(19) RÉPUBLIQUE FRANÇAISE

INSTITUT NATIONAL DE LA PROPRIÉTÉ INDUSTRIELLE

COURBEVOIE

(11) **N° de publication :** **3 140 969**

(à n'utiliser que pour les commandes de reproduction)

(21) **N° d'enregistrement national :** **22 10701**

(51) Int Cl[8] : ***G 06 F 9/44*** *(2023.01), G 06 F 9/46*

(12) DEMANDE DE BREVET D'INVENTION A1

(22) **Date de dépôt :** 17.10.22.

(30) **Priorité :**

(43) **Date de mise à la disposition du public de la demande :** 19.04.24 Bulletin 24/16.

(56) **Liste des documents cités dans le rapport de recherche préliminaire :** *Se reporter à la fin du présent fascicule*

(60) **Références à d'autres documents nationaux apparentés :**

( ) **Demande(s) d'extension :**

(71) **Demandeur(s) :** *Commissariat à l'énergie atomique et aux énergies alternatives Etablissement public à caractère industriel et commercial* — FR.

(72) **Inventeur(s) :** TORTECH THIBAUD, AIT HMID MOHA et FAURE CYRIL.

(73) **Titulaire(s) :** Commissariat à l'énergie atomique et aux énergies alternatives Etablissement public à caractère industriel et commercial.

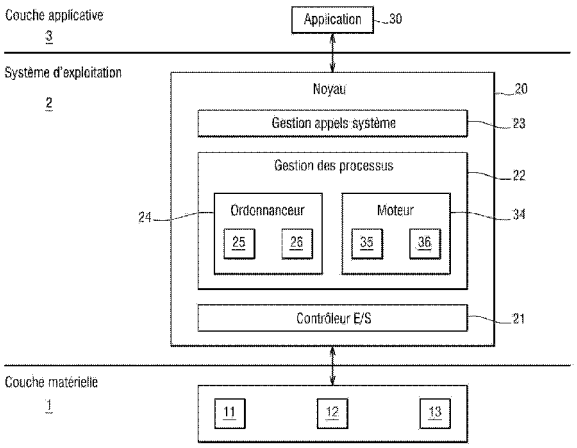
(74) **Mandataire(s) :** Lavoix.

(54) Procédé de gestion d'un appel système et produit programme d'ordinateur associé.

(57) Procédé de gestion d'un appel système et produit programme d'ordinateur associé

Ce procédé, réalisé par un noyau (20) muni d'un moteur (34) consiste: détermination si un appel système d'un fil d'exécution applicatif est bloquant ; dans l'affirmative, mise en sommeil du fil d'exécution; création d'une tâche asynchrone pour réaliser l'appel système et mise à jour d'une liste de tâches; exécution par le moteur de la tâche sur une pile du noyau avec des informations relatives à la tâche stockées dans une zone mémoire, jusqu'à atteindre un état d'exécution bloquant nécessitant un évènement attendu; lors de la survenue de l'évènement attendu, reprise de l'exécution de la tâche; vérification si l'état d'exécution atteint est un état d'exécution final de la tâche ; et, dans l'affirmative, finalisation de l'exécution de la tâche et suppression de la tâche de la liste de tâches; et retour de l'appel système au fil d'exécution.

Figure pour l'abrégé: Figure 1



FR 3 140 969 - A1

Description

Titre de l'invention : Procédé de gestion d'un appel système et produit programme d'ordinateur associé

[0001] L'invention a pour domaine celui de la programmation système.

[0002] Le système d'exploitation (« operating system » en anglais) d'un ordinateur est un programme d'abstraction de la couche matérielle, jouant le rôle d'interface entre la couche applicative et la couche matérielle.

[0003] Une application est un programme exécuté « au-dessus » du système d'exploitation et qui accède par conséquent à la couche matérielle indirectement via le système d'exploitation.

[0004] Le système d'exploitation comporte un noyau et éventuellement d'autres programmes associés.

[0005] Le noyau (« kernel ») est une partie du système d'exploitation qui répond aux appels système, aux interruptions et aux exceptions.

[0006] Une application en cours d'exécution est un processus (« process »).

[0007] Un processus est subdivisé en une pluralité de fils d'exécution (« thread »). Un fil d'exécution comporte un bloc d'instructions élémentaires exécutable en une seule fois par le processeur de l'ordinateur.

[0008] Un fil d'exécution peut être dans différents états d'exécution : un état « running » (ou « en cours d'exécution »), un état « ready » (ou « prêt à être exécuté ») ou un état « waiting » (ou « en attente »).

[0009] Un fil d'exécution est associé à un contexte d'exécution. C'est l'ensemble des informations décrivant l'état d'exécution du fil d'exécution et permettant au processeur de poursuivre, d'interrompre ou de reprendre l'exécution de ce fil d'exécution. Le contexte comporte notamment une pile d'exécution (« stack ») dans la mémoire de l'ordinateur, le contenu des registres du processeur (comme les valeurs d'un compteur de programme, un pointeur de pile,...).

[0010] Les instructions sont exécutées par le processeur dans un mode prédéfini de fonctionnement du processeur, en fonction des privilèges octroyés à ces instructions, notamment d'accès à certains espaces de la mémoire. Par exemple, les instructions d'un fil d'exécution d'une application sont exécutés dans un mode non-privilégié, par exemple le mode « utilisateur », alors que les instructions du système d'exploitation (notamment le noyau) sont exécutées dans un mode privilégié, comme le mode « noyau ».

[0011] Dans un système d'exploitation multi-tâches, plusieurs processus sont exécutés en « parallèle ». Plus exactement, un ordonnanceur (« scheduler ») du noyau place les fils

d'exécution qui sont dans l'état « ready » dans une file d'attente (« queue »), qui est une liste ordonnée des fils d'exécution. L'ordonnanceur utilise des règles de priorité pour ordonner les fils d'exécution dans cette file d'attente. Le processeur exécute le fil d'exécution placé au bas de la file d'attente. Celui-ci bascule alors de l'état « ready » à l'état « running ». Le processeur exécute ce fil d'exécution soit jusqu'à atteindre la fin de l'exécution des instructions de ce fil d'exécution, soit lorsqu'un temps d'exécution maximal pour un file d'exécution est atteint.

[0012] Lorsque le processeur passe de l'exécution d'un premier fil d'exécution d'un premier processus à l'exécution d'un second fil d'exécution d'un second processus, il effectue un changement de contexte. Il y a changement de contexte même si les deux fils d'exécution appartiennent à un même processus. Tout au plus certaines informations communes aux deux fils d'exécution, comme l'espace d'adressage, les données de protections mémoire, etc. peuvent ne pas être modifiées. Mais les informations spécifiques à chaque fil d'exécution, comme le compteur de programme, le pointeur de pile, etc. doivent être modifiées.

[0013] Lors d'un changement de contexte, le noyau est responsable de sauvegarder le contexte du premier fil d'exécution et de restaurer le contexte du second fil d'exécution.

