(12) DEMANDE INTERNATIONALE PUBLIÉE EN VERTU DU TRAITÉ DE COOPÉRATION EN MATIÈRE DE BREVETS (PCT)

(19) Organisation Mondiale de la Propriété Intellectuelle
Bureau international

(43) Date de la publication internationale
2 février 2006 (02.02.2006)

PCT

(10) Numéro de publication internationale
WO 2006/010812 A2

(51) Classification internationale des brevets :
G06F 11/14 (2006.01)

(21) Numéro de la demande internationale :
PCT/FR2005/001564

(22) Date de dépôt international : 22 juin 2005 (22.06.2005)

(25) Langue de dépôt : français

(26) Langue de publication : français

(30) Données relatives à la priorité :
04 07180 30 juin 2004 (30.06.2004) FR

(71) Déposant *(pour tous les États désignés sauf US)* : **MEIOSYS** [FR/FR]; Centre Industriel d'Innovation de Basso Cambo, 42, avenue du Général de Croutte, F-31100 Toulouse (FR).

(72) Inventeur; et

(75) Inventeur/Déposant *(pour US seulement)* : **VERTES, Marc** [FR/FR]; 858 chemin Guiraoudéou, F-31470 Saint Lys (FR).

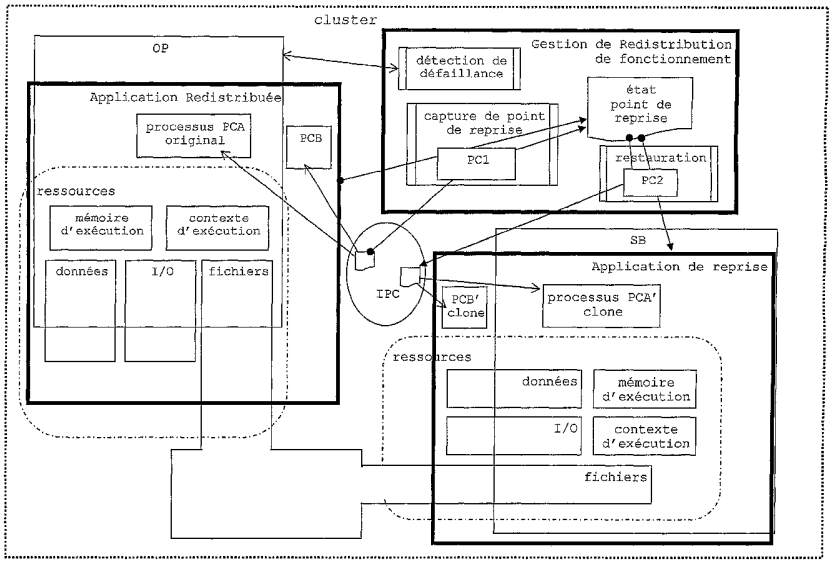
(81) États désignés *(sauf indication contraire, pour tout titre de protection nationale disponible)* : AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SM, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) États désignés *(sauf indication contraire, pour tout titre de protection régionale disponible)* : ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), eurasien (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), européen (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

[Suite sur la page suivante]

(54) **Title:** METHOD FOR CONTROLLING A SOFTWARE PROCESS, METHOD AND SYSTEM FOR REDISTRIBUTING OR CONTINUING OPERATION IN A MULTI-COMPUTER ARCHITECTURE

(54) **Titre :** PROCEDE DE GESTION D'UN PROCESSUS LOGICIEL, PROCEDE ET SYSTEME DE REDISTRIBUTION OU DE CONTINUITE DE FONCTIONNEMENT DANS UNE ARCHITECTURE MULTI-ORDINATEURS



(57) **Abstract:** The invention relates a method for controlling a software application in a multi-computer architecture (cluster). Said control can be applied, for example for analysis or modification of the environment thereof in a transparent as possible manner with respect to said application. The control is applicable for analysing, capturing and restoring operations of the status of one or several application processes. Said operations use a controller which is external to the application and injects system call instructions into the working memory of the controllable process(es).

[Suite sur la page suivante]

WO 2006/010812 A2

Publiée :

— *sans rapport de recherche internationale, sera republiée dès réception de ce rapport*

En ce qui concerne les codes à deux lettres et autres abréviations, se référer aux "Notes explicatives relatives aux codes et abréviations" figurant au début de chaque numéro ordinaire de la Gazette du PCT.

(57) Abrégé : La présente invention concerne un procédé de gestion d'une application logicielle fonctionnant dans une architecture multi-ordinateurs (cluster). Cette gestion s'applique par exemple à l'analyse ou la modification de son environnement d'exécution, d'une façon la plus transparente possible vis-à-vis de cette application. Cette gestion est appliquée à des opérations d'analyse, de capture et de restauration de l'état d'un ou plusieurs processus de l'application. Ces opérations utilisent un contrôleur externe à l'application qui réalise une injection d'instructions d'appels système à l'intérieur de la mémoire de travail du ou des processus à gérer.

« Procédé de gestion d'un processus logiciel, procédé et système de redistribution ou de continuité de fonctionnement dans une architecture multi-ordinateurs »

La présente invention concerne un procédé de gestion d'une application logicielle fonctionnant dans une architecture multi-ordinateurs (cluster), par exemple d'analyse ou de modification de son environnement d'exécution, d'une façon la plus transparente possible vis-à-vis de cette application. Elle concerne également un procédé pour modifier ou ajuster le fonctionnement d'une telle application en utilisant ce procédé de gestion de fonctionnement pour réaliser une redistribution de ses processus au sein d'un cluster. Ce procédé de redistribution peut en particulier être utilisé pour modifier la répartition de la charge de travail entre différentes machines d'un réseau, ou pour fiabiliser l'application en en améliorant la continuité de fonctionnement. L'invention concerne également un système multi-ordinateurs mettant en œuvre ce procédé de redistribution de fonctionnement.

Le domaine de l'invention est celui des réseaux ou clusters d'ordinateurs formés de plusieurs ordinateurs collaborant entre eux. Ces clusters sont utilisés pour exécuter des applications logicielles apportant un ou des services à des utilisateurs. Une telle application peut être mono-processus ou multi-processus, et être exécutée sur un seul ordinateur ou distribuée sur plusieurs ordinateurs, par exemple sous la forme d'une application distribuée de type MPI (« Message Passing Interface »).

A un instant donné, dans un contexte d'architecture redondante et communicante, une application est exécutée sur un ordinateur ou un groupe d'ordinateurs du cluster, appelé nœud primaire ou opérationnel, tandis que les autres ordinateurs du cluster sont appelés nœuds secondaires ou « stand-by ». Or, l'exploitation de tels clusters montre que se posent des problèmes de fiabilité qui peuvent être dus à des défaillances du matériel ou du système d'exploitation, à des erreurs humaines, ou à la défaillance des applications elles-mêmes.

Pour résoudre ces problèmes de fiabilité, il existe actuellement des mécanismes, dits de haute disponibilité, qui sont mis en œuvre sur la

plupart des clusters actuels et qui sont basés sur un redémarrage automatique à froid de l'application sur un nœud de secours parmi l'un des nœuds secondaires du cluster.

Or, pour revenir à une situation proche de celle existant lors de la défaillance, ces mécanismes basés sur un redémarrage à froid présentent souvent une durée et une complexité de mise en œuvre importante, ce qui nuit à la bonne continuité du service fourni par l'application en cours d'exécution au moment de la défaillance.

Pour améliorer cette continuité, il est également connu, par exemple par le brevet FR 02/09855, de prévoir un ou plusieurs clones du nœud opérationnel, maintenus à jour périodiquement ou au fil de l'eau sur des nœuds secondaires.

De plus, au cours de l'exploitation de tels clusters, certaines ressources matérielles comme des ordinateurs ou des voies ou lignes de communication peuvent présenter une charge de travail trop importante, créant ainsi des goulots d'étranglement alors que d'autres sont sous-utilisées.

Pour améliorer les performances de l'application il est possible de réorganiser la répartition de l'application au sein du cluster.

Toutefois, l'ensemble de ces techniques nécessite d'intervenir sur des processus en cours d'exécution, par des opérations de gestion de fonctionnement comme des opérations d'analyse, de capture ou de restauration des processus ou des ressources utilisées par l'application.

Or, de telles fonctionnalités ne sont pas forcément prévues dans l'application et les informations à relever ou à modifier ne sont pas toujours accessibles aux fonctions externes à l'application, par exemple au niveau du système d'exploitation.

Lorsque de telles fonctionnalités ne sont pas prévues directement à l'intérieur de l'application, il est alors coûteux et complexe voire impossible de les y intégrer ultérieurement, et cela nécessite souvent l'intervention du concepteur de l'application.

Pour implémenter de telles fonctionnalités sans intervenir directement dans la programmation de l'application, il est possible de modifier certaines instructions utilisées par l'application pour les enrichir des fonctionnalités

nécessaires, ou d'ajouter ces fonctionnalités à divers stades de la compilation ou de l'exécution du code de l'application.

Pour cela, il est possible de modifier ou d'enrichir certains modules du système d'exploitation, par exemple au niveau du noyau (kernel).

Toutefois, de telles modifications nuisent à l'homogénéité des différentes configurations utilisées au sein du réseau, et ne peuvent être facilement modifiées en cours d'exécution.

Des bibliothèques supplémentaires peuvent également être intégrées au cours de la compilation, pour ajouter ces fonctionnalités de façon permanente au code exécutable. De telles bibliothèques peuvent même réaliser une interposition entre les appels prévus dans l'application et les bibliothèques d'origines comme décrit dans le brevet FR 02/00398, permettant de détourner ces appels vers une nouvelle bibliothèque, modifiable en cours d'exécution.

Toutefois ces méthodes nécessitent d'intervenir au stade de la compilation de l'application, ce qui est coûteux et complexe, peut nécessiter une intervention du concepteur de l'application et être malgré tout source d'erreurs ou d'incompatibilités.

Au sein d'une telle architecture, l'implémentation de certaines fonctionnalités de gestion d'un processus est donc délicate à réaliser sans modification ou intervention dans l'application ou dans le système ou les deux, ce qui est source de coût, de complexité et de risques d'erreurs.

Un objectif de l'invention est alors de permettre une gestion plus complète d'un processus applicatif, de façon plus transparente pour le fonctionnement de cette application.

Cet objectif est atteint avec un procédé de gestion d'une application logicielle comprenant au moins un premier processus logiciel, dit processus cible, s'exécutant sur au moins un ordinateur et dans un environnement d'exécution comprenant au moins un espace mémoire d'exécution.

Selon l'invention, ce procédé comprend une opération d'injection d'au moins une instruction exécutable dans l'espace mémoire du processus cible, par au moins un deuxième processus logiciel, dit processus contrôleur, extérieur à l'application et apte à agir sur le déroulement du processus

cible, cette instruction exécutable réalisant une analyse ou une modification de l'environnement d'exécution de ce processus cible.

Plus particulièrement, l'opération d'injection comprend des étapes de :

- interruption de l'exécution du processus cible par le processus contrôleur ;
- écriture par le processus contrôleur dans une partie, dite zone réattribuée, de l'espace mémoire d'exécution du processus cible, d'instructions injectées réalisant le mécanisme d'analyse ou de modification ;
- exécution, par le processus cible, de ces instructions injectées ;
- restauration par le processus contrôleur, par écriture dans la zone réattribuée, des instructions du processus cible qui y étaient mémorisées avant l'interruption ;
- exécution de la suite des instructions du processus cible.

Avantageusement, ce procédé de gestion de fonctionnement comprend en outre une combinaison des caractéristiques suivantes :

L'étape d'interruption du processus cible peut être suivie d'au moins une étape de lecture et sauvegarde des instructions mémorisées dans la zone réattribuée et/ou de l'état du contexte d'exécution du processus cible lors de son interruption.

L'étape d'écriture d'instructions injectées peut être précédée d'une étape d'écriture dans la zone réattribuée de données réalisant une correspondance par adressage entre cette zone réattribuée et un autre espace mémoire déterminé, dit zone de mapping.

L'étape d'exécution des instructions injectées peut être précédée d'une étape d'écriture dans la zone réattribuée de données constituant des arguments des instructions injectées.

L'étape d'exécution des instructions injectées peut également être précédée d'une étape de modification du contexte d'exécution en fonction de paramètres correspondant aux instructions injectées.

L'étape d'exécution des instructions injectées peut être suivie d'une étape de lecture de données mémorisées dans la zone réattribuée et/ou de lecture de l'état du contexte d'exécution du processus cible.

L'étape d'écriture d'instructions injectées peut comprendre l'écriture d'au moins une instruction d'interruption d'exécution dans la zone réattribuée après les instructions injectées.

Un autre but de l'invention est de faciliter l'implémentation dans le fonctionnement une application, de façon la plus transparente possible pour cette application, de fonctionnalités permettant l'analyse, la capture ou la modification de l'environnement de cette application ou des ressources qu'elle utilise.

