International Journal of Mathematical Archive-2(5), May - 2011, Page: 716-719
Available online through www.ijma.info ISSN 2229 – 5046

Gestion du TAS: Application sur J2ME

Mostafa HANOUNE, Laila Moussaid

Laboratoire des Technologies de l'Information et Modélisation (TIM)

Faculté des sciences Ben M'Sik, Université Hassan II, Mohammedia, Casablanca, MAROC

E-mail : mhanoune@gmail.com, lailahelp@yahoo.fr

(Recived on: 16-03-11; Accepted on: 06-04-11)

Résumé

Pour gérer automatiquement la mémoire de microéditions telles que les téléphones portables, les PDA.., Java utilise la technique du ramasse-miettes (RM) qui intègre, divers algorithmes selon le cas traité : ramasse-miettes par comptage de références, ramasse-miettes par marquage balayage, ramasse-miettes par recopie… Dans cet article nous présentons une nouvelle approche pour l'optimisation de la mémoire des microéditions. Cette approche hérite l'avantage du RM par marquage-balayage et minimise la limitation du RM par référence. Dans la première partie nous présentons l'environnement J2ME, dans la deuxième partie nous décrivons notre approche proposée et nous conclurons notre article par des perspectives.

Mots Clés: *J2ME, Ramasse miettes, Système embarqué.*

I. INTRODUCTION:

Aujourd'hui, les systèmes embarqués deviennent de plus en plus populaires dans divers domaines professionnels et personnels, la chose qui les rend un objectif pour les chercheurs afin de surmonter leurs problèmes standards tel que l'optimisation de l'énergie et de la mémoire…

Sun propose la plateforme Java 2 Micro Edition (J2ME) aux appareils de faible capacité ; de petite taille comme les téléphones portables, PDA…

Elle a deux configurations:

(i) Connected Device Configuration (CDC).

(ii) Connected Limited Device Configuration (CLDC).

Les deux configurations CDC et CLDC nécessitent deux machines virtuelles optimisées respectivement KVM, et CVM.

Pour gérer automatiquement la mémoire du tas de petits appareils, Java utilise le ramasse-miettes (RM) qui est une partie importante de la KVM .il est responsable de la libération des objets java dans la mémoire du tas.

Il est basé sur un ou plusieurs algorithmes tels que l'algorithme comptage de références, l'algorithme marque-balayage et l'algorithme recopie ...

Dans ce papier nous présentons l'état de l'art des algorithmes dans la première section, dans la deuxième section nous décrivons notre algorithme proposé dédié à l'environnement j2me et nous conclurons ce papier par nos perspectives.

II. ETAT DE L' ART:

Dans une application java, un objet commence sa vie dans la Mémoire lorsque le mot clef new est exécuté, sa mémoire nécessaire est allouée sur le TAS puis il est consommé par l'application.

Mais pour la mort d'un objet, le développeur java ne se préoccupe pas de sa destruction comme en c++ qui exige d'avoir un destructeur d'objet après la création de celui-ci.

---

*Corresponding author: Mostafa HANOUNE, *E-mail: mhanoune@gmail.com

En effet, Java utilise pour faciliter la gestion automatie-que de la mémoire le mécanisme RM qui intègre, divers algorithmes selon le cas utilisée.

Il existe plusieurs classes d'algorithmes principales permettant la gestion automatique de la mémoire dynamique [1]: RM par comptage de références, RM par marquage balayage, et RM par recopie.

(1) Le RM par recopie:

L'algorithme de ramasse-miettes par recopie est basé sur un partitionnement du tas en deux zones de tailles égales. Les allocations se font en incrémentant un pointeur dans une première zone (donc en temps constant), jusqu'à rencontrer la fin de celle-ci. L'algorithme se déclenche alors, parcourant les objets de la première zone et les recopiant dans la seconde. Enfin, le rôle des deux zones est interverti.

Un inconvénient à cette recopie est la nécessité d'utiliser des barrières en lecture qui sont assez coûteuses en temps d'exécution.