[0014] Pour la réalisation de certaines fonctions dites systèmes, un fil d'exécution fait appel aux services correspondant fournis par le noyau. On dit que le fil d'exécution fait un appel système. Pour cela les étapes suivantes sont successivement réalisées :

[0015] - Le processeur exécute les instructions du fil d'exécution dans le mode non privilégié sur une pile d'exécution non-privilégiée du fil d'exécution.

[0016] - Lorsqu'une de ces instructions correspond à un appel système, le processeur bascule dans du mode non-privilégié vers le mode privilégié, et exécute les instructions du noyau associées à la fonction système appelée sur une pile d'exécution privilégiée.

[0017] - Une fois les instructions associées à la fonction système exécutées, le noyau retourne l'appel système en indiquant le résultat de l'appel système.

[0018] - Le processeur bascule dans du mode privilégié vers le mode non-privilégié, et reprend l'exécution des instructions du fil d'exécution appelant sur la pile d'exécution non-privilégiée, en tenant compte du résultat de l'appel système.

[0019] L'exécution du fil d'exécution appelant reprend par conséquent là où elle avait été suspendue suite à l'appel système.

[0020] Il existe cependant deux types d'appels système, les appels système non-bloquants et les appels système bloquants.

[0021] Dans un appel système non-bloquant, le noyau peut réaliser la tâche correspondant au service demandé, car il dispose des informations requises. Le noyau retourne alors immédiatement l'appel système. C'est ce qui a été présenté ci-dessus.

[0022] Dans un appel système bloquant, le noyau ne peut pas réaliser la tâche correspondant au service demandé, car il n'a pas toutes les informations nécessaires. On est donc dans une situation où le fil d'exécution est bloqué tant que les informations requises ne sont pas disponibles.

[0023] C'est par exemple le cas pour un appel système de lecture dans un tuyau de données (« pipe ») lorsque ce tuyau est vide. Le fil d'exécution est alors bloqué en lecture sur le tuyau. Le noyau retournera l'appel système uniquement lorsqu'une donnée sera disponible dans le tuyau surveillé, c'est-à-dire lors de la survenu d'un évènement du type « écriture » d'une donnée dans le tuyau surveillé.

[0024] C'est par exemple encore le cas d'un appel système « sleep » consistant à attendre qu'un compteur de temps atteigne une certaine durée prédéfinie. Le fil d'exécution est alors bloqué tant que le compteur de temps n'a pas atteint la valeur requise. Le noyau retournera l'appel système uniquement lorsque le compteur de temps aura atteint la durée prédéfinie, c'est-à-dire lors de la survenu d'un évènement du type « égalité » entre la valeur courante du compteur de temps et la durée prédéfinie.

[0025] La gestion d'un tel blocage du fil d'exécution, ainsi que la reprise de son exécution sont entièrement à la charge du noyau.

[0026] Notamment, un fil d'exécution bloqué par un appel système bloquant est mis en attente par le noyau. Plus précisément, il est basculé de l'état « running » à l'état « waiting » et est retiré de la file d'attente des fils d'exécution à l'état « ready » de l'ordonnanceur pour qu'il ne puisse pas être exécuté. Son contexte d'exécution est sauvegardé (notamment la pile non-privilégiée sur laquelle il était exécuté) pour permettre au processeur de passer à l'exécution d'un autre fil d'exécution.

[0027] Pour la reprise de l'exécution d'un fil d'exécution dans l'état « waiting », le noyau tient à jour une table indiquant tous les fils d'exécution dans l'état « waiting » et, pour chaque fil d'exécution, l'évènement nécessaires à sa reprise.

[0028] Lors de la survenue de l'évènement attendu pour la reprise d'un fil d'exécution, le noyau reprend et termine l'exécution des instructions de l'appel système bloquant. Il met à jour le contexte du fil d'exécution avec le résultat de l'appel système. Il bascule enfin le fil d'exécution de l'état « waiting » à l'état « ready ».

[0029] Le fil d'exécution est alors replacé dans la file d'attente par l'ordonnanceur en vue d'être à nouveau exécuté par le processeur.

[0030] Actuellement, il existe deux modèles de noyau en fonction de la manière précise dont le noyau gère des appels systèmes bloquants : le noyau « process model » (ou noyau PM) d'une part et le noyau « IRQ Model » (ou noyau IRQ) d'autre part.

[0031] Le noyau PM est fondé sur la présence de deux piles pour chaque fil d'exécution : une pile en mode non-privilégié (ou pile non-privilégiée) et une pile en mode privilégié (ou pile privilégiée). Si un fil d'exécution est bloqué, le contexte noyau du fil

d'exécution est stocké dans la pile privilégiée du fil d'exécution, au moment où le fil d'exécution bascule de l'état « running » à l'état « waiting ».

[0032] Lorsque l'évènement nécessaire à la reprise de ce fil d'exécution survient, le noyau utilise informations de la pile privilégiée du fil d'exécution pour finaliser l'appel système. La finalisation de cette tâche conduit automatiquement à une reprise de l'exécution du fil d'exécution appelant en le basculant de l'état « waiting » vers l'état « ready ».

[0033] Cependant, dans ce premier modèle, le dimensionnement de la pile privilégiée d'un fil d'exécution est problématique. Il y a un risque élevé de débordement (« overflow ») sur d'autres cellules de la mémoire du noyau. Ceci soulève des questions majeures de sécurité et de fiabilité. Pour y parer, il y a une tendance à surdimensionner la pile privilégiée pour chaque fil d'exécution. Ceci conduit à une consommation importante des ressources de l'ordinateur.

[0034] Mais, le code du noyau reste simple à développer et à comprendre par son développeur, car son aspect séquentiel est conservé. La sauvegarde du contexte noyau du fil d'exécution dans la pile privilégiée est transparente au développeur car gérée par le compilateur.

[0035] Le noyau IRQ est fondé sur une pile non-privilégiée par fil d'exécution et une pile privilégiée commune à tous les fils d'exécution. La pile privilégiée commune est donc partagée entre les différents fils d'exécution.

[0036] La pile privilégiée commune est cependant réinitialisée à l'issue de chaque appel système (bloquant ou non-bloquant), c'est-à-dire qu'aucune information n'est mémorisée sur la pile privilégiée commune.

[0037] C'est donc au développeur du code du noyau de coder, pour les appels systèmes bloquants, la manière de stocker le contexte noyau du fil d'exécution dans des structures de données annexes, avec les conditions de déblocage, pour permettre la finalisation de l'appel système lors de la survenue de l'évènement attendu et la reprise du fil d'exécution appelant mis en sommeil.

[0038] Ainsi, la partie du code source d'un noyau IRQ gérant la sauvegarde du contexte noyau est laissée aux soins du développeur.

[0039] Si un tel noyau permet d'avoir une approche sécuritaire, la lecture du code source n'est plus séquentielle. Le code source est donc particulièrement difficile à lire et à vérifier.

