via DAHMANI selma

PLAN:

via DAHMANI selma

INTRODUCTION

*Le microprocesseur , a su s'imposer au sein du micro_ordinateur comme étant un composant essentiel a son fonctionnement . car ce petit circuit intégré , **exécute** les différentes instructions de programmes installés sur nos machines .*
*Ce composant a connu une évolution très rapide et conséquente **depuis** 1971 ,quand Intel a réussit pour la première fois, à placer tous les transistors qui constituent un processeur sur un seul circuit intégré , **jusqu'à** maintenant ou on parle de multi_coeurs .*
La fréquence d'évolution n'a pas été la même pour les Systèmes d'exploitation : les programmes qui gèrent les différents composants de l'ordinateur .
Ce décalage a causé la réduction des performances de l'ordinateur ,.parce que ces Systèmes d'exploitation y compris Windows , ne gèrent pas totalement les multi-coeurs .

Comment Windows gère t-il ces multi-coeurs ?
quelles sont les problèmes rencontrés lors de sa gestion ? Et comment y remédier ?

I . Architecture des micro_processeurs multi_coeur

Le Multicœur *: le terme multicœur est employé pour décrire un processeur composé d'au moins deux cœurs ou unités de calculs gravées au saint de la même puce, On doit à IBM le premier micro processeur multicoeur à avoir été commercialisé, il s'agit du POWER4 en 2001, puis en 2003, Sun lance l'ULTRASPARK IV composé de deux cœurs, suivis de l'ULTRASPARK V l'année suivante, HP quant à lui, lance le PA-8800 lui aussi composé de deux cœurs. Les premiers exemplaires de processeurs multicoeurs d'INTEL et AMD sont arrivés sur le marché des ordinateurs personnels en 2005, il s'agissait de cœurs homogènes, c'est-à-dire identiques (le cas de cœurs différents utilisées dans des domaines bien précis existent mais sont spécialisées) quant aux premiers processeurs double cœurs basés sur l'architecture X86, ceux ci apparus la même année sur le marché des serveurs,*

Pourquoi les processeurs multicoeurs ?

Selon la loi de Gordon MOOR 1965 : (le nombre de transistors par circuits de même taille va doubler tout les 18 mois). Jusque là, les fabricants afin d'évoluer la capacité des processeurs (nombre de transistors) optaient essentiellement pour une augmentation de la fréquence de ces derniers, cependant une telle miniaturisation atteint ces limites et on ne peut intégrer d'avantage de circuits, et vue leur complexité, l'augmentation de la fréquence devient de plus en plus difficile, ceci implique l'augmentation de la dissipation thermique ainsi que la consommation du processeur, en d'autres termes les gains en performance des processeurs provenaient essentiellement de l'augmentation de leur fréquence ainsi que de la complexité de leur architecture, mais arrivé aux limites de ces techniques, il fut donc proposé d'utiliser plusieurs microprocesseurs fonctionnant parallèlement, et c'est ainsi que fut introduit l'architecture parallèle, afin d'exploiter tout la puissance disponible d'un machine.

1. Structure interne d'un processeur multicoeur :

Principaux composants d'un processeur :

Un processeur est chargé d'effectuer des calculs sur des opérandes, ces calculs son effectués par des unités de calcul, ces unités de calcul sont :

- **ALU** : unité arithmétique et logique ou UAL.
- **La FPU** : unité de calcul de virgule flottante.
- **Les registres** : les quels sont : les registres généraux, les registres de bases et d'index, les registres segments, le registre mot d'état du processeur.
- **Les Bus** : il y a trois types de Bus : le Bus d'adresse, le Bus de données , le Bus de contrôle.