Pour cela, l'invention propose un procédé de gestion du fonctionnement d'une application logicielle tel que ci-dessus, réalisant une opération d'introspection d'au moins deux processus introspectés, chacun de ces processus introspectés utilisant une première ressource comportant elle-même un pointeur désignant une deuxième ressource comportant elle-même un attribut qui est accessible audit processus à travers ledit pointeur , le procédé comprenant les étapes suivantes :

- injection par le processus contrôleur dans chacun des deux processus introspectés d'au moins une instruction système réalisant une lecture initiale de la valeur de l'attribut de la deuxième ressource correspondant à chacun desdits processus introspectés ;
- injection par le processus contrôleur dans l'un des deux processus introspectés, dit processus de test, d'au moins une instruction système réalisant une modification de la valeur de l'attribut de la deuxième ressource correspondant audit processus test ;
- injection par le processus contrôleur dans l'autre processus introspecté, dit processus témoin, d'au moins une instruction système réalisant une deuxième lecture de la valeur de l'attribut de la deuxième ressource correspondant audit processus témoin ;
- comparaison par le processus contrôleur de la valeur de deuxième lecture avec la valeur de lecture initiale dudit processus témoin ;
- mémorisation par le processus contrôleur d'une donnée représentant le résultat de ladite comparaison et injection par le processus contrôleur dans le processus de test, d'au moins une instruction système réalisant une modification de la valeur de l'attribut de la

deuxième ressource correspondant audit processus test, de façon à lui redonner sa valeur de lecture initiale.

Pour cela, l'invention propose également un procédé de gestion du fonctionnement d'une application logicielle tel que ci-dessus, réalisant une opération de capture de l'état du processus cible, dit processus capturé, et comprenant des étapes de :

- prise de contrôle du processus capturé par un processus contrôleur ;
- injection par le processus contrôleur dans le processus capturé d'au moins une instruction d'appel système réalisant une analyse de la structure de l'environnement d'exécution du processus capturé ;
- mémorisation ou transmission de données de résultat représentant le résultat de cette analyse et restauration de l'espace mémoire du processus capturé ;
- exécution de la suite des instructions du processus capturé.

Lorsque l'application à gérer est de type multi-processus, multi-tâches ou « multi-threads », l'opération de capture décrite ci-dessus peut également se combiner avec les caractéristiques suivantes.

Le procédé de gestion de fonctionnement peut en particulier réaliser une opération de capture de l'état d'au moins deux processus de cette application, l'interruption de ces deux processus se faisant soit de façon simultanée soit en des points de leurs déroulements respectifs dont l'un est calculé en fonction de l'autre.

Lorsque le processus capturé échange des données de communication avec au moins un autre processus par l'intermédiaire d'au moins un agent logiciel interprocessus extérieur à l'application, l'opération de capture peut comprendre en outre des étapes de :

- injection, par le processus contrôleur dans le processus capturé d'au moins une instruction d'appel système réalisant la lecture dans l'agent interprocessus d'au moins une donnée de communication en provenance d'un autre processus de l'application et non encore reçue par le processus capturé ;
- mémorisation ou transmission de cette donnée de communication en tant que donnée de résultat.

Lorsque l'environnement d'exécution du processus capturé supporte la transmission de caractéristiques entre processus par des relations d'héritage, l'opération de capture peut comprendre en outre des étapes de :

- injection, par le processus contrôleur dans le processus capturé d'au moins une instruction d'appel système réalisant une analyse des relations d'héritage du processus capturé avec au moins un autre processus de l'application ;
- mémorisation ou transmission de données de résultat représentant les relations d'héritage du processus capturé.

Dans le même esprit, l'invention propose également un procédé de gestion de fonctionnement d'une application logicielle tel que ci-dessus, réalisant une opération de restauration, par un processus contrôleur à partir de données dites de reprise, de l'état d'au moins un processus d'application logicielle, dit processus de reprise. L'opération de restauration comprend alors des étapes de :

- interruption de l'exécution du processus de reprise par le processus contrôleur ;
- injection par le processus contrôleur dans le processus de reprise d'au moins une instruction d'appel système créant ou modifiant la structure d'au moins un objet logiciel appartenant à l'environnement d'exécution du processus de reprise, en fonction des données de reprises ;
- écriture, à partir des données de reprise, de l'espace mémoire d'exécution du processus de reprise ;
- lancement du processus de reprise et exécution de la suite de ses instructions.

Lorsque l'application à gérer est de type multi-processus, multi-tâches ou multi-threads, l'opération de restauration également décrite ci-dessus peut également se combiner avec les caractéristiques suivantes.

Lorsque l'environnement d'exécution du processus de reprise supporte ou utilise l'échange de données de communication entre plusieurs processus par l'intermédiaire d'au moins un agent logiciel interprocessus extérieur à l'application, l'opération de restauration peut comprendre en outre une étape de :

- injection, par le processus contrôleur dans le processus capturé d'au moins une instruction d'appel système réalisant, à partir des données de reprise, l'écriture au sein de l'agent interprocessus d'au moins une donnée représentant une donnée de communication à destination du processus de reprise.

Lorsque l'environnement d'exécution du processus de reprise supporte la transmission de caractéristiques entre processus par des relations d'héritage, l'opération de restauration peut comprendre en outre une étape de :

- injection, par le processus contrôleur dans le processus de reprise d'au moins une instruction d'appel système créant ou modifiant, à partir des données de reprise, au moins une relations d'héritage du processus de reprise avec au moins un autre processus de l'application.

Une telle implémentation de fonctionnalités de gestion d'un processus d'une application permet en particulier d'intervenir sur le fonctionnement de cette application et des services qu'elle produit, à moindre coût et en diminuant la complexité et le risque d'erreurs.

Or, pour gérer le fonctionnement d'une application, il est utile de mieux gérer la façon dont une application utilise des ressources matérielles au sein d'un cluster, tout en limitant les interventions à l'intérieur du fonctionnement d'une application et les risques et complexités que cela comporte.

Un autre but de l'invention est ainsi de pouvoir déplacer l'exécution de tout ou partie de cette application d'une ressource matérielle à une autre, par exemple d'un ordinateur à l'autre ou d'un nœud à un autre.

Pour cela l'invention propose d'utiliser le procédé ci-dessus pour réaliser un procédé de réplication d'au moins un processus de l'application, dit processus original, en un processus clone, comprenant les étapes suivantes :

- capture de l'état du processus original par un procédé selon l'une des revendications 2 à 6 ;

- utilisation de données de résultat, issues de la capture, pour mémoriser un objet logiciel appelé point de reprise, représentant un état de ce processus original en un point de son exécution ;
- utilisation de données du point de reprise pour restaurer au moins un processus clone dans un état reproduisant l'état du processus original.

Dans le même esprit, l'invention propose également d'utiliser le procédé ci-dessus pour réaliser un procédé de redistribution de tout ou partie d'une application logicielle dite redistribuée, exécutée dans une architecture multi-ordinateurs (cluster) et comprenant au moins un processus, dit processus initial, fournissant un traitement de données en étant exécuté à un instant donné sur au moins un ordinateur du cluster, appelé nœud primaire ou opérationnel, d'autres ordinateurs dudit cluster étant appelés nœuds secondaires, ce procédé de redistribution comprenant les étapes suivantes :

- réplication d'au moins un processus initial en au moins un processus secondaire exécuté sur un nœud secondaire ;
- basculement de tout ou partie du traitement de données du processus initial vers au moins un processus secondaire.

Une telle redistribution permet en particulier de transférer telle ou telle tâche de calcul d'un nœud à l'autre au sein du cluster. Il est ainsi possible de redistribuer la charge de travail des différentes machines, pour obtenir un meilleur équilibrage de cette charge de travail au sein du cluster. Il est également possible de déplacer certains processus sur des machines plus proches des ressources qu'utilisent ces processus ou disposant de meilleures communications, par exemple pour réduire les temps de transmission entre certains processus et les bases de données qu'ils utilisent.

Selon une particularité, le procédé de redistribution comprend en outre les étapes suivantes :

- réplication de tous les processus exécutés par le nœud opérationnel en un ou plusieurs processus secondaires exécutés sur au moins un nœud secondaire ;
- basculement de la totalité des traitements de données desdits processus vers au moins un desdits processus secondaires.

Il est ainsi possible de déplacer tous les processus utilisés par un matériel déterminé. Cela permet en particulier rendre l'application indépendante de ce matériel, par exemple dans le cas d'un ordinateur devant éteint pour entretien ou remplacement.

Dans un but similaire, l'invention propose également d'utiliser le procédé ci-dessus pour réaliser un procédé de suspension d'une application logicielle comprenant au moins un processus exécuté sur au moins un ordinateur, ce procédé de suspension comprenant les étapes suivantes :

- capture de l'état de l'ensemble des processus de l'application, par un procédé selon l'une des revendications précédentes 3 à 5 ;
- utilisation de données de résultat, issues de la capture, pour mémoriser un objet logiciel appelé point de reprise, représentant un état de cette application en un point de son exécution ;
- utilisation de données du point de reprise pour restaurer un ou plusieurs processus clones dans un état reproduisant l'état de l'ensemble des processus capturés.

Il est ainsi possible de sauvegarder dans des moyens de mémorisation l'ensemble d'une application dans son état à un moment donné. Une telle sauvegarde peut alors être conservée et stockée, par exemple à titre de témoignage ou de sécurité.

L'étape de restauration peut s'effectuer sur une même machine ou sur une autre, au moment choisi. Il est ainsi possible de faciliter la maintenance ou le remplacement d'une machine, en particulier lorsqu'il n'est pas possible de transférer l'application dans une autre partie d'un cluster. Il est également de possible ainsi de faciliter le transfert d'une application vers une ou plusieurs autres machines, par exemple avec lesquelles il n'y a pas de communications numériques directes.

Un autre but est de proposer un procédé pour réaliser une amélioration de la continuité de fonctionnement d'une application logicielle s'exécutant dans une architecture multi-ordinateurs.

Ce but est atteint par un procédé de fiabilisation du fonctionnement d'une application logicielle, dite application fiabilisée, exécutée dans une architecture multi-ordinateurs (cluster) et fournissant un service déterminé, au moins un processus de cette application étant exécuté à un instant

donné sur au moins un ordinateur du cluster, appelé nœud primaire ou opérationnel, d'autres ordinateurs dudit cluster étant appelés nœuds secondaires. Ce procédé de fiabilisation met en œuvre un procédé de gestion décrit ci-dessus pour réaliser au moins une opération de capture et au moins une opération de restauration, et comprend les étapes suivantes :

- capture par au moins un processus contrôleur de l'état de l'ensemble des processus de cette application fiabilisée ;
- utilisation de données de résultat, issues de la capture, pour mémoriser un objet logiciel appelé point de reprise, représentant un état de cette application fiabilisée en un point de son exécution ;
- détection au sein du nœud opérationnel d'une défaillance matérielle ou logicielle affectant le fonctionnement de l'application fiabilisée ;
- utilisation de tout ou partie des données du point de reprise pour restaurer, sur au moins un nœud secondaire, un ou plusieurs processus d'une application de secours en un état reproduisant l'état de l'ensemble des processus de l'application fiabilisée ;
- basculement de tout ou partie du service vers l'application de secours d'au moins l'un desdits noeuds secondaires.

Plus particulièrement, le procédé de gestion de fonctionnement selon l'invention peut associer, sélectivement ou non, des opérations de capture à des opérations de restauration pour réaliser une réplication holistique de l'état d'une application dite originale en une application clone. Le procédé de réplication décrit ci-dessus est alors mis en œuvre pour répliquer l'ensemble des processus et ressources de l'application originale en tant que processus et ressources de l'application clone.

Selon le même concept inventif, ce procédé de continuité de fonctionnement peut bien sûr mettre à jour ou restaurer un ou plusieurs processus clones après la détection d'une défaillance plutôt qu'avant, ou réaliser une combinaison des deux.

Ainsi, l'invention propose également un procédé de fiabilisation d'une application logicielle dite fiabilisée, exécutée dans une architecture multi-ordinateurs (cluster) et fournissant un service déterminé, au moins un processus de cette application, dit processus fiabilisé, étant exécuté à un instant donné sur au moins un ordinateur du cluster, appelé nœud primaire

ou opérationnel, d'autres ordinateurs dudit cluster étant appelés nœuds secondaires, ce procédé de fiabilisation comprenant les étapes suivantes :

- mise en œuvre d'un procédé de réplication holistique pour répliquer, sur au moins un nœud secondaire, une application de secours dans un état identique à celui de l'application fiabilisée ;
- détection au sein du nœud opérationnel d'une défaillance matérielle ou logicielle affectant le fonctionnement de l'application fiabilisée ;
- basculement de tout ou partie du service vers ladite application de secours d'au moins un des nœuds secondaires.