[0040] Le but de la présente invention est de proposer un modèle de noyau alternatif aux modèles de noyau de l'état de la technique, offrant une nouvelle approche pour la gestion des appels systèmes bloquants, tout en essayant de combiner les avantages des deux modèles connus et en évitant leurs inconvénients respectifs.

[0041] Pour cela l'invention a pour objet un procédé de gestion d'un appel système, mis en

œuvre par un noyau d'un système d'exploitation exécuté par un ordinateur, l'appel système ayant été émis par un fil d'exécution d'une application exécuté par l'ordinateur, caractérisé en ce que, ledit noyau comportant un ordonnanceur et un moteur d'exécution de tâches asynchrones, ledit procédé comporte les étapes de : a)- détermination par le noyau si l'appel système est un appel système bloquant ; et, dans l'affirmative : b)- mise en sommeil par le noyau du fil d'exécution ; c)- création par le noyau d'une tâche asynchrone pour réaliser l'appel système, la tâche asynchrone étant équivalente une machine à états, une transition d'un premier état d'exécution vers un second état d'exécution étant initiée par un évènement attendu, et mise à jour par le noyau d'une liste de tâches asynchrones à exécuter par le moteur avec les informations relatives à la tâche asynchrone créée ; d)- exécution par le moteur de la tâche asynchrone sur une pile du noyau en utilisant les informations de la zone mémoire relatives à la tâche asynchrone, jusqu'à atteindre un état d'exécution bloquant nécessitant un évènement attendu correspondant, l'exécution de la tâche asynchrone étant alors bloquée et la liste de tâches asynchrones mise à jour avec l'état d'exécution bloquant atteint et l'évènement attendu correspondant ; e)- lors de la survenue de l'évènement attendu correspondant, reprise de l'exécution par le moteur de la tâche asynchrone sur la pile du noyau de manière à effectuer la transition de l'état d'exécution bloquant vers un état d'exécution bloquant suivant ; f)- vérification si l'état d'exécution bloquant suivant est un état d'exécution final de la tâche ; et dans l'affirmative, g)- finalisation de l'exécution de la tâche asynchrone et suppression de la tâche asynchrone de la liste de tâches asynchrones à exécuter par le moteur ; et, h)- retour par le noyau de l'appel système au fil d'exécution et réveil par le noyau du fil d'exécution.

[0042] Suivant des modes particuliers de réalisation, … comporte une ou plusieurs des caractéristiques suivantes, prises isolément ou suivant toutes les combinaisons techniquement possibles :

[0043] - lorsque l'étape de vérification conduit à conclure que l'état d'exécution bloquant suivant n'est pas l'état d'exécution final de la tâche asynchrone, les étapes d), e) et f) sont itérées.

[0044] - le procédé comporte, en outre, lorsque le noyau détermine que l'appel système reçu est un appel système non-bloquant, les étapes de : exécution par le noyau de l'appel système sur une pile du noyau ; et, retour par le noyau de l'appel système au fil d'exécution.

[0045] - la mise en sommeil par le noyau du fil d'exécution consiste à basculer un état d'exécution du fil d'exécution d'un état « en cours d'exécution» à un état « en attente » et le réveil par le noyau du fil d'exécution consiste à basculer l'état d'exécution du fil d'exécution de l'état « en attente» à un état «prêt à être exécuté».

[0046] - entre la mise en œuvre des étapes d) et e), un autre fil d'exécution présent dans une file d'attente de l'ordonnanceur est exécuté.

[0047] - lors d'une interruption indicative de la survenue d'un nouvel évènement, le moteur est exécuté pour identifier la tâche asynchrone de la liste des tâches asynchrones pour laquelle le nouvel évènement constitue l'évènement attendu.

[0048] L'invention a également pour objet un produit programme d'ordinateur comportant des instructions qui, lorsqu'elles sont exécutées par un processeur d'un ordinateur, munissent ledit ordinateur d'un noyau, caractérisé en ce que ledit noyau comporte un ordonnanceur et un moteur d'exécution de tâches asynchrones, ledit moteur de tâches asynchrones étant associé à une liste de tâches asynchrones à exécuter et une zone mémoire dédiée dans une mémoire de l'ordinateur, le noyau étant adapté, lors de la réception d'un appel système émis par un fil d'exécution d'une application exécutée par l'ordinateur, pour réaliser un procédé de gestion dudit appel système conforme au préocédé précédent.

[0049] **De préférence, le p**roduit programme d'ordinateur est tel que le noyau est associé à une unique pile noyau.

[0050] De préférence encore, le produit programme d'ordinateur résulte de la compilation d'un code source, ledit code source se présentant sous une forme séquentielle même pour les appels systèmes bloquants.

[0051] L'invention et ses avantages seront mieux compris à la lecture de la description détaillée qui va suivre d'un mode de réalisation particulier, donné uniquement à titre d'exemple non limitatif, cette description étant faite en se référant aux dessins annexés sur lesquels :

[0052] [Fig.1] La [Fig.1] est une représentation schématique sous forme de modules fonctionnels d'un mode de réalisation préféré d'un noyau selon l'invention ;

[0053] [Fig.2] La [Fig.2] est une représentation schématique sous forme de blocs des étapes d'un appel système bloquant en exécutant le noyau de la [Fig.1] ; et,

[0054] [Fig.3] La [Fig.3] est une représentation schématique du principe de fonctionnement du noyau de la [Fig.1].

[0055] La présente invention porte sur un produit programme d'ordinateur qui, lorsque ses instructions sont exécutées par un ordinateur, fournit une fonctionnalité de noyau à cet ordinateur. Dans ce qui suit on parlera plus simplement de noyau.

[0056] En tant que programme d'ordinateur exécutable, le noyau est un code objet résultant de la compilation d'un code source, écrit dans un langage de programmation adapté.

[0057] De manière générale, le noyau selon l'invention gère un blocage lié à un appel système bloquant au moyen d'une programmation asynchrone des appels systèmes bloquants.

[0058] La [Fig.1] représente un système d'exploitation 2 formant interface entre une couche

matérielle 1 (processeur 11, mémoire 12, ports d'entrée/sortie 13, ...) et une couche applicative 3, comportant un ou plusieurs programme(s) applicatif(s) (ou application(s)), tels que l'application 30.

[0059] Le système d'exploitation 2 comporte, entre autres choses, un noyau 20.

[0060] De manière classique, le noyau 20 comporte, entre autres choses, un module de contrôle d'entrée/sortie 21, un module de gestion des processus 22 et un module de gestion des appels système 23.

[0061] Le module de gestion des processus 22 comporte, entre autres choses, un ordonnanceur 24. Celui-ci est associé à une liste des fils d'exécution 25 et à une file d'attente 26.

[0062] Selon l'invention, le module de gestion des processus 22 du noyau comporte, en outre, un moteur d'exécution de tâches asynchrones 34.