2.Principes de fonctionnement :

Comme il fut cité précédemment, Afin de contourner les limite, les constructeurs se sont tournés vers la fragmentation des puces, il existait déjà des ordinateurs fonctionnant avec plusieurs processeurs distincts, mais l'idée ici est de reproduire ce parallélisme au saint d'une unique puce, en bref, introduire plusieurs unités de calcul dans un même processeur, le principe est simple, plutôt que d'avoir un processeur simple à fréquence élevée, on utilise par exemple deux cœurs de fréquence moitié moindre, on obtient alors un processeur théoriquement de même puissance mais avec une fréquence d'horloge beaucoup plus basse, ainsi qu'une oindre consommation électrique. Le processeur ne rencontre pas le problème d'alimentation et celui de surchauffe de son homologue le monocoeur. En plus de ces avantages purement techniques, il s'avère que ces microprocesseurs sont également bien pus efficaces dans le traitement multitache, sur les systèmes multicoeurs, dans la mesure où chaque cœur dispose de sa propre mémoire cache, le système d'exploitation dispose de plusieurs ressources matérielles pour traiter en parallèle les taches de calculs, si plusieurs applications sont exécutées en même temps sur l'ordinateur, celui-ci permet de repartir le travail entre les cœurs des processeurs plutôt que d'effectuer en alternance sur un seul processeur.
dre pour les fabricants de processeurs.

3.1. Notions de mémoire virtuelle

La mémoire virtuelle fut utilisée dans les premières générations de machines pour remédier au manque chronique d'espace mémoire pour stocker les programmes et les

données. Elle est maintenant plutôt employée pour donner l'impression à une application qu'elle est la seule dans le système et qu'elle peut accéder librement à tout l'espace mémoire. Elle protège ainsi le système d'exploitation des erreurs et des malveillances, tout en évitant que les applications interfèrent les unes avec les autres.

3.2. Les Caches:

Pour réduire les latences d'accès mémoire, une approche classique consiste à intégrer des mémoires de petite taille très proches de l'unité de calcul de manière à réduire les latences d'accès aux données. Plusieurs niveaux de cache sont également embarqués sur le processeur. Deux niveaux de caches (L1 et L2) sont utilisés généralement dans les processeurs mono-cœurs . Le cache L1 est divisé en un cache pour les instructions et un cache pour les données. Il a une latence de quelques cycles, mais une taille très restreinte. À l'inverse le cache L2 à une taille plus importante, mais également une latence d'accès plus grande. Il est souvent unifié (c'est-à-dire qu'il contient à la fois les données et les instructions).Il est possible d'utiliser différentes stratégies pour gérer les caches L1 et L2. La première consiste à rendre le cache L2 inclusif, c'est-à-dire qu'il contient également les données qui sont dans le cache L1. La seconde est de rendre le cache exclusif, c'est-à-dire qu'une zone mémoire n'est jamais conjointement dans les deux caches. Lorsque le processeur cherche à accéder à une donnée, soit la donnée est en L1 (succès du cache L1) soit elle n'y est pas (échec du cache L1). Dans le deuxième cas, le processeur va ensuite interroger le cache L2. Dans le cas où la donnée n'est pas dans ce cache, le processeur va alors aller la chercher en mémoire et la copier dans le cache L1. Cette copie peut nécessiter de remplacer une ligne de cache déjà présente. La politique de remplacement généralement appliquée est de remplacer la donnée utilisée le moins récemment (aussi appelée politique LRU, least recently used). Dans le cas d'un cache exclusif, cela provoque la copie de cette ligne dans le cache L2 (provoquant potentiellement une éviction dans ce cache). Le remplacement d'une ligne de cache est limité par l'associativité du cache. Un cache associatif à N entrées (N-way) laisse N possibilités de remplacement. N représente le nombre de lignes de cache associées à une adresse. L'algorithme de remplacement choisira une ligne parmi ces N possibilités. Le choix de l'associativité est un facteur déterminant de la performance des caches. Plus N est petit, moins le nombre de possibilités de remplacement est important mais plus l'algorithme de remplacement est rapide. À l'inverse, plus N est grand, plus le nombre de possibilités de remplacement est grand et permettra un remplacement pertinent. Cependant, le coût de l'algorithme de remplacement sera plus important. Par conséquent, les caches L1 ont un N généralement petit pour avoir des temps d'accès faibles au détriment de la taille de stockage. À l'inverse, les caches L2 ont un N plus grand pour trouver un bon compromis entre taille et latence. Avec un cache, les accès mémoire fonctionnent de la manière suivante. Lorsque le processeur a besoin d'une donnée, il va d'abord regarder son cache L1 et également demander la traduction de l'adresse virtuelle en adresse physique. Le cache L1 peut être interrogé avec l'adresse virtuelle (et marqué avec l'adresse physique), ce qui permet de paralléliser ces deux opérations. Si l'adresse n'est pas dans le cache L1, le processeur va alors vérifier si la donnée est dans le cache L2 et ensuite va

éventuellement accéder à la mémoire. Enfin, pour améliorer encore la possibilité d'avoir la donnée en cache, le processeur essaye de détecter le modèle d'accès mémoire et, en fonction internes, charge automatiquement.

