémaphores

L' utilisation de sémaphores permet de contrôler l' accès des tâches à une ressource.
Chaque sémaphore est constitué d' un compteur et d' une variable contenant le numéro de la première tâche en attente du sémaphore s'il en existe. Les tâches en attente d'un sémaphore sont insérées dans une file d'attente ordonnée suivant lesniveaux de priorité.
Le compteur est initialisé avec le nombre d'accès simultanés possibleà la resssource 'en général 1) ou avec la valeur zéro. Dans ce cas, le sémaphore est dans l' état initial bloqué.
Si une tâche demande un accès à la ressource, le sémaphore est incrémenté d'une unité.

Réciproquement, le sémaphore est incrémenté d'une unité lorsque la tâche libère la ressource.
Si la valeur du sémaphore est strictement positive, le sémaphore est libre et la ressource disponible.
Si le sémaphore vaut zéro, tous les accès vers la ressource sont bloqués.
Si la valeur du sémaphore est strictement négative, sa valeur absolue indique le nombre de tâches en attente de la ressource.
On a prévu d' utiliser jusqu'à 16 sémaphores numérotés de 0 à 15.
Les primitives utilisées pour gérer les sémaphores sont :

- ACTIVER_SEMAPHORE (idf_semaphore, valeur_init)
  Cette primitive signale au noyau que l' on va utiliser le sémaphore dont on a fourni le numéro et la valeur initiale.

- LIRE_SEMAPHORE(idf_semaphore)
  Pour connaître la valeur d' un sémaphore.

- ATTENTE_SEMAPHORE(idf_semaphore, temps_attente)
  Cette primitive permet à une tâche d' accéder à la ressource. Si la valeur du sémaphore appelé est négative ou nulle, le sémaphore est décrémenté de 1, le BCT de la tâche est inséré dans la file des tâches en attente au sémaphore en fonction de sa priorité et il y a commutation. Si le paramètre temps d' attente est différent de zéro, la tâche sera réactivée après le délai spécifié même si elle n' a pas eu accès à la ressource.
  Si la valeur du sémaphore est strictement positive, le sémaphore est décrémenté de 1 et la tâche reprend son exécution en occupant la ressource.

- LIBERER_SEMAPHORE(idf_semaphore)
  La tâche appelante signale au noyau qu' elle libère la ressource. Le sémaphore est incrémenté de 1. Si la valeur du sémaphore est négative ou nulle, des tâches sont en attente de la ressource. La tâche appelante reprend son exécution.
  Si la valeur du sémaphore est négative ou nulle, des tâches sont en attente de la ressource. La première tâche en attente dans la file va être réactivée afin de pouvoir accéder à la ressource.

- DESACTIVER_SEMAPHORE(idf_semaphore)
  Cet appel signale au noyau que le sémaphore précisé doit être désactivé. Si des tâches sont en attente au sémaphore, elles sont réactivées et réinsérées dans la file des tâches prêtes.

12 Signalisation

La signalisation permet d' exprimer des protocoles de synchronisation. Elle nécessite une opération pour chacune des tâches concernées : attente du signal pour l'une et émission du signal pour l' autre. Ce mécanisme sera matérialisé par des objets appelés événements.
Un événement est représenté par une variable booléenne qui traduit à un instant donné le faît que l'événement est arrivé (valeur 1) ou ne l'est pas (valeur 0).
Si l' événement survient alors qu'il n'est pas attendu, deux possibilités sont à envisager :

1) l 'événement est mémorisé et la tâche réceptrice sera réactivée dès sa demande de mise en attente.
2) l' événement n' est pas mémorisé et la tâche réceptrice restera bloquée après sa mise en attente.

Pour permettre à une tâche d 'attendre plusieurs événements simultanément, on crée des groupes d' événements, chaque bit étant significatif d' un événement.
Notre noyau utilisera des groupes d' événements de 16 bits, tout événement sera mémorisé jusqu' à sa demande d 'effacement.

Les événements

Les primitives gérant les groupes d' événements de 16 bits sont les suivantes :

- CREER_EVEN

  Un groupe d' événements de 16 bits initialisés à 0 est créé en mémoire, le noyau renvoie le numéro d' identification de ce groupe à la tâche appelante.

- ETAT_EVEN(idf_evt)

  Le noyau renvoie à la tâche appelante les 16 bits correspondant à la valeur du groupe d' événements.