[0063] La réalisation par le moteur d'une tâche asynchrone doit être vue comme une machine à états, où un évènement cause une transition d'un état initial vers un état final de l'ensemble des états possibles de la tâche asynchrone, l'état final étant l'état « tâche terminée ».

[0064] Le moteur 34 est associé à une liste de tâches asynchrones 35 et à des zones mémoire 36 dédiées.

[0065] La liste de tâches asynchrones 35 indique l'ensemble des tâches asynchrones en cours d'exécution, et, pour chaque tâche, l'état d'exécution courant dans laquelle se trouve cette tâche, et l'évènement attendu pour effectuer une transition de l'état courant vers un état suivant.

[0066] Le moteur 34 ne gère pas une pile, mais une zone mémoire 36 par tâche asynchrone à réaliser. Cette zone mémoire permet au moteur de stocker les valeurs des variables de la tâche asynchrone en cours de réalisation. La zone mémoire associée à une tâche asynchrone est de taille définie.

[0067] Pour des raisons de clarté, sur la [Fig.1], la liste 25, la file 26, la liste 35 et des zones mémoires 36 sont représentées dans les modules 24 et 34, mais il s'agit bien de structures dans la mémoire de l'ordinateur.

[0068] La manière de programmer un appel système bloquant dans le noyau selon l'invention est donnée dans la tableau suivant relatif à la partie du code source du noyau pour l'exemple de l'appel système « sys_sleep » en utilisant le langage de programmation Rust (« marque déposée »):

[0069] [Tableaux1]

```
fn sys_sleep<T: ContextTrait>(ctx:&'static mut T){
// Entrées de l'appel système. Les paramètres d'entrée sont décodés.
let now =crate::timer::GTimer::now();
let duration ctx.syscall_in::<sys::Sleep>().duration;
let alarm = now. duration;
// Blocage du thread en cours d'exécution.
let kernel = Sefl::kernel_mut();
let th = kernel.threads.block_running();
let tid = th.read().id();
// Lancer une tâche asynchrone pour gérer l'appel système bloquant. Le code suivant
sera exécuté par le moteur.
crate::tasks::spawn( tid,asyncmove{
kernel.delay.deadline( alarm ).await;
kernel.threads.unblock( th );
let res =crate::timer::GTimer::now()- now;
ctx::syscall_out::<sys::Sleep>(&res );
});
}
```

[0070] Dans cet exemple, l'appel système « sys_sleep » a pour but de mettre en attente un fil d'exécution appelant pendant un temps donné.

[0071] Cette fonction contient le blocage du fil d'exécution en cours (instruction « kernel.threads.block_running() »).

[0072] Cette fonction contient ensuite la fonction création d'une tâche asynchrone (« crate::tasks::spawn »). L'exécution de cette fonction conduit à la création d'une tâche asynchrone (instruction « async move »).

[0073] La tâche asynchrone est exécutée par la moteur 34.

[0074] C'est dans cette tâche asynchrone que se trouve le détail des instructions de l'appel système en tant tel.

[0075] Elle comporte les évènements attendus pour faire avancer l'exécution de ces instructions, comme par exemple ici l'attente de la survenue du temps écoulé (instruction « .await »).

[0076] La tâche asynchrone comporte également les instructions à exécuter, comme par exemple ici les instructions pour finaliser la tâche asynchrone lorsque l'évènement attendu est arrivé (« res = crate::timer::GTimer::now() – now »).

[0077] La tâche asynchrone comporte finalement le retour de la fonction système (« ctx::syscall_out::<sys::Sleep>(&res) »)

[0078] Ainsi, dans le code source du tableau ci-dessus, qui est une expression extrêmement synthétique des instructions du noyau lors d'un appel système bloquant, on constate qu'une tâche asynchrone comporte, en plus des instructions de l'appel système en tant que tel (c'est-à-dire le code du service rendu par le noyau), la machine à états caractérisant l'avancement de l'exécution de la tâche asynchrone, ainsi que la gestion du réveil du fil d'exécution appelant.

[0079] Comme illustré sur la [Fig.2], le procédé 100 de gestion d'un appel système par le noyau 20 s'effectue de la manière suivante.

[0080] Dans une étape 110, alors qu'un fil d'exécution, par exemple d'un processus de l'application 30, est en cours d'exécution par le processeur dans le mode non-privilégié sur une pile non-privilégiée dédiée au fil d'exécution, ce dernier fait un appel système, cet appel système étant identifié par un identifiant.

[0081] Dans une étape 120, du fait de cet appel système, le processeur 11 bascule du mode non-privilégié vers le mode privilégié pour la réalisation par le noyau 22 des instructions de l'appel système. La sauvegarde du contexte non-privilégié du fil d'exécution est réalisée dans la pile non-privilégiée du fil d'exécution.

[0082] Dans l'étape 130, le module de gestion des appels système 23 récupère l'identifiant de l'appel système et détermine, sur la base de cet identifiant, s'il s'agit d'un appel système du type non-bloquant ou d'un appel système du type bloquant.

[0083] Dans le premier cas, l'appel système non-bloquant est géré de manière classique, c'est-à-dire que les instructions de l'appel système sont exécutées par le processeur sur la pile privilégiée unique du noyau et retourne l'appel système au fil d'exécution appelant (étape 140).

[0084] Du fait de ce retour à l'appel système, le processeur bascule du mode privilégié vers le mode non-privilégié (étape 150) pour poursuivre l'exécution des instructions du fil d'exécution appelant (étape 160), en tenant compte du résultat de l'appel système et des informations présentes dans la pile non-privilégiée du fil d'exécution.

[0085] Dans le second cas, dans l'étape 170, sur requête du module 23, l'ordonnanceur 24 place le fil d'exécution appelant dans l'état « waiting » dans la liste 25. Il enlève le fil d'exécution appelant de la file d'attente 26.

[0086] Parallèlement, dans l'étape 180, le module 23 crée une tâche asynchrone pour gérer cette mise en attente du fil d'exécution appelant. La création d'une tâche asynchrone consiste à définir la machine à états associée à cette tâche asynchrone et à enregistrer une nouvelle entrée dans la liste des tâches asynchrone 35, définissant l'état courant de la tâche asynchrone et l'évènement attendu pour faire évoluer cet état. Une zone mémoire 36 est réservée pour la tâche asynchrone.

[0087] Puis, dans l'étape 190, le moteur est exécuté. Il exécute le contenu de la section « async» de la tâche asynchrone sur la pile noyau en utilisant les informations né-

cessaires à l'exécution de la tâche asynchrone stockées dans la zone de la mémoire de l'ordinateur dédiée au moteur.

[0088] L'exécution de la tâche avance jusqu'à un état d'exécution bloquant nécessitant la survenue d'un évènement attendu.

[0089] Dans l'exemple précédent, ceci correspond à l'exécution de la tâche jusqu'à l'instruction « .await » (cf. tableau ci-dessous). L'instruction « .await » indique qu'un évènement est attendu.