La parallélisation des instructions:

Pour améliorer l'efficacité des accès à la mémoire, une solution est de pouvoir effectuer d'autres instructions pendant qu'une instruction est en attente de données. Pour cela, le processeur dispose d'un pipeline permettant d'exécuter plusieurs instructions indépendantes en parallèle. L'efficacité de cette technique est d'autant plus grande que le processeur est capable d'exécuter les instructions dans un ordre différent de celui spécifié par le programmeur (tout en garantissant la cohérence des données manipulées) et de faire de la prédiction de branchement. La prédiction consiste à extrapoler le résultat d'un branchement pour charger le pipeline avec la suite des instructions à exécuter. Si la prédiction de branchement s'avère mauvaise, le pipeline doit nécessairement être vidé, ce qui entraîne un coût supplémentaire. L'efficacité de branchement est donc un critère important pour la performance des processeurs.
Ces optimisations permettent d'exécuter sur les processeurs modernes entre 3 et 5 instructions par cycle. Ces processeurs capables d'effectuer plusieurs instructions par cycle sont appelés processeurs super scalaires. Le parallélisme des instructions ainsi engendré est appelé ILP (Instruction Level Parallelism).

*Les microprocesseurs ont connues une évolution dans leur architecture passant par différentes techniques (**[*1]SMT** , **[*2]accès mémoire uniforme, [*3]accès mémoire non uniforme**) ainsi que par différents concepts afin de permettre une meilleur gestion des ressources matériel par le système d'exploitation ce dernier s'adapte aussi à ces nouvelles architectures en proposant des perspectives de systèmes entièrement dédiées et adaptées au multicoeur.*

***[*1]** "Simultaneous multithreading" est l'exécution de plusieurs threads en parallèle sur un processeur*
***[*2]** UMA (Uniform Memory Access) :c'est un technique où tous les coeurs ont la même latence pour accéder à n'importe quelle zone de la mémoire.*

[*3]*NUMA (Non-Uniform Memory Access) : c'est une technique qui consiste à diviser la mémoire pour permettre des accès parallèles à des zones mémoire disjointes. L'accès mémoire est non uniforme, car le cœur accédera soit à sa mémoire locale, soit à une mémoire plus ou moins proche et donc les coûts d'accès ne seront pas identiques en fonction de la zone mémoire accédée*

II. Gestion des processeurs multi_coeur par Windows

II.1 Les éléments de base de gestion d'un système d'exploitation

II.2 Gestion du multi_coeur a travers les différentes versions de windows .

II.3 *Étude de cas* : Gestion du multi_coeur par windows 7

II.1 Les éléments de base de gestion d'un système d'exploitation :

Un des principaux rôles d'un os , consiste en la gestion des ressources . et plus particulièrement la gestion du micro_processeur, considéré comme le cerveau de l'ordinateur .
Avec l'arrivée des processeurs multi_coeur il a fallu repenser les systèmes d'exploitation y compris Windows pour une meilleurs gestion de ces processeurs équipés de deux, quatre voire jusqu'à 100 cœurs.
Alors Comment windows gère - il cette nouvelle génération de processeurs ? et quelles sont les différents éléments de gestion de windows ?

Dans la gestion des processeurs , nous distinguons deux principaux gestionnaires qui appartiennent au système d'exploitation :

A . LE SCHEDULER
B . LE GESTIONNAIRE DE LA MÉMOIRE (CACHE)

II.1.1 *Windows , un Système d'exploitation...*

** Destinée au ordinateurs de type PC ,Windows est une gamme de systèmes d'exploitation produite par Microsoft **

* **Le système d'exploitation** est un ensemble de logiciels qui joue deux principaux rôles qui sont :

- ***l'abstraction du détail matériel par rapport a l'utilisateur***
- ***La gestion des ressources (mémoire , processeur(s) , périphérique)***

Et est composé d'un :

Noyau (**en anglais kernel**) :Contient les fonctions qui gèrent la mémoire, les processus, les fichiers, et les entrées/sorties principales,

Interpréteur de commande (**en anglais shell**) : Permet la communication de l'utilisateur avec le système d'exploitation par l'intermédiaire d'un langage de commandes,

système de fichiers (**en anglais «*file system*»**) : Permet d'enregistrer les fichiers sous forme d'une arborescence .