En effet, le déplacement des objets par le RM pendant l'exécution du programme implique au moins un degré d'indirection lors de la lecture d'une référence.

Hormis cet aspect, cet algorithme présente à peu près les mêmes propriétés qu'un RM par marquage-balayage. Il élimine toutefois la fragmentation de la mémoire.

(2) Le RM par marquage-balayage:

Un RM par marquage-balayage consiste à différencier l'ensemble des objets accessibles du reste du tas. Il s'effectue à partir d'un état initial dans lequel aucun objet n'est marqué.

Ces objets sont alors parcourus et marqués récursivement à partir des références dites racines (pile d'exécution, etc.), ceux qui n'ont pas été visités sont considérés comme inaccessibles, et la mémoire qu'ils occupent est récupérée. Contrairement à l'algorithme par comptage de références, celui-ci ne nécessite pas de barrière en écriture.

Il permet de plus la récupération des structures cycliques. Le surcoût en taille mémoire nécessaire peut également être réduit, puisqu'un seul bit suffit à stocker l'information de marquage d'un objet.

En revanche, le temps au pire cas de l'allocation devient beaucoup plus important, dans la mesure où il nécessite au moins deux parcours de l'ensemble des objets. Ce genre de mécanisme peut alors occasionner des pauses de durées considérables pendant l'exécution d'un programme.

Il n'empêche pas la fragmentation de la mémoire.

G. Chen a bien détaillé cet algorithme dans son article, comme technique utilisable dans la machine KVM [3].

(1) Le RM par comptage de références:

Cette technique associe à chaque objet un compteur, qui représente le nombre de références sur cet objet.il est incrémenté lors de l'apparition d'une nouvelle référence lorsqu'une de ces références disparaît, il est décrémenté.si le compteur devient nul, l'objet n'est plus référencé et la mémoire qu'il occupe peut alors être récupérée.

Cet algorithme est très simple à mettre en œuvre : il suffit d'ajouter un champ à tous les objets, et de détecter l'apparition, la modification, ou la suppression d'une référence (une barrière en écriture est une portion de code s'exécutant à chaque écriture de référence ; le surcoût occasionné par ce genre de mécanisme peut généralement être très réduit [2]).

De plus, il permet une allocation en temps prédictible, puisque la mémoire est récupérée dès la mort d'un objet. En outre, la libération se faisant récursivement elle a un coût d'exécution au pire de cas proportionnel au nombre d'objets du TAS (libération de tout son contenu).

En revanche, il est inopérant sur les données cycliques,

Il n'empêche pas la fragmentation du TAS par conséquent une allocation peut échouer alors que la taille de la mémoire libre est suffisante, mais qu'elle est trop fragmentée.

Des versions améliorées de ces différents mécanismes existent déjà, mais les avancées sur un point se font souvent au détriment d'un autre. Par exemple, Hudson et Moss [4] ont proposé un algorithme par recopie réduisant la taille de la

mémoire supplémentaire nécessaire mais cet algorithme devient très peu efficace s'il doit traiter de grandes structures cycliques.

III. APPROCHE PROPOSÉE:

Afin de surmonter le problème des structures cycliques Et minimiser le nombre des compteurs des objets Pour le RM par comptage de références nous présentons dans ce papier une nouvelle approche qui contourne cette faiblesse et optimise la consommation de mémoire :
Nous découpons la mémoire en petit morceau nommé "page" de 8 à16ko. Notre algorithme repose sur l'idée que les pages beaucoup utilisées dans un passé proche ont beaucoup de chance d'être à nouveau utilisées dans le futur proche. Autrement dit, une page peu utilisée dans un passé proche a peu de chance d'être utilisée dans un futur proche et par conséquent nous allons appliquer le RM par marquage-balayage et défragmen-tation dans les pages beaucoup utilisées dans un passé proche,et le RM par marquage-balayage sans défragme-ntation dans les pages moins utilisées dans le passé proche.