L'invention propose également un système multi-ordinateurs mettant en œuvre le procédé selon l'invention.

Un avantage de l'utilisation d'un processus contrôleur différent du processus à gérer, c'est-à-dire du processus cible, est en particulier de pouvoir implémenter les opérations nécessaires aux fonctionnalités de continuité ou de redistribution de fonctionnement sous la forme d'opérations externes à l'application, c'est-à-dire à l'extérieur de l'espace mémoire du processus cible. Ces opérations externes sont par exemple des définitions de points de reprise, des déclenchements de captures ou de restauration d'états, des analyses ou modifications de structures de ressources, ou des lectures ou écritures de données dans ces ressources.

Ces calculs et opérations représentent en effet un certain volume de calcul dont seule une petite partie a besoin d'être exécutée depuis le processus cible. Il est donc avantageux d'injecter cette petite partie tout en réalisant le reste de la gestion de la redistribution ou de la continuité de fonctionnement en dehors de l'application devant être redistribuée ou fiabilisée. Cela permet au processus cible, donc à l'ensemble de l'application cible, de rester inchangé avant et après une opération de capture lors d'un point de reprise (checkpointing) ou de restauration (par démarrage ou mise à jour d'un clone)

Combiné avec la gestion par un contrôleur extérieur à l'application, le fait d'utiliser une méthode d'implémentation par injection de code permet ainsi d'accéder à des fonctionnalités systèmes de l'intérieur de l'application pour les tâches qui le réclament, sans intervenir dans l'application. Comparé aux méthodes d'intervention externe utilisées par les programmes de mise

au point (ou débugueurs), par exemple « GDB », cet accès de l'intérieur permet à la gestion d'un processus de ne pas dépendre des limites de fonctionnalités propres à ces débugueurs. Par exemple, la présente invention permet de n'être pas limité, à travers la liste de « symboles de debug » de l'application cible, aux fonctions déjà présentes dans cette application cible.

De plus, les appels systèmes réalisés par injection permettent d'utiliser des paramètres mémorisés dans les registres, et au sommet de la pile (stack) comme c'est le cas de nombreux débugueurs. Ainsi, cette méthode par injection permet également de s'affranchir des autorisations d'accès à certaines ressources comme la pile exécutable (stack execution permission), qui peuvent exister dans certains systèmes d'exploitation comme SELinux, SUN-Solaris ou OpenBSD.

Cette combinaison de contrôleur et d'injection d'instructions permet de réaliser un procédé de déclenchement capture d'un point de reprise ou un procédé de restauration qui soit simple et direct. A titre d'ordre de grandeur, un programme basique de démonstration réalisant ces fonctionnalités de réplication pour un processus unique sans fichiers ni connexions peut représenter environ 500 lignes de programmes en langage C.

Par ailleurs, l'aspect restreint et temporaire de la méthode d'injection d'appels systèmes (system call injection) permet de n'insérer que peu d'instructions dans l'espace mémoire du processus à gérer et dont rien ne subsiste en fin d'opération. Cela permet donc d'éviter de « polluer » le processus cible, ce qui est un avantage du point de vue de la fiabilité comme de la maintenance de l'application.

Le procédé selon l'invention présente l'avantage d'être utilisable aussi bien avec une application cible utilisant des fichiers exécutables statiques, c'est-à-dire comprenant toutes les routines nécessaires, que dynamiques, c'est-à-dire faisant appel à des bibliothèques de sous-programmes à l'extérieur de l'application.

Par ailleurs, le procédé selon l'invention permet de réaliser une redistribution ou une continuité de fonctionnement en intervenant peu ou pas en dehors du domaine de travail de l'utilisateur. En particulier,

l'implémentation des opérations de capture de point de reprise (checkpointing) et de restauration en elles-mêmes ne nécessitent que peu ou pas de modification du système (kernel) ou d'ajout de ressources systèmes (kernel modules). En évitant d'intervenir dans le système ou le noyau des nœuds concernés, cet aspect permet entre autres de minimiser les besoins en spécialistes systèmes, et d'homogénéiser les configurations systèmes installées sur les différents ordinateurs du cluster.

De plus, le fait que le processus contrôleur puisse effectuer une restauration de l'état d'un processus de reprise sans avoir réalisé lui-même le démarrage de ce processus de reprise permet de travailler sur un processus de reprise existant. Cette possibilité permet à la gestion de redistribution ou de continuité de fonctionnement, de ne pas interférer avec les modes de démarrage d'une application cible ou de ses processus, ce qui facilite par exemple l'application de l'invention à des applications distribuées (MPI).

D'autres particularités et avantages de l'invention ressortiront de la description détaillée d'un mode de mise en œuvre nullement limitatif, et des dessins annexés sur lesquels :

- la figure 1a représente l'organisation d'un cluster exécutant une application logicielle, dont le fonctionnement est fiabilisé par une application de redistribution mettant en œuvre un procédé selon l'invention pour réaliser une redistribution complète ;
- la figure 1b représente l'organisation d'un cluster exécutant une application logicielle, dont le fonctionnement est ajusté par une application de redistribution mettant en œuvre un procédé selon l'invention pour réaliser une redistribution partielle ;
- la figure 2 est un schéma symbolique du déroulement d'une opération d'injection d'instructions de programme par un processus contrôleur au sein d'un processus cible ;
- la figure 3 est un schéma symbolique du fonctionnement d'une opération de capture de l'état d'un processus ;
- la figure 4 est un schéma symbolique du fonctionnement d'une opération de restauration d'un processus de reprise ;

- la figure 5 est un schéma illustrant la structure de deux processus utilisant des descripteurs de fichiers partagés ou séparés ;
- la figure 6 est un schéma illustrant le déroulement d'un procédé d'introspection multi-processus utilisant une injection d'appels systèmes.

Dans la suite de la description, des exemples de commandes ou d'instructions utilisées pour réaliser le procédé selon l'invention sont présentés en utilisant le langage C et pour un environnement ou système d'exploitation de type Unix ou dérivés, en particulier POSIX. D'autres langages ou environnements systèmes peuvent bien sûr être utilisés pour implémenter l'invention.

En figures 1a et 1b sont illustrées des utilisations d'un procédé de réplication selon l'invention dans une application de redistribution de fonctionnement. Cette application de redistribution de fonctionnement est utilisée pour redistribuer le fonctionnement d'une application logicielle, dite application redistribuée, exécutée sur un nœud opérationnel OP d'une architecture multi-ordinateurs ou cluster. Un tel nœud peut être un ordinateur unique au sein du cluster ou comprendre plusieurs ordinateurs travaillant ensemble au sein du cluster.

L'application redistribuée comprend au moins un processus, dit processus original PCA, travaillant dans un environnement d'exécution dans lequel il accède à un certain nombre de ressources de différents types. De façon courante, ces ressources comprennent :

- un espace mémoire d'exécution alloué dans la mémoire de travail du nœud OP, et où mémorisées les instructions exécutées constituant le processus ;
- un contexte d'exécution, incluant des registres mémoires et différents types de ressources d'état telles que flags, mutex, etc. ;
- des zones mémoires I/O (Input/Output) utilisées par l'ordinateur pour gérer les entrées et sorties avec l'utilisateur ou d'autres intervenants matériels ou logiciels ;
- des données mémorisées, par exemple des variables gérées par le processus ou des fichiers de données dont certaines peuvent être

partagées avec d'autres applications non représentées communiquant avec l'application redistribuée.

Parmi les ressources accessibles à un processus, certaines peuvent se trouver réparties sur plusieurs ordinateurs ou plusieurs nœuds, en particulier dans le cas d'applications distribuées, par exemple pour des variables mémorisées dans des zones de mémoire partagée ou sous forme de fichiers partagés ou de bases de données extérieures.

L'application de redistribution de fonctionnement est exécutée sur un ou plusieurs ordinateurs du cluster communiquant avec le nœud opérationnel de l'application et au moins un nœud secondaire SB. Cette redistribution de fonctionnement se fait en mémorisant de façon régulière ou sur évènement, en un point de reprise (« checkpoint »), un état instantané d'un ou plusieurs processus originaux PCA de l'application redistribuée.

Lors du déclenchement d'un point de reprise, l'application de redistribution réalise une opération de capture de point de reprise, selon un procédé décrit ci-après. Selon l'invention, cette opération de capture de point de reprise, utilise un procédé de gestion de fonctionnement de l'application redistribuée, décrit ci-après, mis en œuvre par un processus contrôleur PC1 temporaire agissant sur le processus original PCA de l'application redistribuée.

A l'issue de cette capture de point de reprise, l'application de redistribution mémorise un objet logiciel, dit état de point de reprise, dans des moyens de mémoire au sein du cluster. En plus de l'opération de capture selon l'invention, certaines ressources de l'application redistribuée, comme des bases de données ou des fichiers, peuvent également être sauvegardées ou répliquées au fil de l'eau ou par étapes, selon des moyens connus.

Dans un mode de réalisation, l'application de redistribution réalise une redistribution complète de l'application redistribuée, c'est-à-dire de l'ensemble de ses processus et des liens qui les unissent.

Ainsi qu'illustré en figure 1a, une telle redistribution complète peut en particulier être utilisée pour fiabiliser l'application redistribuée, en constituant une application de secours, qui maintiendra une certaine

continuité dans le service fourni en cas de défaillance du nœud opérationnel OP.

Pour cela, l'application de redistribution de fonctionnement utilise un état de point de reprise pour effectuer une ou plusieurs restaurations de l'application redistribuée sous la forme d'au moins une application de secours, dite application de reprise. Une telle application de reprise comprend un processus clone exécuté sur un nœud secondaire SB du cluster et des ressources lui assurant un état correspondant à l'état du processus original PCA lors de la capture de ce point de reprise.

Cette restauration peut se faire de façon régulière ou sur évènement, et peut comprendre un démarrage complet avec création du processus clone, également appelé processus de reprise, ou effectuer une restauration par mise à jour d'un processus clone déjà existant.

Lors de cette restauration, l'application de redistribution réalise une opération de mise à jour du processus clone à partir d'un point de reprise, selon un procédé décrit ci-après. Selon l'invention, cette opération de mise à jour utilise un procédé de gestion de fonctionnement, décrit ci-après, mis en œuvre par un processus contrôleur PC2 temporaire agissant sur le processus clone de l'application de reprise par injection d'appels système, comme décrit ci-après.

En cas de défaillance affectant le fonctionnement, sur le nœud opérationnel, de l'application fiabilisée, l'application de redistribution de fonctionnement en est avertie par une fonction de surveillance ou de détection de défaillance, selon des moyens connus. L'application de redistribution de fonctionnement effectue alors un basculement de service vers l'application de secours, et le processus clone reprend alors le rôle que jouait le processus original PCA avant la défaillance.

Dans d'autres modes de réalisation, non représentés, l'application de redistribution de service peut également réaliser une mise à jour de l'application de reprise après la défaillance, ou un démarrage complet de cette application de reprise suivi d'une mise à jour selon le procédé de l'invention.

Dans d'autres particularités non illustrées ici, une telle redistribution complète peut être utilisée également pour déplacer complètement une

application d'un nœud à un autre, par exemple pour libérer ce nœud pour une intervention matérielle.

En conservant les données de l'état de point de reprise un certain temps avant de restaurer l'application de reprise, on peut également réaliser un archivage de l'application redistribuée, ou une suspension de cette application par exemple pendant le temps d'une intervention matérielle sur le nœud opérationnel. En mémorisant les données de l'état de point de reprise sur un support transportable, il est également possible de déplacer cette application vers un autre ordinateur ou un autre cluster, sans nécessité d'une liaison informatique.

Dans un mode de réalisation illustré en figure 1b, l'application de redistribution réalise une redistribution partielle de l'application redistribuée, c'est-à-dire par une réplication d'une partie seulement de ses processus et des liens qui les unissent entre eux, tout en réactualisant les liens qui les unissent aux autres processus.

Lorsque l'application de redistribution de fonctionnement reçoit une commande de redistribution partielle, elle réalise un état de point de reprise portant sur le ou les processus à répliquer, ou identifie un état de point de reprise déjà mémorisé portant sur ces mêmes processus.

Pour chaque processus, dit processus original PCA, à répliquer, l'application de redistribution de fonctionnement crée un processus clone PCA' au sein du nœud SB vers lequel doit être redistribué le processus original PCA.

A partir de cet état de point de reprise, l'application de redistribution de fonctionnement effectue une restauration du processus clone PCA' dans l'état du processus original PCA au moment de l'établissement du point de reprise. Cette restauration comprend également une restauration, entre les différents processus clones, de l'état des liens qui existent entre leurs processus originaux respectifs. Si le processus original PCA comporte des liens avec un autre processus PCB qui n'a pas été répliqué, un lien dans le même état sera créé et restauré entre cet autre processus PCB et le processus clone PCA'.