[0090] S'il n'est pas encore survenu, l'exécution de la tâche asynchrone est bloquée. Le moteur met à jour la liste 35 en indiquant l'état courant de la tâche et l'évènement attendu pour pouvoir reprendre l'exécution de la tâche.

[0091] Le processeur peut alors exécuter le fil d'exécution suivant dans la file d'attente 26 ou d'autres tâches du noyau (en utilisant la pile privilégiée).

[0092] En particulier, dans une étape 210, suite à une interruption indicative de la survenue d'un nouvel évènement (étape 200), le moteur est exécuté sur la pile privilégiée du noyau pour parcourir la liste des tâches 35 afin de vérifier si ce nouvel évènement correspond à l'évènement attendu par l'une ou l'autre des tâches asynchrones de la liste 35.

[0093] Lorsque cet évènement correspond à l'évènement attendu par la tâche asynchrone associée à l'appel système de l'étape 110, alors dans l'étape 220, l'exécution de cette tâche asynchrone par le noyau est reprise de manière à réaliser la transition de l'état d'exécution courant à l'état d'exécution suivant. Plus précisément, le moteur poursuit l'exécution du contenu de la section « async» de la tâche asynchrone, jusqu'à atteindre une autre instruction « .await ».

[0094] A l'étape 230, le nouvel état atteint par la tâche asynchrone est testé pour savoir s'il correspond à l'état « tâche terminée ». Dans la négative, les étapes 190 à 230 sont itérées.

[0095] En revanche, si à l'étape 230, il s'avère que l'état de la tâche asynchrone est l'état « tâche terminée », alors, dans l'étape 240, le noyau exécute la fin de la section « async» de la tâche asynchrone de manière à la finaliser. Puis, cette tâche est détruite en la retirant de la liste 35.

[0096] La suppression de la tâche de la liste 35 étant constatée par le module 23, dans l'étape 250, le moteur retourne l'appel système en mettant à jour le contexte d'exécution du fil d'exécution appelant avec le résultat de l'exécution de la tâche asynchrone. Le moteur modifie l'état du fil d'exécution appelant en le basculant vers l'état « Ready ».

[0097] Dans l'étape 260, le fil d'exécution appelant étant dans l'état « Ready », l'ordonnanceur peut le replacer dans la liste d'attente 36 pour poursuivre l'exécution de ce fil d'exécution par le processeur dans le mode non-privilégié en tenant compte

des informations présentes dans la pile non-privilégiée de ce fil d'exécution.

[0098] Ainsi, les fils d'exécution bloqués sont gérés par le moteur d'exécution de tâches asynchrones. L'ordonnanceur du noyau gère l'exécution des fils d'exécution non bloqués. Les fils d'exécution bloqués ne sont pas de son ressort.

[0099] Ainsi, le moteur, lorsqu'il est appelé par le noyau, gère l'exécution des fils d'exécution bloqués jusqu'à leur déblocage. Le moteur consulte la liste des tâches asynchrones en cours et fait évoluer leur exécution. Si un événement attendu par une tâche asynchrone (suite à un « .await » dans le code du tableau ci-dessus) est survenu, cela permet au moteur de continuer à exécuter les instructions suivantes jusqu'à la fin de la tâche ou jusqu'à ce que la tâche soit de nouveau bloquée (i.e. présence d'un autre « .await « dans le code de la tâche).

[0100] La [Fig.3] illustre le principe de la programmation asynchrone de manière générale,). Il montre l'exécution de deux fonctions asynchrones appelées séquentiellement dans un même fil d'exécution.

[0101] On considère qu'un processus a besoin de deux données, A et B, la donnée A étant obtenue par un premier appel système bloquant à une fonction fA effectué par un premier fil d'exécution du processus et la donnée B étant obtenue par un second appel système bloquant à une fonction fB effectué par un second fil d'exécution du processus.

[0102] Le premier fil d'exécution du processus effectue l'appel système à la fonction fA.

[0103] Le noyau place le premier fil d'exécution en attente (état « waiting ») tout en créant une première tâche asynchrone pour la réalisation par le moteur de la fonction fA.

[0104] L'exécution du processus se poursuit, par exemple par l'élection du second fil d'exécution.

[0105] Le second fil d'exécution du processus effectue l'appel système à la fonction fB.

[0106] Le noyau place le second fil d'exécution en attente (état « waiting ») tout en créant une seconde tâche asynchrone pour la réalisation par le moteur de la fonction fB.

[0107] Le premier fil d'exécution appelant reprend son exécution une fois que la tâche correspondant à l'appel à la fonction fA est terminée.

[0108] Similairement, le second fil d'exécution appelant reprend son exécution une fois que la tâche correspondant à l'appel à la fonction fb est terminée.

[0109] On constate que le processus disposera dans un temps plus court des données A et B que dans le cas par exemple où les deux appels systèmes sont gérés séquentiellement par le noyau.

[0110] Un parallèle peut être dressé entre fil d'exécution / noyau selon la présente invention et utilisateur / serveur Web dans le domaine des communications internet. Un serveur Web traite en effet les requêtes de différents utilisateurs de manière asynchrone afin d'optimiser le nombre de connexions à chaque instant.

[0111] Les avantages du noyau selon l'invention sont nombreux :

[0112] - la programmation asynchrone peut être vue comme des tâches coopératives qui se partagent un même contexte d'exécution, notamment une pile privilégiée commune.

[0113] - un gain en sureté puisque la gestion est réalisée automatiquement par le langage et le compilateur associé et non pas par le programmeur, réduisant ainsi le risque d'erreur. Il n'y a plus de problème de dimensionnement de la taille de la pile en mode privilégié. La zone mémoire est de taille prédéfinie et adaptée à la réalisation de la tâche asynchrone correspondante. Le noyau selon l'invention est donc synonyme d'un haut niveau de confiance.

[0114] - un coût de développement réduit puisque le code source du noyau selon l'invention conserve l'aspect séquentiel (comme l'illustre le tableau ci-dessus), ce qui permet au programmeur de le lire facilement et de le mettre à jour aisément, participant à une réduction du coût de développement. De plus la gestion de l'état d'avancement des fils d'exécution bloqués est faite, non pas manuellement par le développeur, mais par le langage et le compilateur.

[0115] - un gain en performance du fait de l'utilisation minimale de ressources mémoire (une seule pile privilégié) et l'utilisation optimale du CPU (pas de changement intempestif du contexte).

[0116] - Tout le noyau (y compris le moteur d'exécution des taches asynchrones) utilise une seule et même pile privilégiée. Mais l'état des taches asynchrones est sauvegardé, non pas dans une pile dédiée (« Process Model ») ou dans des structures manuellement implémentées pas le développeur (« IRQ model ») mais dans une mémoire dédiée, qui est gérée automatiquement.

[0117] La seule contrainte de la mise en œuvre de la présente invention est de disposer d'un compilateur supportant la programmation asynchrone pour pouvoir compiler le code source de manière à obtenir un code objet exécutable par un ordinateur.