II.1.2 *L' ordonnanceur (scheduler) :*

DÉFINITION :

Dans les systèmes d'exploitation , " **l'ordonnanceur**" ou (scheduler) en anglais désigne le composant du noyau du système d'exploitation qui choisit **les processus** qui vont être exécutés par le processeur d'un ordinateur .

Le comportement du Scheduler dépend de l'algorithme utilisé . Parmi ces algorithmes, les plus connues sont :

L'algorithme utilisant la méthode **FIFO :** la stratégie employé est que le premier processus arrivé aura la plus haute priorité .

L'algorithme utilisant la méthode **ROUND ROBIN :** Algorithme d'ordonnancement courant dans les systèmes d'exploitation, ce dernier attribue des tranches de temps a chaque processus en proportion égale , sans accorder de priorité au processus.

Et parmi les éléments que le Scheduler doit prendre en compte lors de sa gestion pour aboutir a des performances optimales nous distinguons :

-**Le debit** (throughput): Le nombre de processus qui ont terminés leur exécution pendant une unité de temps .

-**L'équité** (Fairness) : Attribuer le processeur aux différents processus de manière équitable .

II.1.2.1 *Le Scheduling : mono_coeurs vs multi_coeurs :*

Les instructions dans le processeur mono_coeurs sont organisés en une seul file d'attente; les éléments de cette file se dispute l'accès a la **CPU** .
C'est le **Scheduler** qui gère cette file en appliquant un algorithme quelconque et en définissant des priorités au différents processus qui s'exécutent sur la **CPU** **(figure 2.1)**.

Dans le cas des processeurs multi_coeur , le **Scheduler** a la responsabilité de répartir les différents processus sur les différents cœurs disponibles. La première idée qui vient a l'esprit serai d'avoir plusieurs fils d'attente une pour chaque coeur, cette methode de Scheduling n'est pas très concluante car :

Dans les architectures multi_coeur , chaque coeur manage sa propre mémoire cache de niveau 1 [*1] **,** Et en ayant plusieurs files d'attente , Pendant la migration des taches [*2] , on perd le contenu de la mémoire cache du nouveau coeur, ce qui réduit considérablement les performances des différents coeurs .Car le processeurs sera obligé de faire appel a la mémoire centrale et ceci coûte chére en cycle machine . donc , dégrade considérablement les performances globale de l'ordinateur .

Conclusion :

On ne peut pas appliquer la même technique de Scheduling qui est pour les mono_coeur pour les multi_coeur , dans ce qui suit nous allons voir la stratégie adopté pour une meilleur gestion de ces processeurs .