De façon à permettre à l'application redistribué de continuer à fonctionner correctement, l'application de redistribution de fonctionnement

va également créer pour le processus clone PCB une version virtualisée de tout ou partie des ressources utilisées par le processus original PCA, ou d'une copie de ces ressources. Une telle virtualisation peut s'appliquer par exemple à des identificateurs de processus (PID), ou à des identités de descripteurs de fichiers.

Selon les besoins, l'application de redistribution de fonctionnement pourra alors supprimer le processus original PCA sans interrompre la continuité de fonctionnement de l'application redistribuée ni les services fournis.

Une telle redistribution partielle peut en particulier être utilisée pour ajuster le fonctionnement de l'application redistribuée, en déplaçant certains processus vers d'autres nœuds de façon à modifier la répartition de la charge de travail au sein du cluster, par exemple en vue d'améliorer les performances. Cette charge de travail peut être par exemple du calcul, ou des accès fichiers, ou des communications réseau internes au cluster ou avec l'extérieur. Une redistribution partielle peut aussi être utilisée pour libérer un nœud ou une ligne de communication au sein du cluster, par exemple pour effectuer des interventions sur le matériel qui le constitue.

La figure 2 illustre plus précisément le procédé de gestion de fonctionnement cité ci-dessus.

Ce procédé est mis en œuvre par un processus contrôleur et appliqué sur un processus à gérer, ou processus cible, sur lequel il réalise un mécanisme d'injection d'instructions de programme. Sur cette figure, en regard de certaines étapes ou groupes d'étapes, sont illustrées graphiquement certaines opérations réalisées par l'étape concernée : le rectangle vertical représente la mémoire d'exécution ME contenant les instructions exécutées par le processus cible, le groupe de rectangles sur sa droite représente les registres de travail R utilisés par ce processus, et le triangle sur sa gauche représente le pointeur d'exécution PE du processus au sein de la mémoire d'exécution.

Dans la première étape 201 illustrée, le processus contrôleur prend le contrôle du processus cible, par exemple par une commande « attach » basée sur la routine « ptrace ».

En une étape 202, le processus contrôleur interromps l'exécution du processus cible, et défini une zone réattribuée 2030, ou « scratch area », au sein de la mémoire d'exécution de ce processus cible.

Le processus contrôleur réalise alors 203 une lecture du contenu de la zone réattribuée SA, de la position du pointeur d'exécution PE, et de l'état des registres de travail R, et réalise une sauvegarde 204 de l'état initial de ces éléments.

Le processus contrôleur vérifie 205 que la zone réattribuée SA est suffisamment grande pour réaliser les opérations suivantes. Dans le cas contraire, il peut effectuer 206 un adressage (mapping) de cette zone selon des moyens connus, pour la faire correspondre à un autre espace mémoire plus important, dit zone de mapping, déterminé en dehors de la mémoire d'exécution ME du processus cible. Cette zone de mapping peut alors être utilisée par le processus cible en lieu et place de la zone réattribuée.

Ensuite 207, le processus contrôleur écrit à l'intérieur de la zone réattribuée SA le code IIJ correspondant aux instructions de programme à injecter, et écrit une instruction d'interruption (breakpoint) à la fin de la zone réattribuée SA.

Ensuite 208, le processus contrôleur peut écrire dans la zone réattribuée SA des données ARJ correspondant aux éventuels arguments que doivent utiliser les instructions IIJ.

Ensuite 209, le processus contrôleur modifie l'état des registres de travail R pour leur donner les valeurs RIJ correspondant à l'exécution des instructions à injecter IIJ.

Le processus contrôleur va alors 210 positionner le pointeur d'exécution PE sur la première instruction IIJ du mécanisme injecté et lancer l'exécution du processus cible.

Le processus cible exécute alors 211 les instructions IIJ du mécanisme injecté, par exemple des appels système réalisant une analyse ou une modification de la structure des ressources du processus cible. Selon sa nature, l'exécution du mécanisme injecté peut recevoir des données en retour, qui seront mémorisées dans la zone réattribuée SA ou dans ses registres de travail R, par exemple les réponses retournées par le système d'exploitation aux appels système compris dans le mécanisme injecté.

Lorsque 212 le pointeur d'exécution PE arrive à l'instruction d'interruption écrite précédemment 207, le processus cible s'interromps à nouveau et rappelle le processus contrôleur.

Le processus contrôleur va alors 213 recueillir les résultats de l'exécution du mécanisme injecté, sous la forme de données de résultat lues dans la zone réattribuée SA et dans les registres de travail R, et sauvegarder ces données de résultats indépendamment de l'environnement d'exécution du processus cible.

Ensuite 241, le processus contrôleur utilise les données d'état initial sauvegardées 204 précédemment pour écrire dans la zone réattribuée SA et les registres de travail R et les remettre dans l'état où ils étaient lors de l'interruption initiale 202.

L'espace mémoire d'exécution est ainsi restauré dans l'état où il se trouvait avant l'injection des instruction IIJ. L'opération d'injection peut ainsi être considérée comme provisoire ou temporaire, ce qui évite de polluer le processus cible où l'application qui l'utilise.

Le processus contrôleur peut alors 215 repositionner le pointeur d'exécution PE sur l'instruction ce qui était initialement la prochaine à exécuter, et relancer le processus cible.

Une fois le processus cible à nouveau en exécution, le processus contrôleur le libère de son contrôle, par exemple par une instruction ou une commande « detach », basée sur la routine « ptrace » de façon similaire à la commande « attach ».

La figure 3 illustre l'utilisation du procédé de gestion de fonctionnement selon l'invention pour réaliser une opération de capture de l'état d'un processus, dit processus capturé, et de son environnement d'exécution, par un processus contrôleur.

Dans la première étape représentée 301, le processus contrôleur commence par prendre le contrôle du processus capturé, par exemple par une instruction « attach » basée sur la routine « ptrace ». Le processus contrôleur peut alors interrompre l'exécution du processus capturé lors de cette étape et suspendre tout ou partie des ressources qu'il utilise.

Une étape suivante 302 consiste à réaliser une introspection de l'environnement d'exploitation du processus capturé pour établir une liste

303 des ressources de cet environnement d'exécution. Le processus contrôleur analyse la structure des ressources auxquelles il a accès.

La plupart de ces ressources sont directement accessibles par le processus contrôleur, par exemple par l'instruction de pseudo système de fichiers « /proc ».

Ainsi, l'instruction

« /proc/pid/fd » : fournit la liste des descripteurs de fichiers (fd) actuellement ouverts donc à sauvegarder, pour le processus concerné (pid).

« /proc/pid/maps » : fournit l'organisation et l'adressage des segments de mémoire utilisés ;

Une fois qu'il a identifié 304 les ressources qui ne lui sont pas directement accessibles, le processus contrôleur établit une liste d'instructions à injecter dans le processus capturé pour accéder à ces ressources, par exemple sous la forme d'une liste d'appels système 305 et de leurs paramètres.

En une étape 306 récursive, le processus contrôleur injecte chaque instruction ou groupe d'instruction de cette liste et en recueille les données de résultats, selon le procédé de gestion de fonctionnement décrit plus haut. Par cette injection d'appels système, le processus contrôleur obtient des données 307 représentant la structure des ressources qui ne lui étaient pas directement accessibles.

Pour l'introspection de certaines ressources dont la structure n'est pas directement accessible par un appel système au sein d'un seul processus cible, cette étape 306 met en œuvre un procédé d'introspection multi-processus avec injection d'instructions systèmes. Ce procédé réalise plusieurs opérations d'injection coordonnées entre elles, appliquées à plusieurs processus cibles. Les opérations d'injections introduisent des modifications dans cette ressource à travers au moins l'un de ces processus cibles. Les résultats de ces opérations sont alors comparés entre eux pour obtenir une information portant sur le mode de fonctionnement de la ressource introspectée.

A partir de la structure obtenue par introspection directe 302 ou par injection d'appels système 306, le processus contrôleur peut alors capturer 308 le contenu de ces même ressources et le sauvegarder 310 pour

constituer un état de point de reprise 311, c'est-à-dire une image de l'état du processus capturé.

Ainsi, l'instruction

« /proc/pid/mem » permet de lire le contenu de l'espace mémoire sous la forme de fichier en accès lecture.

« ptrace(PT_GETREGS,...) » permet d'accéder aux registres de travail

Le processus contrôleur relance alors l'exécution du processus capturé et le libère 312 de son contrôle, par exemple par une commande « detach », basée sur la routine « ptrace » de façon similaire à la commande « attach ».

Si nécessaire, la phase d'injection d'appels système 306 peut également être utilisée pour obtenir le contenu ou l'état de certaines ressources, en injectant les instructions de lecture correspondantes.

Ci-dessous figurent, à titre d'exemple en langage C pour un environnement POSIX, des instructions de programme utilisées dans un processus contrôleur PC1 pour réaliser une prise de contrôle 301 d'un processus dont l'identifiant est « pid », c'est-à-dire dont la valeur est contenue dans la variable nommée « pid ».

Instruction de chargement de la fonction « ptrace » :

```
#include <sys/ptrace.h>
```

Définition de la fonction « attach » qui réalise cette prise de contrôle :

```
int attach(int pid)
{
        int status;

        /* Prise de controle d'un processus par ptrace. Le processus
         * est défini par son process id
         */
        ptrace(PTRACE_ATTACH, pid, 0, 0);

        /* lorsque le process est bloqué, SIGSTOP nous est renvoyé */
        waitpid(pid, &status, 0);
        if (WIFSTOPPED(status)) /* STOP est dans le masque des signaux */
                return 0K;
```

```
        return ERROR;
    }
```

Ci-dessous figurent, à titre d'exemple en langage C pour un environnement POSIX, des instructions de programme réalisant une injection d'instructions destinée à capturer la position du pointeur d'écriture d'un descripteur de fichier ouvert par le processus capturé.

Déclaration d'une fonction nommée « ptrace_syscall », utilisée pour injecter n'importe quel appel système « syscall » associé à des arguments « argc », dans un processus dont l'identifiant est « pid » :

```
int ptrace_syscall(pid_t pid, pid_t *tpid, int scratch, int syscall, int argc, ...);
```

Définition d'une macro utilisant la fonction « ptrace_syscall », à utiliser pour réaliser l'injection de l'appel système « l_seek » dans le processus dont l'identifiant est « p » :

```
#define PT_LSEEK(p, fd, off, w) \
    ptrace_syscall(p, 0, 0, SYS_lseek, 3,  \
            0, 0, fd,          \
            0, 0, off,         \
            0, 0, w)
```

Définition d'une fonction, utilisée dans l'application de redistribution de fonctionnement, appelant la macro « PT_SEEK » pour capturer la position du pointeur d'écriture, en injectant l'appel système « lseek », assorti du paramètre « SEEK_CUR », dans le processus dont l'identifiant est « pid » :

```
int get_file_pos(int pid,        /* process id du programme attaché */
                 int fd) /* descripteur de fichier ouvert par pid */
{
        int file_pos = PT_LSEEK(pid, fd, 0, SEEK_CUR);
        return file_pos;
```

}Les figures 5 et 6 illustrent un exemple d'un procédé d'introspection multi-processus, appliqué à l'analyse d'un descripteur de fichier. Lorsqu'un processus fils utilise descripteur de fichier hérité d'un processus père, les deux processus père et fils utilisent deux descripteurs différents, mais qui pointent tous les deux sur le même fichier ou container de données doté

d'un unique pointeur de position. Il s'agit alors de deux instances différentes d'un même objet initial, appelées descripteurs « partagés », par opposition à des descripteurs « séparés ». Or, il peut être utile de sauvegarder la nature de tels descripteurs de fichiers dans le cadre d'une capture d'état, de façon à garder une même cohérence au sein des processus qui seront ultérieurement restaurés à partir de cette capture.

Le procédé d'introspection multi-processus est alors utilisé pour déterminer si deux descripteurs de fichiers FDA et FDB, utilisés par deux processus différents PA et PB et pointant sur des fichiers FA et FB, sont des descripteurs séparés ou partagés.

En une étape 501, un processus contrôleur PC1 injecte une instruction d'appel système au sein du premier processus cible PA. Cet appel système réalise une lecture ptA0 de la position du pointeur de lecture/écriture du descripteur de fichier FDA de ce premier processus cible PA.

Ce processus contrôleur PC1 injecte des instructions d'appel système au sein du deuxième processus cible PB. En une étape 502, l'un de ces appels système réalise tout d'abord une lecture ptB0 de la position du pointeur de lecture/écriture du descripteur de fichier FDB de ce deuxième processus cible PB.

En une étape 503, un autre de ces appels système, par exemple une instruction « lseek » réalise ensuite une modification de la position de ce même pointeur.