Revendications

[Revendication 1] Procédé (100) de gestion d'un appel système, mis en œuvre par un noyau (20) d'un système d'exploitation (2) exécuté par un ordinateur, l'appel système ayant été émis par un fil d'exécution d'une application (30) exécuté par l'ordinateur, caractérisé en ce que, ledit noyau comportant un ordonnanceur (24) et un moteur d'exécution de tâches asynchrones (34), ledit procédé comporte les étapes de :

a)- détermination (130) par le noyau si l'appel système est un appel système bloquant ; et, dans l'affirmative :

b)- mise en sommeil (170) par le noyau du fil d'exécution ;

c)- création (180) par le noyau d'une tâche asynchrone pour réaliser l'appel système, la tâche asynchrone étant équivalente une machine à états, une transition d'un premier état d'exécution vers un second état d'exécution étant initiée par un évènement attendu, et mise à jour par le noyau d'une liste de tâches asynchrones à exécuter par le moteur avec les informations relatives à la tâche asynchrone créée ;

d)- exécution (190) par le moteur de la tâche asynchrone sur une pile du noyau en utilisant les informations de la zone mémoire relatives à la tâche asynchrone, jusqu'à atteindre un état d'exécution bloquant nécessitant un évènement attendu correspondant, l'exécution de la tâche asynchrone étant alors bloquée et la liste de tâches asynchrones mise à jour avec l'état d'exécution bloquant atteint et l'évènement attendu correspondant ;

e)- lors de la survenue (200) de l'évènement attendu correspondant, reprise de l'exécution (220) par le moteur de la tâche asynchrone sur la pile du noyau de manière à effectuer la transition de l'état d'exécution bloquant vers un état d'exécution bloquant suivant ;

f)- vérification (230) si l'état d'exécution bloquant suivant est un état d'exécution final de la tâche ; et dans l'affirmative,

g)- finalisation (240) de l'exécution de la tâche asynchrone et suppression de la tâche asynchrone de la liste de tâches asynchrones à exécuter par le moteur ; et,

h)- retour (250) par le noyau de l'appel système au fil d'exécution et réveil par le noyau du fil d'exécution.

[Revendication 2] Procédé selon la revendication 1, dans lequel, lorsque l'étape de vérification (230) conduit à conclure que l'état d'exécution bloquant suivant n'est pas l'état d'exécution final de la tâche asynchrone, les étapes d), e)

et f) sont itérées.

[Revendication 3] Procédé selon la revendication 1 ou la revendication 2, comportant, lorsque le noyau détermine (130) que l'appel système reçu est un appel système non-bloquant, les étapes de :
- exécution (140) par le noyau de l'appel système sur une pile du noyau ; et,
- retour (150) par le noyau de l'appel système au fil d'exécution.

[Revendication 4] Procédé selon l'une quelconque des revendications précédentes, dans lequel la mise en sommeil par le noyau du fil d'exécution consiste à basculer un état d'exécution du fil d'exécution d'un état « en cours d'exécution» à un état « en attente » et le réveil par le noyau du fil d'exécution consiste à basculer l'état d'exécution du fil d'exécution de l'état « en attente» à un état «prêt à être exécuté».

[Revendication 5] Procédé selon l'une quelconque des revendications précédentes, dans lequel, entre la mise en œuvre des étapes d) et e), un autre fil d'exécution présent dans une file d'attente de l'ordonnanceur est exécuté.

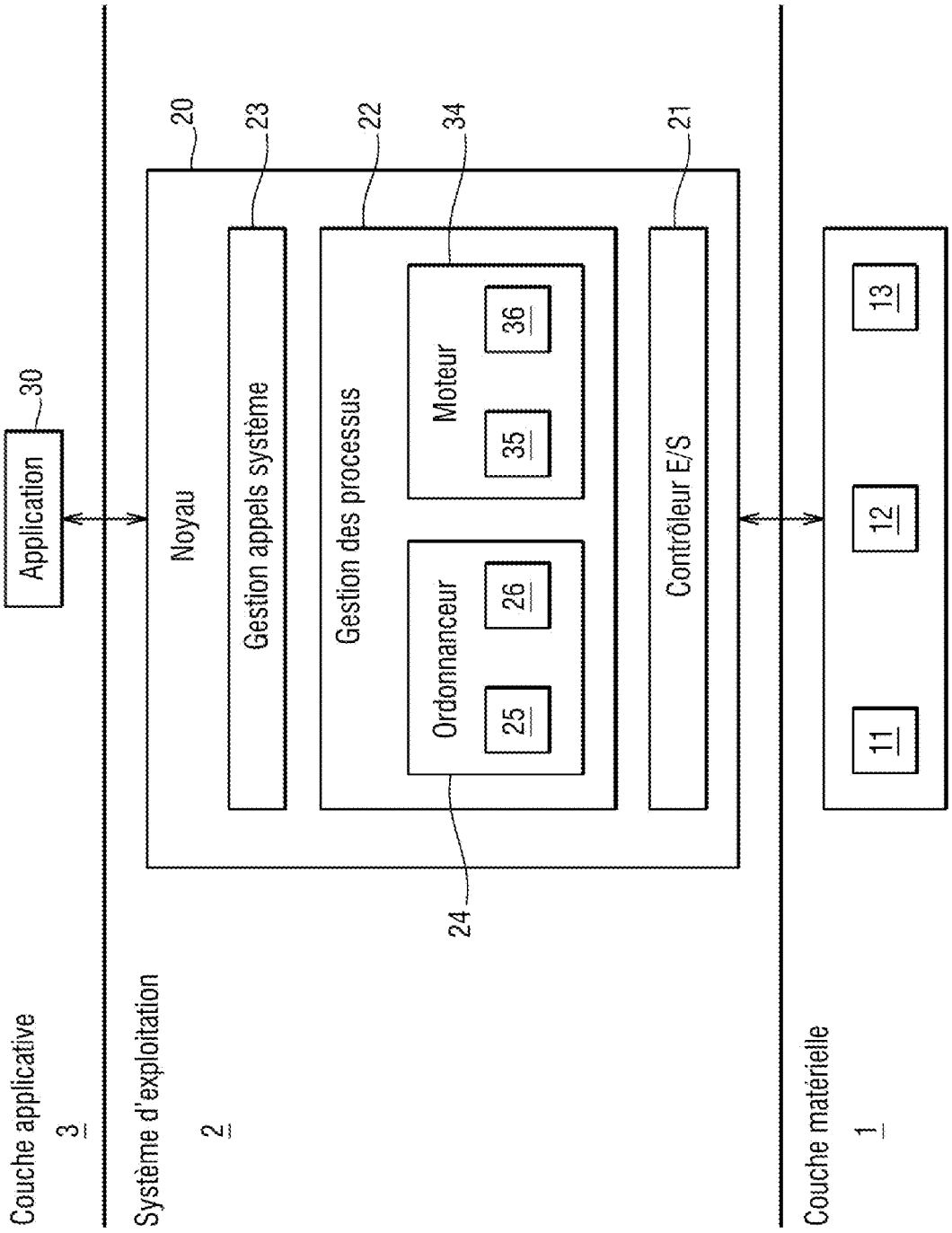
[Revendication 6] Procédé selon l'une quelconque des revendications précédentes, dans lequel, lors d'une interruption indicative de la survenue d'un nouvel évènement, le moteur est exécuté pour identifier la tâche asynchrone de la liste des tâches asynchrones pour laquelle le nouvel évènement constitue l'évènement attendu.