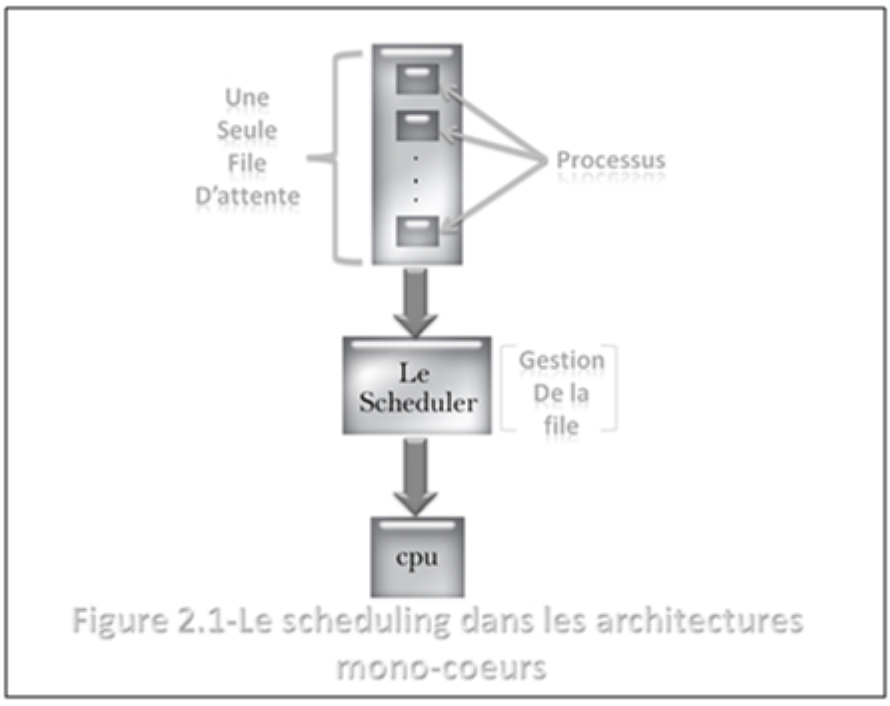


Figure 2.1-Le scheduling dans les architectures mono-coeurs

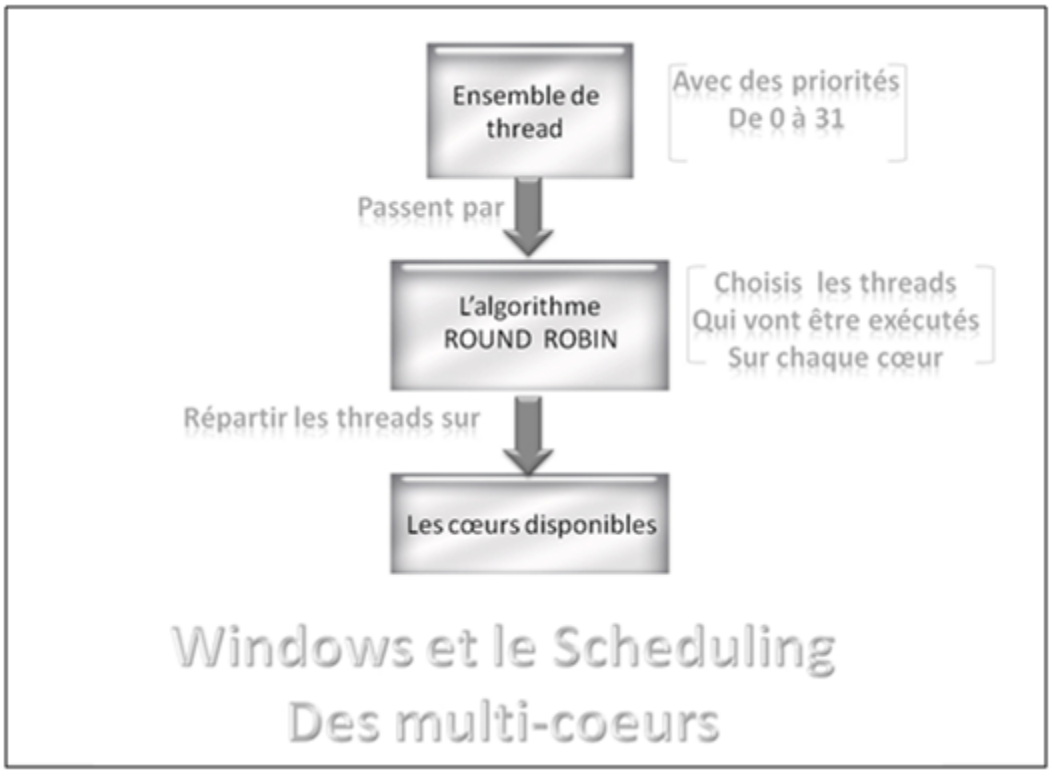
II.1.2.2 *Windows et le scheduling des multi_coeurs*

via DAHMANI selma

Dans **Windows** , le Scheduling concerne les threads , ce dernier est basé sur le principe de priorités entre les threads . ces priorités sont rangés de 0 a 31, des tranches de temps sont alloués aux threads en utilisant un algorithme appelé **ROUND ROBIN .(figure 2.2)**

Ces tranches de temps sont assignés au thread possédants la plus haute priorité , ainsi le thread qui a une basse priorité ne pourra s'exécuter que si celui a la plus grande priorité libère la cpu . outre la priorité basse d'un thread , windows change dynamiquement les priorités des threads pour assurer l'interactivité des systèmes d'exploitation.**[3]**