En une étape 504, le processus contrôleur PC1 injecte une instruction d'appel système au sein du premier processus cible PA. Cet appel système réalise une nouvelle lecture ptA1 de la position du pointeur de lecture/écriture du descripteur de fichier FDA de ce premier processus cible PA.

En une étape 505, le processus contrôleur PC1 compare alors les valeurs ptA0 et ptA1 obtenues par les deux lectures de position du pointeur du premier descripteur FD1.

Si ces valeurs sont égales, alors cela signifie que ces deux descripteurs FDA, FDB utilisent le même pointeur, et sont donc des

descripteurs partagés. En une étape 506, le processus contrôleur PC1 mémorise alors une donnée représentant cette information.

En une étape 507, le processus contrôleur PC1 injecte ensuite une instruction d'appel système au sein d'un des deux processus cibles, par exemple PB, pour ramener le pointeur à sa position initiale ptB0.

Si ces valeurs sont différentes, cela signifie que ces deux descripteurs FDA, FDB n'utilisent pas le même pointeur, et sont donc des descripteurs séparés. En une étape 507, le processus contrôleur PC1 mémorise alors une donnée représentant cette information.

En une étape 508, le processus contrôleur PC1 injecte ensuite une instruction d'appel système au sein du deuxième processus cible PB pour ramener son pointeur à sa position initiale ptB0.

Dans les deux cas, le pointeur modifié est ramené à sa position initiale, et le procédé est donc bien transparent pour les deux processus cible.

La figure 4 illustre l'utilisation du procédé de gestion de fonctionnement selon l'invention pour réaliser une opération de mise à jour ou de restauration d'un processus, dit processus de reprise, et de son environnement d'exécution, par un processus contrôleur.

Cette figure représente une opération de restauration, comprenant une partie 401, 402, 403 de création du processus de reprise.

Le processus contrôleur déclenche cette création en initialisant 401 un nouveau processus, dit processus de reprise, sous son contrôle (technique de « forking »), puis en utilisant une instruction « ptrace(TRACEMEM,...) » avant d'en lancer l'exécution.

Le processus de reprise démarre alors normalement en chargeant 402 les différentes ressources comme lors d'un démarrage à froid classique.

A ce stade commence le procédé proprement dit de mise à jour de l'état d'un processus de reprise, c'est-à-dire le procédé qui peut être utilisé sur un processus de reprise existant déjà.

Si la mise à jour se fait dans la foulée d'un démarrage de processus de reprise, ce processus de reprise s'arrête 404 immédiatement après son chargement, du fait de son mode de lancement, et rappelle le processus contrôleur.

Si la mise à jour se fait sur un processus de reprise préexistant, le processus contrôleur commence par prendre le contrôle 405 du processus capturé, par exemple par une instruction « attach » basée sur la routine « ptrace ».

Le processus contrôleur réalise 406 alors une sélection et une lecture de données sauvegardées précédemment et constituant un état de point de reprise. A partir du contenu de cet de point de reprise, le processus contrôleur évalue les modifications de structure et de contenu à réaliser dans l'environnement d'exécution du processus de reprise tel qu'il se trouve pour l'amener à l'état de point de reprise sélectionné.

Si certaines des modifications de structure sont possibles directement depuis le processus contrôleur, celui-ci les réalise par lui-même 407.

Pour les modifications de structure qui ne lui sont pas accessibles, le processus contrôleur prépare une liste d'appels système qu'il injecte 408 dans le processus de reprise, selon le procédé de gestion de fonctionnement de l'invention.

Cette injection est utilisée par exemple pour modifier l'adressage et la cartographie des segments de mémoire utilisés, en injectant un ou plusieurs appels systèmes « mmap ». Le même principe est utilisé pour tout ou partie des ressources système qui doivent être recrées pour arriver à un état identique à l'état de point de reprise sélectionné. Ces ressources système sont par exemples des ressources de type « file », « socket », « pipe », « timer », « terminal control », etc.

Une fois que les structures de ressources sont adéquates, le processus contrôleur réalise 409 une écriture de ces ressources système en fonction des données de l'état de point de reprise, pour amener le processus de reprise à l'état où se trouvait le processus capturé lors de l'établissement du pont de reprise sélectionné.

Le processus contrôleur relance 410 alors l'exécution du processus de reprise et le libère 411 de son contrôle, par exemple par une commande « detach », basée sur la routine « ptrace » de façon similaire à la commande « attach ».

Si nécessaire, la phase d'injection d'appels système 408 peut également être utilisée pour écrire le contenu ou l'état de certaines ressources, en injectant les instructions de lecture correspondantes.

Du fait qu'elle s'opère depuis un processus extérieur au processus de reprise, cette opération de restauration est bien plus simple et performante que si elle devait se faire par des opérations prévues à l'intérieur même de ce processus de reprise.

Ci-dessous figurent, à titre d'exemple en langage C pour un environnement POSIX, des instructions de programme réalisant une injection d'instructions destinée à restaurer la position du pointeur d'écriture d'un descripteur de fichier ouvert par ou pour le processus de reprise.

Ces instructions utilisent les mêmes fonction « ptrace_syscall » et macro « PT_SEEK » que celles décrites plus haut pour l'opération de capture.

Définition d'une fonction, utilisée dans l'application de redistribution de fonctionnement, appellant la macro « PT_SEEK » pour restaurer la position du pointeur d'écriture, en injectant l'appel système « lseek », assorti du paramètre « SEEK_SET », dans le processus dont l'identifiant est « pid » :

```
int set_file_pos(int pid,
        int fd,
        int filepos) /* extrait du checkpoint ou point de reprise */
{
    return PT_LSEEK(pid, fd, filepos, SEEK_SET);
}
```

Dans le cas d'applications comprenant plusieurs processus, ou tâches, susceptibles de s'exécuter simultanément, l'établissement d'un de point de reprise peut nécessiter de capturer l'état de plusieurs de ces processus. Pour cela, l'utilisation d'un ou plusieurs processus contrôleurs extérieurs aux processus à capturer est un avantage qu'apporte le procédé selon l'invention.

Dans ce cas, l'application de redistribution de fonctionnement réalise une opération de capture selon l'invention sur plusieurs processus capturés, de façon à synchroniser ou à coordonner l'interruption initiale 301 de

chacune des opérations de capture et la suspension des ressources concernées.

Lors d'une capture de plusieurs processus, certaines données en cours de transmission entre plusieurs processus peuvent se trouvées « figées » au sein du mécanisme logiciel interprocessus IPC gérant ces transmissions, par exemple l'objet logiciel « Inter Process Communication » dans un environnement de type Unix.

De façon à éviter de perturber la cohérence de l'état de point de reprise qui sera sauvegardé, l'application de redistribution de fonctionnement utilise le procédé de gestion de fonctionnement selon l'invention pour injecter dans chacun des processus interrompus les appels systèmes pour gérer ces données en cours de transmission. Il pourra s'agir par exemple de purger les files d'attente (pipe) de l'IPC des données non traitées dans le cadre d'une opération de capture d'état de processus lors d'un point de reprise (checkpoint), ou de restaurer ces mêmes données dans le cas d'une mise à jour de processus.

En effet, dans une situation de capture de l'état de plusieurs processus communiquant entre eux, lorsqu'un processus est suspendu pour capture, il peut y avoir des données en attente au sein de l'agent interprocessus IPC, à destination de ce processus suspendu. Une fois que tous les processus à capturer sont interrompus, pour chaque processus à capturer, l'opération de capture comprend alors en outre une analyse et une mémorisation de toutes les données de communications, ou paquets, qui lui ont destinées mais n'ont pas encore été reçues. Dans les systèmes où cet agent interprocessus est géré par le système, par exemple dans un module kernel pour le cas d'Unix, il est avantageux de ne pas avoir à intervenir dans le système. Le processus contrôleur PC1 utilise alors le procédé de gestion de fonctionnement selon l'invention pour injecter dans le processus en cours de capture des appels système qui vont demander une lecture de ces données de communication en cours de transit. Le processus contrôleur récupère alors ces données et les sauvegarde au sein de l'état de point de reprise.

Dans une situation de restauration, lorsque tous les processus de reprise sont suspendus, le processus contrôleur PC2 utilise également le

processus de gestion selon l'invention pour injecter dans chaque processus de reprise des appels systèmes qui vont écrire au sein de l'agent interprocessus IPC les paquets en transit qui avaient été mémorisés dans l'état de point de reprise.

Par ailleurs, lorsqu'une application comprend plusieurs processus, certains de ces processus peuvent avoir entre eux des relations d'héritage. C'est-à-dire qu'un processus « fils » peut avoir été créé à partir d'un processus « père », et hériter par cette relation d'héritage de certaines caractéristiques ou ressources de son environnement d'exploitation, en particulier de type « file descriptor ».

Lors de la capture des processus d'une application, le processus contrôleur PC1 va utiliser le processus de gestion selon l'invention pour injecter dans chaque processus capturé des appels système qui vont analyser ses éventuelles relations d'héritage avec un ou plusieurs autres processus. Les résultats de ces analyses seront alors sauvegardées au sein de l'état de point de reprise en cours de constitution.

Lors de la restauration de ces mêmes processus, le processus contrôleur PC1 va utiliser le processus de gestion selon l'invention pour injecter dans chaque processus de reprise des appels système qui vont recréer ces mêmes relations d'héritage qui avaient été mémorisés dans l'état de point de reprise.

Bien sûr, l'invention n'est pas limitée aux exemples qui viennent d'être décrits et de nombreux aménagements peuvent être apportés à ces exemples sans sortir du cadre de l'invention.

REVENDICATIONS

1. Procédé pour gérer le fonctionnement d'une application logicielle comprenant au moins un premier processus logiciel, dit processus cible, s'exécutant sur au moins un ordinateur et dans un environnement d'exécution comprenant au moins un espace mémoire d'exécution,
caractérisé en ce qu'il comprend une opération d'injection temporaire d'au moins une instruction exécutable dans l'espace mémoire d'exécution du processus cible, par au moins un deuxième processus logiciel, dit processus contrôleur, extérieur à l'application et apte à agir sur le déroulement du processus cible, cette instruction exécutable réalisant une analyse ou une modification de l'environnement d'exécution de ce processus cible.

2. Procédé selon la revendication 1, caractérisé en ce que l'opération d'injection comprend des étapes de :

- interruption de l'exécution du processus cible (202) par le processus contrôleur ;
- écriture (207) par le processus contrôleur dans une partie, dite zone réattribuée, de l'espace mémoire d'exécution du processus cible, d'instructions injectées réalisant le mécanisme d'analyse ou de modification ;
- exécution (211), par le processus cible, de ces instructions injectées ;
- restauration (214) par le processus contrôleur, par écriture dans la zone réattribuée, des instructions du processus cible qui y étaient mémorisées avant l'interruption (202) ;
- exécution (215) de la suite des instructions du processus cible.

3. Procédé selon l'une des revendications 1 ou 2, caractérisé en ce qu'il réalise une opération d'introspection d'au moins deux processus introspectés, chacun de ces processus introspectés (PA, PB) utilisant une première ressource (FDA, respectivement FDB) comportant elle-même un pointeur (IdPtA, respectivement IdPtB) désignant une deuxième ressource (FA, FB) comportant elle-même un attribut (ptA, ptB) qui est accessible

audit processus à travers ledit pointeur , le procédé comprenant les étapes suivantes :

- injection (501, 502) par le processus contrôleur (PC1) dans chacun des deux processus introspectés (PA, PB) d'au moins une instruction système réalisant une lecture initiale de la valeur (ptA0, respectivement ptB0) de l'attribut (ptA, ptB) de la deuxième ressource (FA, FB) correspondant à chacun desdits processus introspectés ;
- injection (503) par le processus contrôleur (PC1) dans l'un des deux processus introspectés, dit processus de test (PB), d'au moins une instruction système réalisant une modification de la valeur (ptB0) de l'attribut (ptB) de la deuxième ressource (FB) correspondant audit processus test (PB) ;
- injection (504) par le processus contrôleur (PC1) dans l'autre processus introspecté, dit processus témoin (PA), d'au moins une instruction système réalisant une deuxième lecture de la valeur (ptA1) de l'attribut (ptA) de la deuxième ressource (FA) correspondant audit processus témoin (PA) ;
- comparaison (505) par le processus contrôleur (PC1) de la valeur de deuxième lecture (ptA1) avec la valeur de lecture initiale (ptA0) dudit processus témoin (PA) ;
- mémorisation (506, 508) par le processus contrôleur (PC1) d'une donnée représentant le résultat de ladite comparaison et injection (507, 509)) par le processus contrôleur (PC1) dans le processus de test (PB), d'au moins une instruction système réalisant une modification de la valeur (ptB0) de l'attribut (ptB) de la deuxième ressource (FB) correspondant audit processus test (PB), de façon à lui redonner sa valeur (ptB0) de lecture initiale.