- INIT_EVEN(idf_evt)

  Cette requête permet de remettre à zéro tous les bits d' un groupe d' événements.

- SUP_EVEN(idf_evt, indic)

  Cette requête permet de supprimer un groupe d' événements.

- ATTEND_EVEN(idf_evt, masque, valeur, délai)

  La tâche faisant cette requête est mise en attente d' un ou de plusieurs événements.
  Si le masque vaut 1, l' appel est un AND et tous les bits ou drapeaux doivent être simultanément à 1 pour que la tâche sorte de son état d' attente.
  Si le masque vaut 0, l' appel est un OR et il suffit qu' un seul bit soit à 1 pour que la tâche reprenne sa place dans la file des tâches prêtes.
  La variable délai précise le nombre maximal de ticks durant lequel la tâche est susceptible d' attendre. Passé ce délai, la tâche est réactivée et un code d' erreur renvoyé si l' événement n' a pas été reçu. Si délai vaut zéro, la tâche reste indéfiniment en attente de l' événement.
  La variable valeur précise la valeur attendue pour le groupe.

- ENVOI_EVEN(idf_evt, valeur)

  Une tâche utilise cette requête pour envoyer un groupe d' événements. Toute tâche en attente de cet événement est réactivée.

13 Gestion des tâches immédiates

Les tâches immédiates sont déclenchées par interruption matérielle et sont prioritaires par rapport aux tâches logicielles.
Sur PC, les interruptions matérielles sont gérées par le PIC ou Contrôleur Programmable d' Interruption. Huit lignes d' interruption sont ainsi contrôlées par niveaux de priorité, soit de la priorité la plus haute à la plus basse :

- IRQ 0 ; timer
- IRQ 1 : clavier
- IRQ 2 : réservée
- IRQ 3 : COM 2
- IRQ 4 : COM 1
- IRQ 5 : disque dur
- IRQ 6 : lecteur de disquettes
- IRQ 7 : imprimante

En l' absence de cartes supplémentaires pouvant générer des interruptions que notre noyau devrait alors gérer et hiérarchiser, on se limitera de prime d 'abord au traitement des interruptions horloge et clavier.

Le problème le plus ardu en ce qui concerne le traitement des interruptions est la priorité que l' on doit leur donner par rapport au noyau lui-même.
Etudions, par exemple, le cas d 'une interruption horloge validée alors que le noyau est en cours de traitement d' un service.
Le programme lié à l' interruption horloge est le suivant :

- incrémenter l' horloge temps réel
- décrémenter le compteur des tâches en attente d' un délai ou d' un événement temporisé.
- si le temps cyclique est programmé, décrémenter le compteur de la tâche en cours d' exécution.

Si effectuer ces opérations ne pose aucun problème, le noyau reprenant ensuite le service pour lequel il était requis, peut retrouver un système incohérent par rapport au traitement qu'il était en train d' effectuer (par exemple une commutation de tâche).
La solution retenue est la suivante :

1) Les interruptions seront inhibées lorsque le noyau traitera une rquête.
2) Les requêtes et services seront accessibles dans leur totalité aux tâches immédiates sauf évidemment celui qui consisterait à se mettre soi-même ou une autre tâche immédiate en attente..
3) Le traitement effectué pour une requête peut être différent selon la catégorie de la tâche appelante.
4) Les tâches immédiates sont prioritaires par rapport aux tâches logicielles
5) Le système peut être interrompu par 255 interruptions en cascade. Dans ces conditions, les programmes d' interruption sont imbriquées.

Les primitives spécifiques aux tâches immédiates sont les suivantes :

- APPEL_INT

  Cet appel est obligatoire lorsqu'une tâche immédiate commence son traitement. Il signale au noyau que tous les appels ultérieurs seront effectués pour le compte d' une tâche immédiate.

- FIN_INT

  Cet appel est obligatoire lorsqu'une tâche immédiate termine son traitement. Ce n' est que lorsque toutes les ISR ont envoyé cette requête que le système est réordonnancé et le contrôle rendu soit à la tâche logicielle interrompue soit à la nouvelle tâche en tête de la file des tâches prêtes.

- INT_TIMER

  Cet appel est effectué par le gestionnaire d' interruption du timer après APPEL_INT et avant FIN_INT. Des opérations spécifiques sont alors effectuées par le noyau comme incrémenter l' horloge temps réel et décrémenter le compteur des tâches en attente d' un délai.