[*1] *elle contient les instructions assembleurs et les données les plus souvent utilisés*
[*2] *iredirection des processus interrempus vers d'autre files qui appartiennent a d'autres coeurs*



- figure 2.2 -

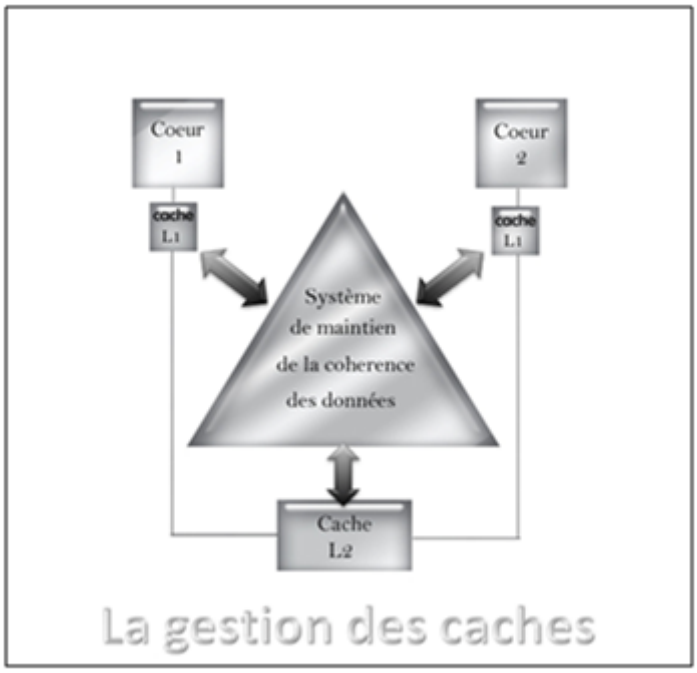
II.1.3 *La gestion des caches dans les multi_coeurs:*

Comme nous l'avons déjà vu auparavant dans l'architecture des Multi-cœurs, ces dernier partagent via un bus les multiples accès simultanés réclamés à la mémoire centrale, qui incluent différents niveaux de mémoires caches .
Chaque cœur se voit doté d'un premier niveau de cache(L1) qui lui est spécifiquement dédié.
À proximité du bus d'accès à la mémoire centrale, on trouve un cache commun (L2) qui fédère l'ensemble des requêtes d'accès à des données que les caches inférieurs n'ont su servi **[1]**..

Un système de maintien de la cohérence des données est mis en place entre ces différents caches : Lorsqu'un cœur accède à une donnée stockée dans le cache associé à un autre cœur, cette donnée est transférée jusqu'au cache associé au cœur qui en a demandé l'accès.**(figure 2.3)**

Le placement des données est l'un des principaux facteurs en ce qui concerne les performances d'exécution d'un programme. Car un cœur accède beaucoup plus rapidement à son propre cache qu'aux autres. Le rapport du coût d'accès à la mémoire distante via la mémoire locale est appelé rapport NUMA.

Plus ce rapport est élevé, plus le coût d'accès à la mémoire des autres nœuds est important. Vu que les accès distants sont beaucoup plus couteux que les accès au cache, et afin d'améliorer les performances d'exécution d'un programme. Les processus qui ont besoin des mêmes données devraient être placées sur un même cœur, comme ça il y aura moins d'accès mémoire distants au profit des accès au cache.



- figure 2.3-

II.2 Gestion des multi_coeur à travers les différentes versions de Windows:

les premiers multicoeurs sont arrivées au marché en 2005 ou Windows XP était le système d'exploitation dominant , après Windows XP , plusieurs versions ont apparues et se sont adaptés à l'utilisation des multicoeurs , là on vas parler des propriétés de chacune de ses version ainsi que des problèmes rencontrés et des solutions mises au point:

Windows XP:

C'est le successeur de Windows2000 et la 1ère version qui a utilisé les multicoeurs , c'est un système d'exploitation multitâche destiné aux machines biprocesseurs , il utilise ce qu'on appelle SMP(multitraitement symétrique) où plusieurs cœurs dans un processeur sont considérés comme des processeurs physiques séparés.