4. Procédé selon l'une des revendications 1 à 3, caractérisé en ce qu'il réalise une opération de capture de l'état du processus cible, dit processus capturé (PCA), comprenant des étapes de :

- prise de contrôle (301) du processus capturé par un processus contrôleur ;

- injection (306) par le processus contrôleur (PC1) dans le processus capturé d'au moins une instruction d'appel système réalisant une analyse (307) de la structure de l'environnement d'exécution du processus capturé ;
- mémorisation (310) ou transmission de données de résultat (311) représentant le résultat de cette analyse et restauration de l'espace mémoire du processus capturé ;
- exécution (312) de la suite des instructions du processus capturé.

5. Procédé selon la revendication 4, caractérisé en ce qu'il réalise une opération de capture de l'état d'au moins deux processus (PCA, PCB) de cette application, l'interruption de ces deux processus se faisant soit de façon simultanée soit en des points de leurs déroulements respectifs dont l'un est calculé en fonction de l'autre.

6. Procédé selon l'une des revendications 4 ou 5, caractérisé en ce que le processus capturé (PCA) échange des données de communication avec au moins un autre processus (PCB) par l'intermédiaire d'au moins un agent logiciel interprocessus (IPC) extérieur à l'application, l'opération de capture comprenant en outre des étapes de :

- injection, par le processus contrôleur dans le processus capturé d'au moins une instruction d'appel système réalisant la lecture dans l'agent interprocessus d'au moins une donnée de communication en provenance d'un autre processus de l'application et non encore reçue par le processus capturé ;
- mémorisation ou transmission de cette donnée de communication en tant que donnée de résultat.

7. Procédé selon l'une des revendications 4 à 6, caractérisé en ce que l'environnement d'exécution du processus capturé (PCA) supporte la transmission de caractéristiques entre processus par des relations d'héritage, l'opération de capture comprenant en outre des étapes de :

- injection, par le processus contrôleur dans le processus capturé d'au moins une instruction d'appel système réalisant une analyse des

relations d'héritage du processus capturé avec au moins un autre processus de l'application ;

- mémorisation ou transmission de données de résultat représentant les relations d'héritage du processus capturé.

8. Procédé selon l'une des revendications 1 ou 2, caractérisé en ce qu'il réalise une opération de restauration, par un processus contrôleur (PC2) à partir de données dites de reprise, de l'état d'au moins un processus d'application logicielle, dit processus de reprise (PCA'), l'opération de restauration comprenant des étapes de :

- interruption (404, 405) de l'exécution du processus de reprise par le processus contrôleur (PC2) ;
- injection (408) par le processus contrôleur dans le processus de reprise d'au moins une instruction d'appel système créant ou modifiant la structure d'au moins un objet logiciel appartenant à l'environnement d'exécution du processus de reprise, en fonction des données de reprises ;
- écriture (409), à partir des données de reprise, de l'espace mémoire d'exécution du processus de reprise ;
- lancement (410) du processus de reprise et exécution (411) de la suite de ses instructions.

9. Procédé selon la revendication 8, caractérisé en ce que l'environnement d'exécution du processus de reprise supporte l'échange de données de communication entre plusieurs processus (PCA', PCB'), par l'intermédiaire d'au moins un agent logiciel interprocessus (IPC) extérieur à l'application, l'opération de restauration comprenant en outre une étape de :

- injection, par le processus contrôleur dans le processus capturé d'au moins une instruction d'appel système réalisant, à partir des données de reprise, l'écriture au sein de l'agent interprocessus (IPC) d'au moins une donnée représentant une donnée de communication à destination du processus de reprise.

10. Procédé selon l'une des revendications 8 ou 9, caractérisé en ce que l'environnement d'exécution du processus de reprise (PCA') supporte la transmission de caractéristiques entre processus par des relations d'héritage, l'opération de restauration comprenant en outre une étape de :

- injection, par le processus contrôleur dans le processus de reprise d'au moins une instruction d'appel système créant ou modifiant, à partir des données de reprise, au moins une relations d'héritage du processus de reprise avec au moins un autre processus de l'application.

11. Procédé selon l'une des revendications précédentes, caractérisé en ce qu'il réalise une réplication d'au moins un processus de l'application, dit processus original, en un processus clone, et comprend les étapes suivantes :

- capture de l'état du processus original par un procédé selon l'une des revendications 2 à 6 ;
- utilisation de données de résultat, issues de la capture, pour mémoriser un objet logiciel appelé point de reprise, représentant un état de ce processus original en un point de son exécution ;
- utilisation de données du point de reprise pour restaurer au moins un processus clone dans un état reproduisant l'état du processus original.

12. Procédé selon la revendication 11, caractérisé en ce qu'il réalise une redistribution de tout ou partie d'une application logicielle dite redistribuée, exécutée dans une architecture multi-ordinateurs (cluster) et comprenant au moins un processus, dit processus initial, fournissant un traitement de données en étant exécuté à un instant donné sur au moins un ordinateur du cluster, appelé nœud primaire ou opérationnel (OP), d'autres ordinateurs dudit cluster étant appelés nœuds secondaires, cette opération de redistribution comprenant les étapes suivantes :

- réplication d'au moins un processus initial en au moins un processus secondaire exécuté sur un nœud secondaire ;
- basculement de tout ou partie du traitement de données du processus initial vers au moins un processus secondaire.

13. Procédé selon la revendication 12, caractérisé en ce qu'il comprend en outre les étapes suivantes :

- réplication de tous les processus exécutés par le nœud opérationnel en un ou plusieurs processus secondaires exécutés sur au moins un nœud secondaire ;
- basculement de la totalité des traitements de données desdits processus vers au moins un desdits processus secondaires.

14. Procédé selon l'une des revendications 1 à 10, caractérisé en ce qu'il réalise une suspension d'une application logicielle comprenant au moins un processus exécuté sur au moins un ordinateur, cette opération de suspension comprenant les étapes suivantes :

- capture de l'état de l'ensemble des processus de l'application ;
- utilisation de données de résultat, issues de la capture, pour mémoriser un objet logiciel appelé point de reprise, représentant un état de cette application en un point de son exécution ;
- utilisation de données du point de reprise pour restaurer un ou plusieurs processus clones dans un état reproduisant l'état de l'ensemble des processus capturés.

15. Procédé selon l'une des revendications 1 à 10, caractérisé en ce qu'il réalise une fiabilisation du fonctionnement d'une application logicielle, dite application fiabilisée, exécutée dans une architecture multi-ordinateurs (cluster) et fournissant un service déterminé, au moins un processus (PCA) de cette application étant exécuté à un instant donné sur au moins un ordinateur du cluster, appelé nœud primaire ou opérationnel (OP), d'autres ordinateurs dudit cluster étant appelés nœuds secondaires (SB), cette fiabilisation comprenant les étapes suivantes :

- capture par au moins un processus contrôleur (PC1) de l'état de l'ensemble des processus de cette application fiabilisée ;
- utilisation de données de résultat, issues de la capture, pour mémoriser un objet logiciel appelé point de reprise, représentant un état de cette application fiabilisée en un point de son exécution ;

- détection au sein du nœud opérationnel d'une défaillance matérielle ou logicielle affectant le fonctionnement de l'application fiabilisée ;
- utilisation de tout ou partie des données du point de reprise pour restaurer, sur au moins un nœud secondaire, un ou plusieurs processus d'une application de secours en un état reproduisant l'état de l'ensemble des processus de l'application fiabilisée ;
- basculement de tout ou partie du service vers l'application de secours d'au moins l'un desdits noeuds secondaires.

16. Procédé selon la revendication 11, caractérisé en ce qu'il réalise une réplication holistique de l'état d'une application dite originale en une application clone, en utilisant ledit procédé de réplication pour répliquer l'ensemble des processus et ressources de l'application originale en tant que processus et ressources de l'application clone.

17. Procédé selon la revendication 16, caractérisé en ce qu'il réalise une fiabilisation d'une application logicielle dite fiabilisée, exécutée dans une architecture multi-ordinateurs (cluster) et fournissant un service déterminé, au moins un processus de cette application, dit processus fiabilisé (PCA), étant exécuté à un instant donné sur au moins un ordinateur du cluster, appelé nœud primaire ou opérationnel (OP), d'autres ordinateurs dudit cluster étant appelés nœuds secondaires, cette fiabilisation comprenant les étapes suivantes :

- mise en œuvre d'un procédé de réplication holistique pour répliquer, sur au moins un nœud secondaire (SB), une application de secours dans un état identique à celui de l'application fiabilisée ;
- détection au sein du nœud opérationnel d'une défaillance matérielle ou logicielle affectant le fonctionnement de l'application fiabilisée ;
- basculement de tout ou partie du service vers ladite application de secours d'au moins un des nœuds secondaires.

18. Système multi-ordinateurs comprenant une gestion de processus applicatifs mettant en œuvre le procédé selon l'une des revendications 1 à 16.

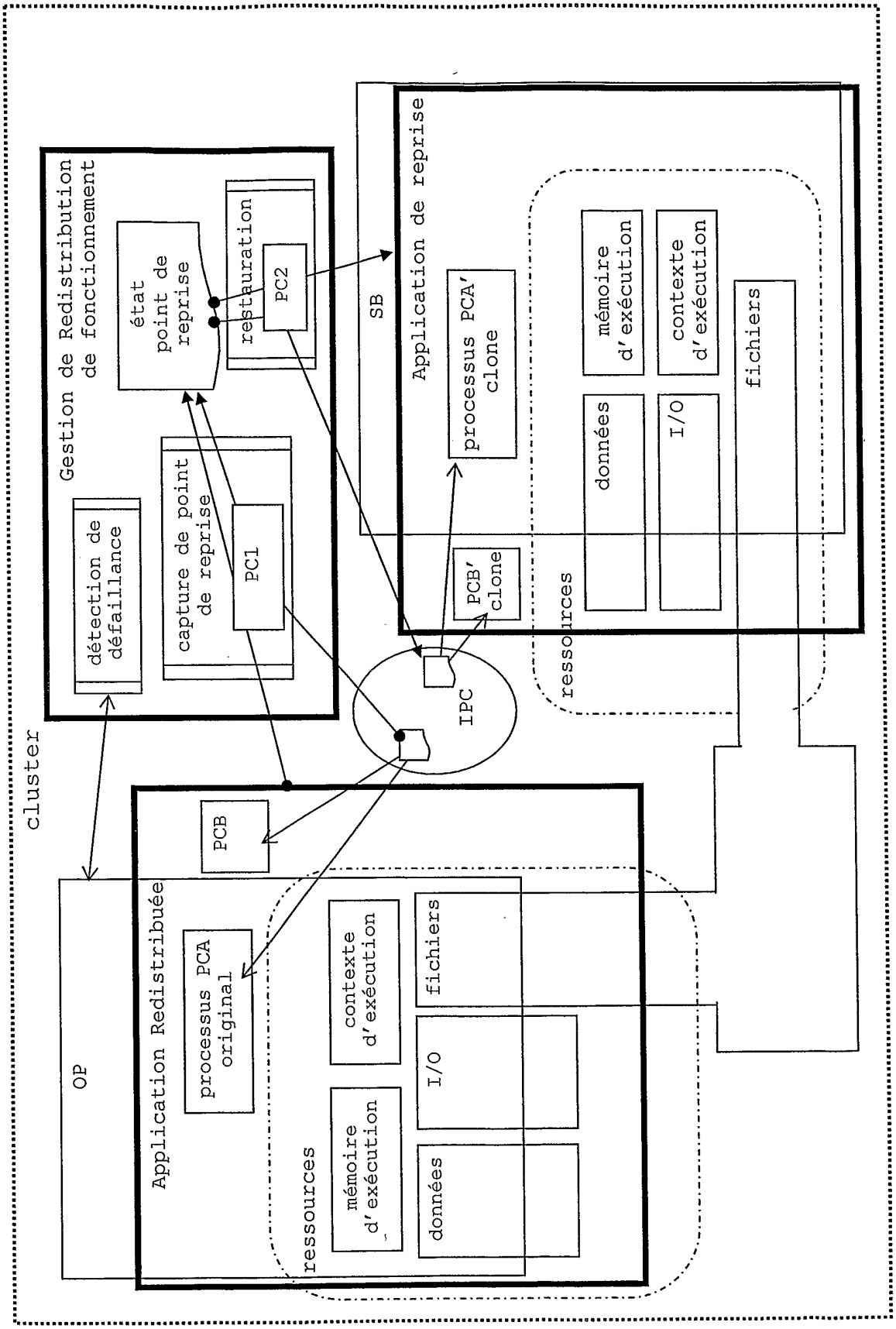

cluster
Gestion de Redistribution de fonctionnement
détection de défaillance
capture de point de reprise
PC1
état point de reprise
restauration
PC2
SB
Application de reprise
processus PCA' clone
mémoire d'exécution
contexte d'exécution
fichiers
données
I/O
PCB' clone
ressources
IPC
OP
PCB
Application Redistribuée
processus PCA original
contexte d'exécution
fichiers
mémoire d'exécution
I/O
données
ressources