- INT_CLAVIER

  Cet appel est effectué par le gestionnaire d' interruption du clavier en cas de frappe d' une touche par l' utilisateur. Le caractère correspondant est rangé dans le buffer interne du noyau.

14 Communication avec l' extérieur

Notre système communique avec l' extérieur par l' intermédiaire de l' écran et du clavier.
Une tâche peut demander au noyau d' afficher un caractère ou un message à l' écran.
Elle peut également se mettre en attente d' une touche frappée au clavier. Elle ne reprendra son exécution que lorsque le noyau lui transmettra le caractère correspondant.
Le noyau peut lui-même afficher des messages à l' écran, des messages d' erreur notamment.

Enfin, on prévoit qu' une tâche puisse demander au noyau l' affichage de l' état d' une autre tâche.
La détection de la frappe d' une touche se fait par interruption et l' affichage se fait directement en mémoire vidéo.
Les primitives utilisées sont les suivantes :

- ATTEND_CARACTERE

  Cette requête permet à une tâche de se mettre en attente d' une touche frappée au clavier. Lorsqu'une interruption clavier signale au noyau la frappe d' une touche, il range le caractère reçu dans un buffer interne et teste la file FIFO des tâches en attente.
  Le caractère est transmis à la tâche en tête de file s' il en existe sinon la prochaine tâche se mettant en attente d' un caractère recevra ce dernier.

- AFFICHE_STATUS(idf_tache)

  Cet appel permet d' afficher dans une fenêtre en bas de l' écran l' état de la tâche dont le numéro est donné en paramètre.

- AFFICHE_CAR (caractère, couleur, ligne, colonne)

  Cet appel permet d' afficher un caractère à l' emplacement donné sur l' écran et avec la couleur spécifiée.

15 Plan qualité

Notre travail étant réalisé dans le cadre d' un cours de génie logiciel, il se doit de vérifier un certain nombre de critères qualité notamment en ce qui concerne la documentation et l' étude préliminaire. Par raison de temps, on élimine tout ce qui concerne les plans de test et de validation. Notre noyau n' est d' ailleurs pas destiné à être industrialisé ni homologué.
La cohérence du système doit être ici posée et pensée dès le départ. Nous avons donc retenu les critères suivants :

- **Complétude** : tous les cas possibles d' erreur doivent être envisagés au niveau de la vérification des requêtes et paramètres que les tâches applicatives adressent au noyau. Un noyau robuste facilitera la programmation des tâches et évitera tout plantage du système par ces tâches.

- **Cohérence** : la cohérence concerne ici la possibilité ou l' impossibilité d' exécuter une requête. On ne peut pas par exemple, mettre une tâche immédiate en attente ou mettre en attente d' un événement une tâche qui serait déjà en attente sur un sémaphore.

- **Extensibilité :** on doit pouvoir éventuellement rajouter des modules au programme sans modifier ceux existant.

- **Intégrité des données ;** ce problème est critique dans notre cas en raison des nombreuses possibilités de commutation et de changement de contexte.

- **Lisibilité :** le problème est important car un programme réalisé en assembleur est généralement peu lisible. La modularité, les commentaires et l' utilisation de macro-instructions peut remédier en partie à ce problème.

- **Evolution :** le programme est conçu de telle façon, notamment au niveau des registres utilisés pour les requêtes qu ' il est possible de construire une librairie d'interface en langage C. Ainsi, on pourra programmer les tâches applicatives en C.

- **Documentation** ; l' utilisateur pourra consulter utilement le manuel utilisateur. Le concepteur pourra se référer aux dossiers établis en cours de réalisation soit :
  1) Le cahier des charges ci-présent fixant les objectifs et les contraintes.
  2) Le dossier de spécifications établi suivant la méthodologie SART. Ce dossier, établi en première analyse, est principalement un exercice de style visant à mettre en œuvre la méthode SART qui sera par contre parfaitement adaptée pour l' analyse des tâches applicatives.
  3) Le dossier de conception globale établi suivant la méthode MACH. Ce dossier fixe l' architecture générale du logiciel, les différents modules et présente les algorithmes fondamentaux.
  4) Le dossier de conception détaillée contient la description précise des modules, des procédures et des structures de données du logiciel. Il décrit un élément fondamental qui n' est autre que la gestion de la pile système.