[Revendication 7] Produit programme d'ordinateur comportant des instructions qui, lorsqu'elles sont exécutées par un processeur d'un ordinateur, munissent ledit ordinateur d'un noyau, caractérisé en ce que ledit noyau comporte un ordonnanceur et un moteur d'exécution de tâches asynchrones, ledit moteur de tâches asynchrones étant associé à une liste de tâches asynchrones à exécuter et une zone mémoire dédiée dans une mémoire de l'ordinateur, le noyau étant adapté, lors de la réception d'un appel système émis par un fil d'exécution d'une application exécutée par l'ordinateur, pour réaliser un procédé de gestion dudit appel système conforme à l'une quelconque des revendications 1 à 6.

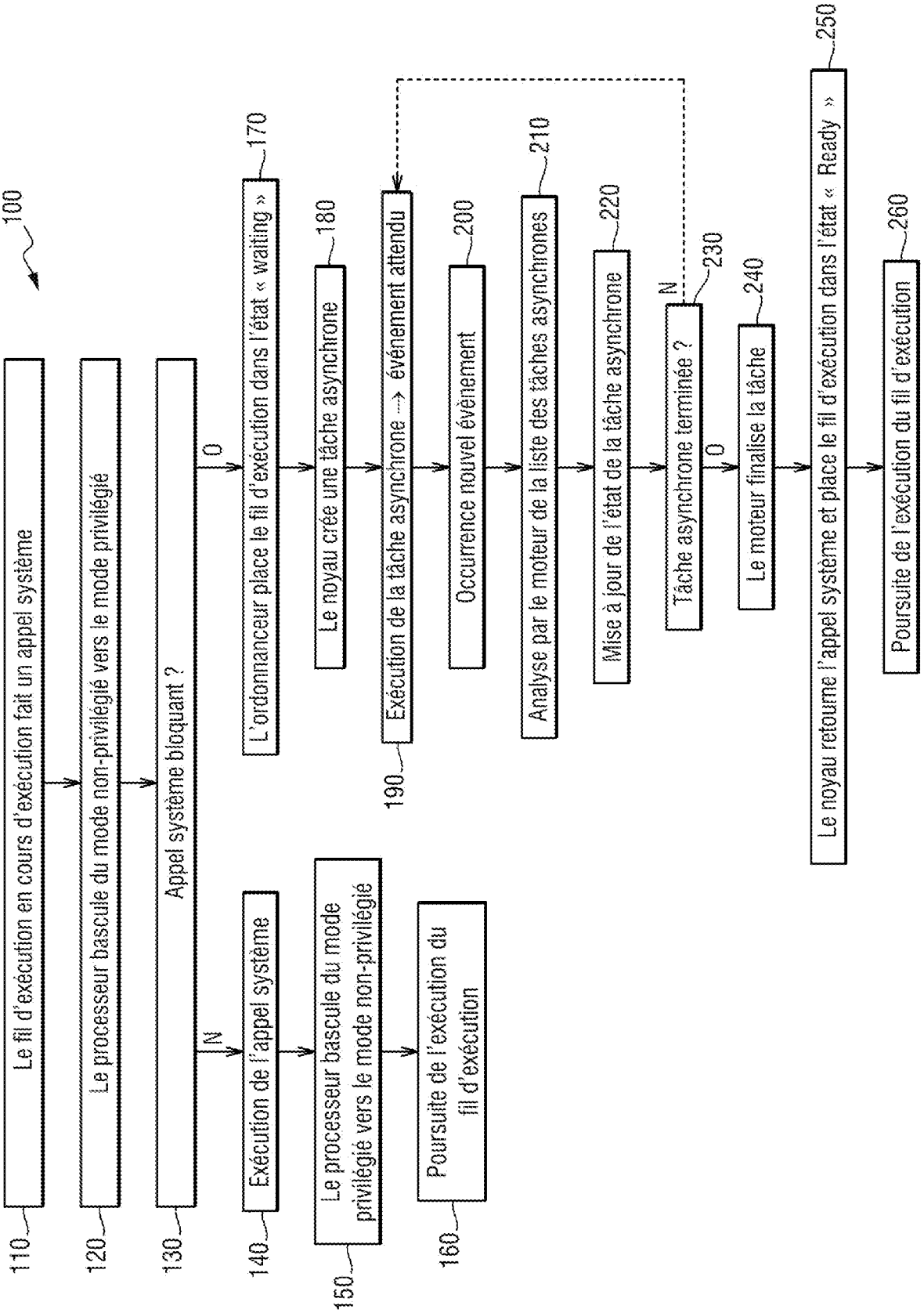
[Revendication 8] Produit programme d'ordinateur selon la revendication 7, dans lequel le noyau est associé à une unique pile noyau.

[Revendication 9] Produit programme d'ordinateur selon l'une quelconque des revendications 7 à 8, résultant de la compilation d'un code source, ledit code source se présentant sous une forme séquentielle même pour les appels systèmes bloquants.