Fig.1a

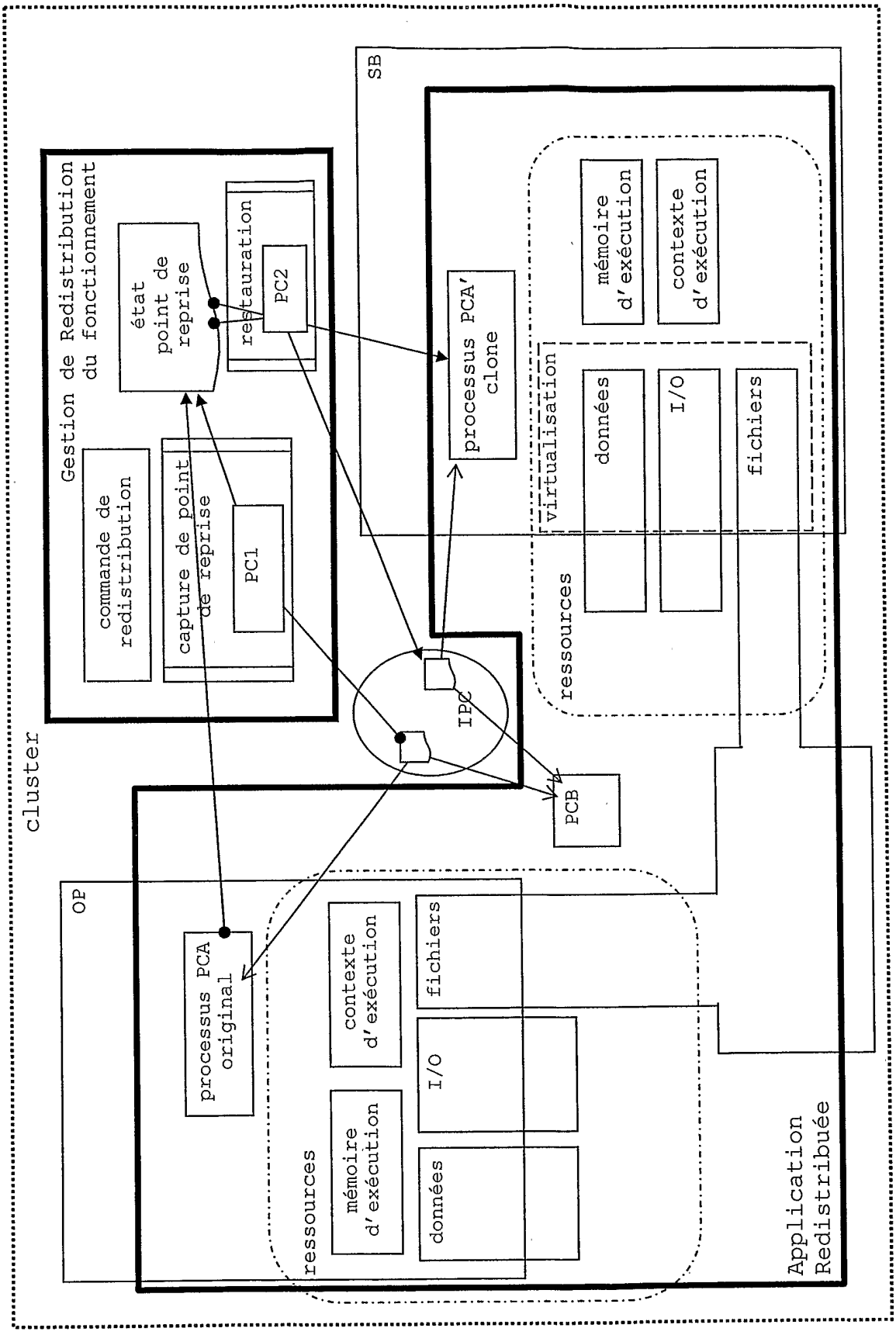

cluster
Gestion de Redistribution du fonctionnement
commande de redistribution
capture de point de reprise
PC1
état point de reprise
restauration
PC2
SB
processus PCA' clone
virtualisation
mémoire d'exécution
contexte d'exécution
données
I/O
fichiers
ressources
IPC
PCB
OP
processus PCA original
contexte d'exécution
fichiers
I/O
mémoire d'exécution
données
ressources
Application Redistribuée

Fig.1b

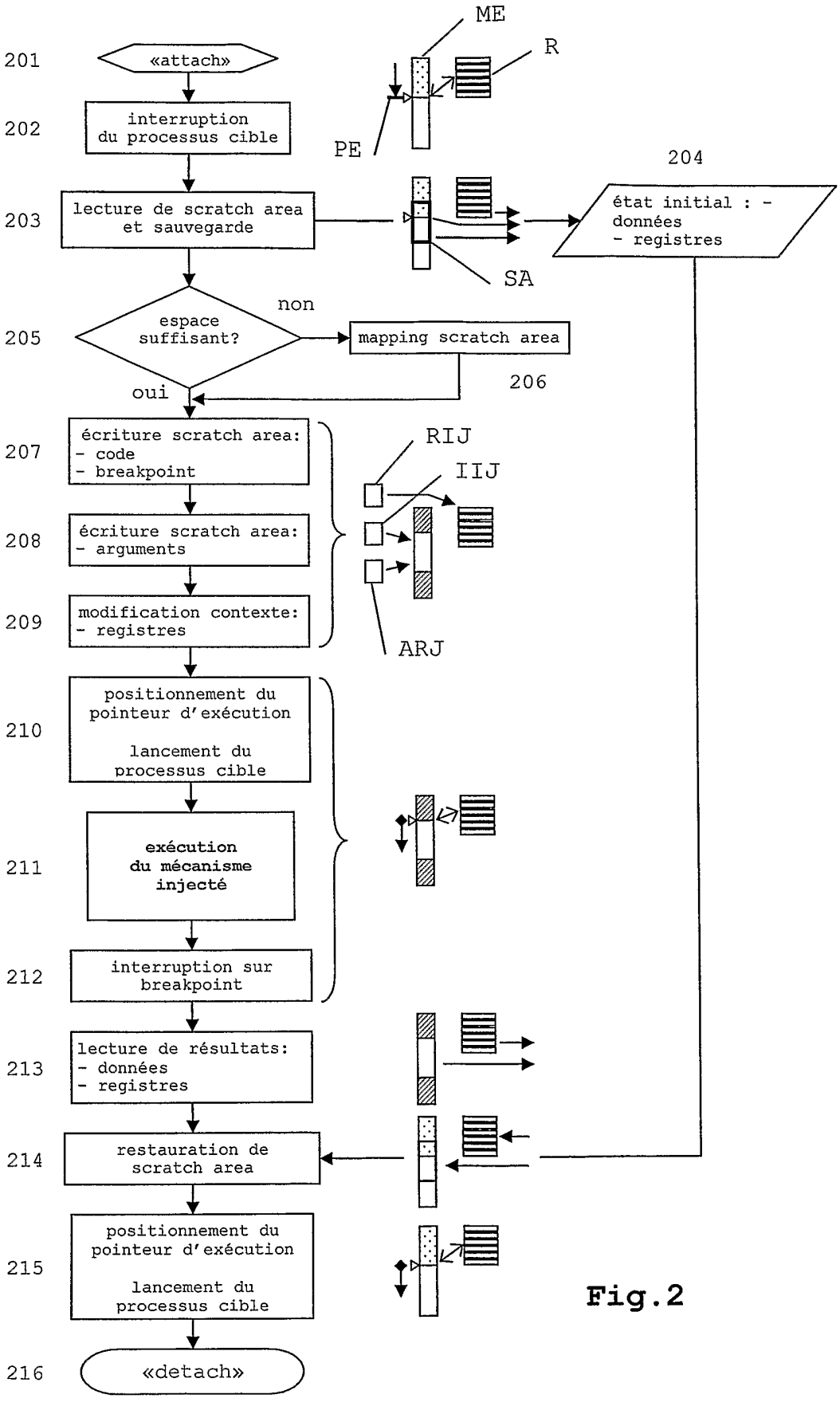
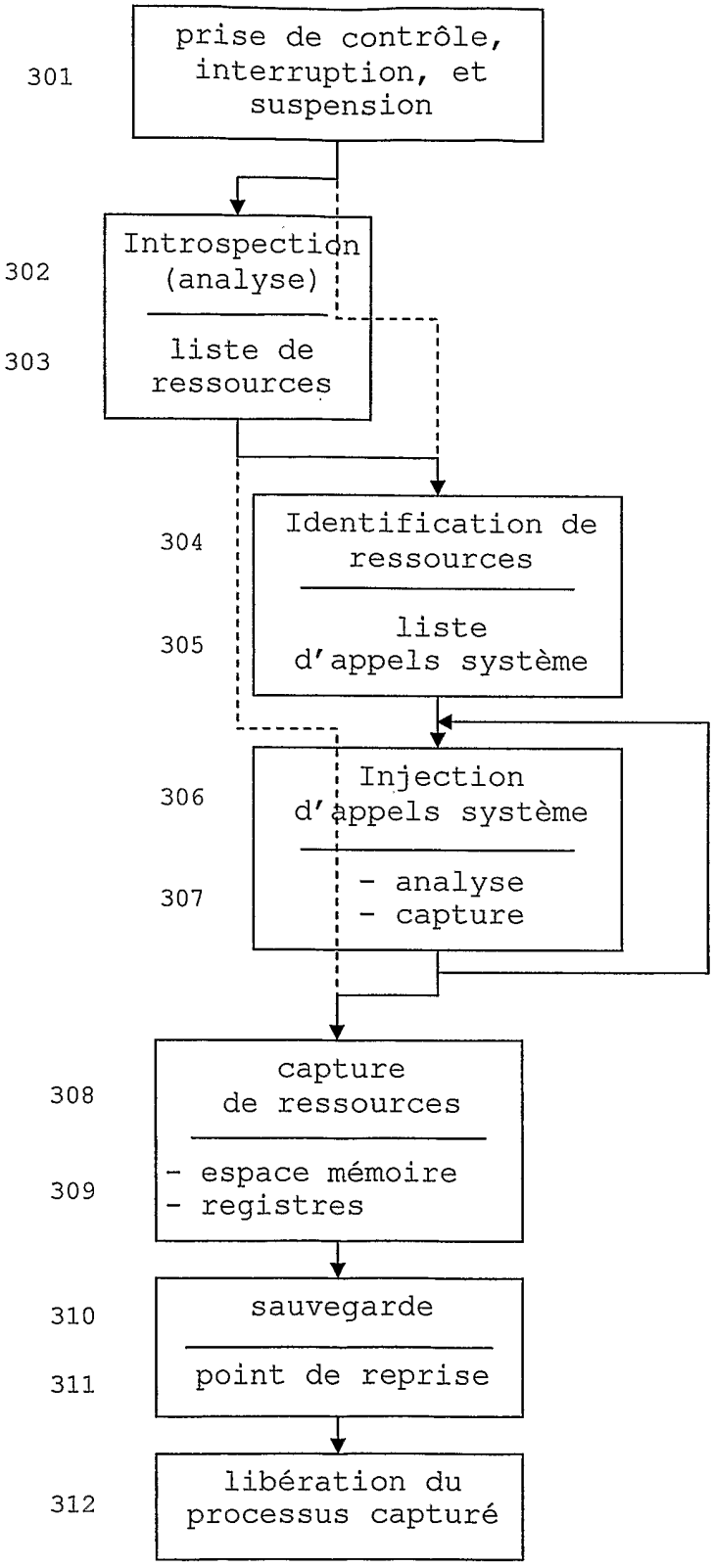

201
«attach»
202
interruption du processus cible
203
lecture de scratch area et sauvegarde
205
espace suffisant?
non
mapping scratch area
206
oui
207
écriture scratch area: - code - breakpoint
208
écriture scratch area: - arguments
209
modification contexte: - registres
210
positionnement du pointeur d'exécution
lancement du processus cible
211
exécution du mécanisme injecté
212
interruption sur breakpoint
213
lecture de résultats: - données - registres
214
restauration de scratch area
215
positionnement du pointeur d'exécution
lancement du processus cible
216
«detach»
ME
R
PE
204
état initial : - données - registres
SA
RIJ
IIJ
ARJ

Fig.2

Fig.3


301
prise de contrôle, interruption, et suspension
302
Introspection (analyse)
303
liste de ressources
304
Identification de ressources
305
liste d'appels système
306
Injection d'appels système
307
- analyse
- capture
308
capture de ressources
309
- espace mémoire
- registres
310
sauvegarde
311
point de reprise
312
libération du processus capturé