Alors, On associe à chaque page un compteur. Il est initialisé à 0 au début, à chaque utilisation de la page le bit de poids fort du compteur R prend la valeur 1 et dans la deuxième utilisation le compteur est décalé de 1 bit vers la droite. Périodiquement, à chaque top d'horloge, typiquement de l'ordre de 20×10−3 secondes, tous les pages sont parcourus .l'exemple ci-dessous (fig1) montre le résultat après 5 interruptions.

Après 160×10−3 secondes, On parcoure toutes les pages en appliquant Le RM avec marquage-balayage et les pages Plus utilisées 1, 2,4 on défragmente leurs objets, (fig2) et ce n'est pas la peine de défragmenter les pages moins utilisées selon le compteur 3,5.

Objet	R	Compteur
1	1	10110000
2	1	10001000
3	0	00010000
4	0	01011000
5	0	00100000

Fig1: situation après 5interruptions d'horloge

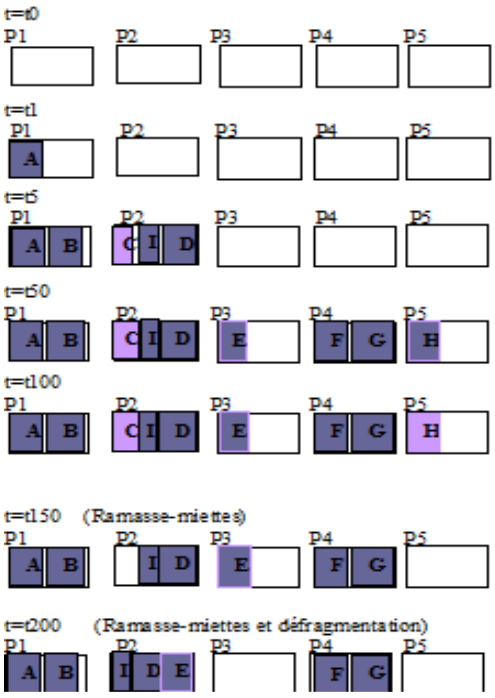
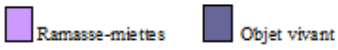


Fig2:Opération du RM Avec M&B et défragmentation



IV. CONCLUSION ET PERSPECTIVE:

Cet algorithme est simple à mettre en œuvre : on affecte un compteur pour chaque page et non pas à un objet co-mme le cas du RM par référence ainsi il surmonte le pr-oblème de structures cycliques reconnu en RM par co-mptage de références, mais le temps d'exécution reste couteux puisqu'il nécessite au moins deux parcours de l'ensemble des objets. Concernant nos futures travaux, on étudiera la possib-ilité d'améliorer notre proposition tout en contournant les inconvénients de cet algorithme dans un premier temps et de généraliser notre algorithme pour le second profile de j2me CDC dans un second temps.

REFERENCES:

[1] Richard Jones and Rafael Lins. Garbage collection: algorithms for automatic dynamic memory management. John Wiley & Sons, Inc., New York, NY, USA, 1996.

[2] Antony L. Hosking, J. Eliot B. Moss, and Darko Stefanovic. A comparative performance evaluation of write barrier implementation. In OOPSLA ’92 : conference proceedings on Object-oriented programming systems, languages, and applications, pages 92–109, New York, NY, USA, 1992. ACM Press.

[3] G. Chen, R. Shetty, M. Kandemir, N. Vijaykrishnan, M. J. Irwin, and M. Wolczko.Tuning garbage collection in an embedded java environment. In HPCA '02: Proceedings of the Eighth International Symposium on High-Performance Computer Architecture (HPCA'02), page 92. IEEE Computer Society, 2002.

[4] Richard L. Hudson and J. Eliot B. Moss. Incremental collection of mature objects. In IWMM ’92 : Proceedings of the International Workshop on Memory Management, pages 388–403, London, UK, 1992. Springer-Verlag.

****