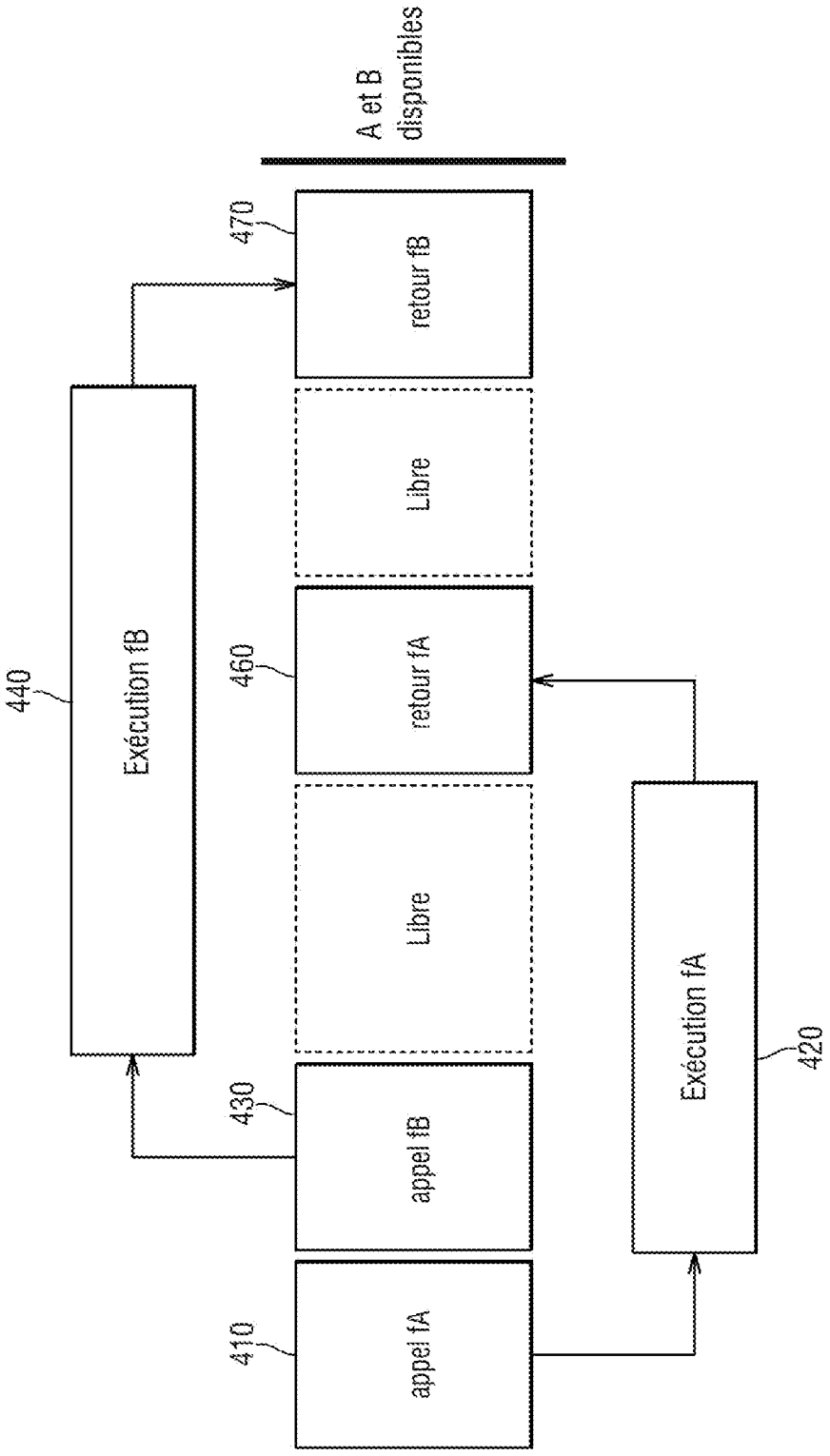
[Fig. 1]


Couche applicative
3
Système d'exploitation
2
Couche matérielle
1
Application
30
Noyau
20
Gestion appels système
23
Gestion des processus
22
Ordonnanceur
25
26
Moteur
34
35
36
Contrôleur E/S
21
24
11
12
13

[Fig. 2]


100
Le fil d'exécution en cours d'exécution fait un appel système
110
Le processeur bascule du mode non-privilégié vers le mode privilégié
120
Appel système bloquant ?
130
O
N
L'ordonnanceur place le fil d'exécution dans l'état « waiting »
170
Le noyau crée une tâche asynchrone
180
Exécution de la tâche asynchrone --> événement attendu
190
Occurrence nouvel événement
200
Analyse par le moteur de la liste des tâches asynchrones
210
Mise à jour de l'état de la tâche asynchrone
220
Tâche asynchrone terminée ?
230
N
O
Le moteur finalise la tâche
240
Le noyau retourne l'appel système et place le fil d'exécution dans l'état « Ready »
250
Poursuite de l'exécution du fil d'exécution
260
Exécution de l'appel système
140
Le processeur bascule du mode privilégié vers le mode non-privilégié
150
Poursuite de l'exécution du fil d'exécution
160

[Fig. 3]


A et B disponibles
470
retour fB
Libre
460
retour fA
Libre
430
appel fB
410
appel fA
440
Exécution fB
Exécution fA
420

RÉPUBLIQUE FRANÇAISE

INSTITUT NATIONAL DE LA PROPRIÉTÉ INDUSTRIELLE

RAPPORT DE RECHERCHE PRÉLIMINAIRE

établi sur la base des dernières revendications déposées avant le commencement de la recherche

N° d'enregistrement national

FA 913866
FR 2210701

DOCUMENTS CONSIDÉRÉS COMME PERTINENTS		Revendication(s) concernée(s)	Classement attribué à l'invention par l'INPI
Catégorie	Citation du document avec indication, en cas de besoin, des parties pertinentes		
A	US 2013/290644 A1 (SOARES LIVIO [US] ET AL) 31 octobre 2013 (2013-10-31) * abrégé * * page 1, alinéa 2 - page 3, alinéa 42 * * page 4, alinéa 47 - page 4, alinéa 57 * -----	1-9	G06F9/44 G06F9/46
A	US 2016/246643 A1 (XU JIATAO [CN] ET AL) 25 août 2016 (2016-08-25) * abrégé * * page 1, alinéa 6 - page 1, alinéa 9 * * page 2, alinéa 24 - page 4, alinéa 64 * -----	1-9	
			DOMAINES TECHNIQUES RECHERCHÉS (IPC)
			G06F

Date d'achèvement de la recherche	Examinateur
2 juin 2023	Lelait, Sylvain

CATÉGORIE DES DOCUMENTS CITÉS

X : particulièrement pertinent à lui seul
Y : particulièrement pertinent en combinaison avec un autre document de la même catégorie
A : arrière-plan technologique
O : divulgation non-écrite
P : document intercalaire

T : théorie ou principe à la base de l'invention
E : document de brevet bénéficiant d'une date antérieure à la date de dépôt et qui n'a été publié qu'à cette date de dépôt ou qu'à une date postérieure.
D : cité dans la demande
L : cité pour d'autres raisons
.....
& : membre de la même famille, document correspondant

EPO FORM 1503 12.99 (P04C14)

ANNEXE AU RAPPORT DE RECHERCHE PRÉLIMINAIRE RELATIF A LA DEMANDE DE BREVET FRANÇAIS NO. FR 2210701 FA 913866

La présente annexe indique les membres de la famille de brevets relatifs aux documents brevets cités dans le rapport de recherche préliminaire visé ci-dessus.
Les dits membres sont contenus au fichier informatique de l'Office européen des brevets à la date du 02-06-2023
Les renseignements fournis sont donnés à titre indicatif et n'engagent pas la responsabilité de l'Office européen des brevets, ni de l'Administration française

Document brevet cité au rapport de recherche	Date de publication	Membre(s) de la famille de brevet(s)	Date de publication
US 2013290644 A1	31-10-2013	US 2013275997 A1	17-10-2013
		US 2013290644 A1	31-10-2013
		US 2014223447 A1	07-08-2014
US 2016246643 A1	25-08-2016	CN 104142858 A	12-11-2014
		US 2016246643 A1	25-08-2016
		WO 2015078394 A1	04-06-2015

EPO FORM P0465