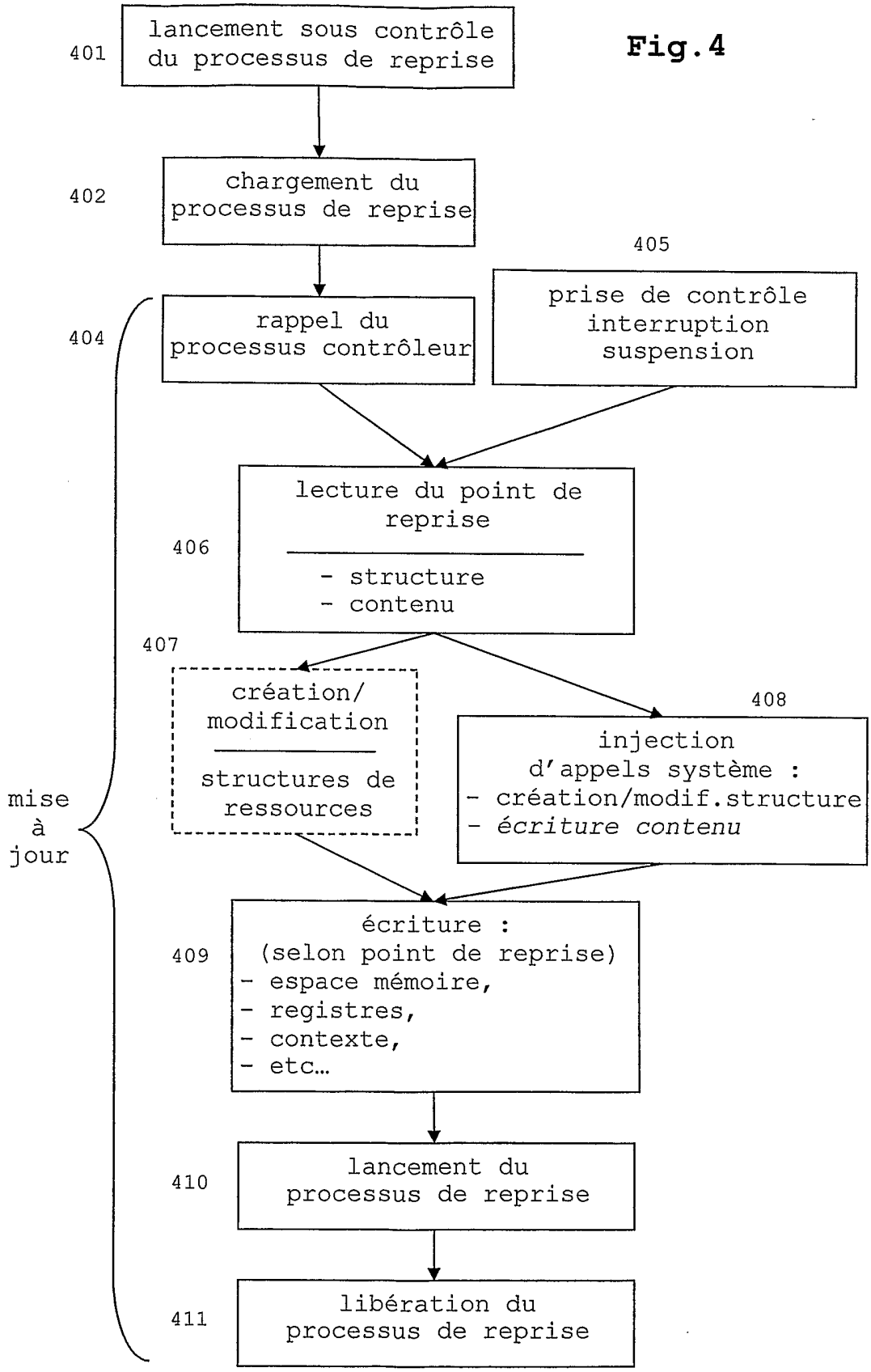

401
lancement sous contrôle du processus de reprise
402
chargement du processus de reprise
405
prise de contrôle interruption suspension
404
rappel du processus contrôleur
406
lecture du point de reprise
- structure
- contenu
407
création/ modification
structures de ressources
408
injection d'appels système :
- création/modif.structure
- écriture contenu
mise à jour
409
écriture :
(selon point de reprise)
- espace mémoire,
- registres,
- contexte,
- etc…
410
lancement du processus de reprise
411
libération du processus de reprise

Fig.4

Fig.5

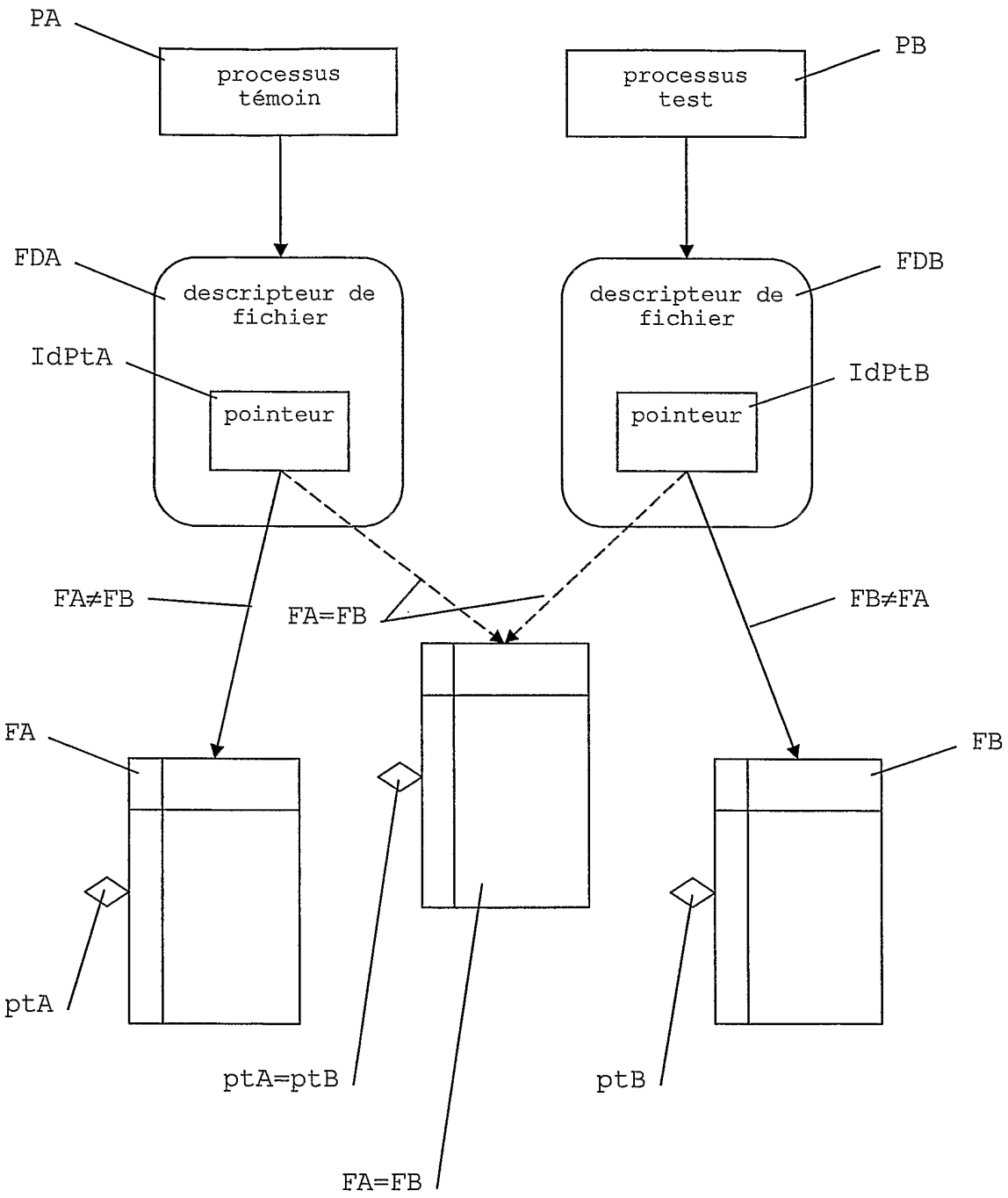
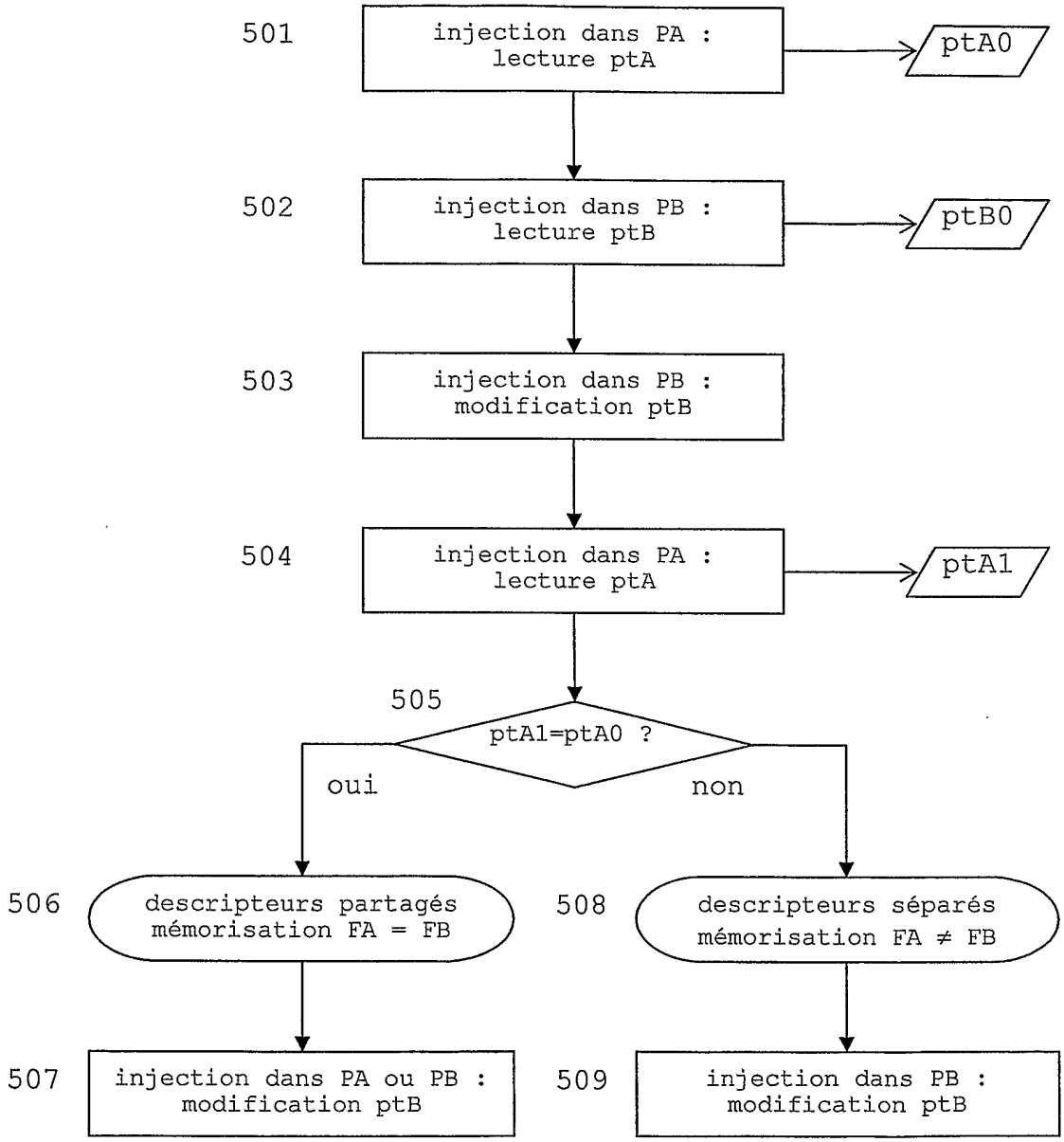

PA
processus témoin
PB
processus test
FDA
descripteur de fichier
FDB
descripteur de fichier
IdPtA
pointeur
IdPtB
pointeur
FA≠FB
FA=FB
FB≠FA
FA
FB
ptA
ptA=ptB
ptB
FA=FB

Fig.6


501
injection dans PA : lecture ptA
ptA0
502
injection dans PB : lecture ptB
ptB0
503
injection dans PB : modification ptB
504
injection dans PA : lecture ptA
ptA1
505
ptA1=ptA0 ?
oui
non
506
descripteurs partagés mémorisation FA = FB
508
descripteurs séparés mémorisation FA ≠ FB
507
injection dans PA ou PB : modification ptB
509
injection dans PB : modification ptB