Windows XP ne reconnaît que le dual_core ou il le considère comme un biprocesseur c'est a dire deux processeurs logiques par cœur, comme Core0 et Core1 .

* xp permet une affinité du processeur , où l'utilisateur final ordonne a une application de s'exécuter sur un core spécifique.
Dans La grande majorité des fois, les applications par défaut s'exécute sur le premier coeur, Et dans certains cas, en particulier les jeux, une application qui essaie de s'exécuter sur deux cœurs ne sera pas bien gérée .

XP a rencontré des problèmes de gestion des multicoeurs sous ses 2 versions **home** et **Pro**, chacune de ses versions possède le **même noyau** et les même **fichiers système** mais elles diffèrent quant a la façon dont elles gèrent les multicore ou :

Pour le Home:

Il ne reconnaît au maximum q'un seule processeur physique donc l'ordinateur ne reconnaît que le premier coeur introduit .

Pour le Pro:

il fonctionne comme un système biprocesseur ou il ne reconnaît et gère que les 2 coeurs , mais le problème qui se pose c'est que la version Pro utilise le premier coeur d'une façon complète c a d a 100% mais il n'utilise qu'une partie de le deuxième (10%) ce qui pose des problèmes de performance .

Pour remédier a ces problèmes Windows a fait plusieurs mises à jour disponible mais ce ne fut pas une solution satisfaisante . En plus , avec l'évolution technologique , des multicoeurs qui possèdent plus de deux coeurs ont apparues , ce qui pose des problèmes de compatibilité entre le Windows XP et les multicore récent.
pour cela Windows a développé une nouvelle version Qui s'appelle Windows Vista.

Windows Vista:

C'est le successeur de Windows XP , il est apparue le 30 janvier 2007 affichant plusieurs améliorations importantes, car ce dernier gère jusqu'à 8 coeurs .

ainsi que l'ajout d'un certain nombre de caractéristiques :

-Le support de l'architecture mémoire non uniforme (NUMA) est un système multiprocesseur dans lequel les zones mémoire sont séparées et placées en différents endroits (et sur différents

bus). Vis-à-vis de chaque processeur, les temps d'accès diffèrent donc suivant la zone mémoire accédée.
Le système NUMA a été conçu pour pallier les limites de l'architecture SMP dans laquelle tout l'espace mémoire est accessible par un unique bus engendrant des problèmes **d'accès concurrents** par les différents processeurs. C'est particulièrement nécessaire pour les systèmes ayant de nombreux processeurs.

-L' introduction de la notion de thread pool (c'est l'endroit où un certain nombre de threads sont créés pour effectuer un certain nombre de tâches , qui sont généralement organisés dans une file d'attente) ou Vista pourrait soutenir plusieurs thread pool par processus, quelque chose de XP n'a pas pu le faire.

- «L'anti-convoi caractéristiques" pour eviter que la performance se dégrade quand un grand nombre de threads sont bloqués, en attente de ressources car sinon L'ensemble du système gèlerait pendant quelques secondes et ne réponderai a aucune entrée, puis tout à coup, bam, le système repart, l'exécution de tous ses processus s'effectue .Si dix threads avait besoin d'une ressource et neufs seulement l'avait obtenu ou le dixième un thread n'a pas pu acquérir la ressource, tous les dix threads serait verrouillées. Vista laissait ces threads qui ont besoin d'une ressource l'obtenir, puis envoyait le thread qui ne pouvait pas l'obtenir de la fin de la ligne.

Avec l'introduction de la NUMA et les problèmes rencontrés qui pose le blocage du système, Windows a introduit une nouvelle version plus compatible avec les multicore et qui les gèrent d'une meilleur façon c'est le windows7 :

Definition:

/* *Windows 7 est la 3eme version de la gamme de Windows qui utilise les multicoeurs , elle est disponible depuis le 22 Octobre 2009 en 32 et 64 bit , c'est le successeur de Windows Vista .*

II.3 Gestion des multi_coeur par windows 7 **: ÉTUDE DE CAS**

Windows 7 a été conçu pour fonctionner avec les processeurs multicore d'aujourd'hui. Toutes les versions 32 bits de Windows 7 peuvent prendre en charge jusqu'à 32 cœurs de processeur alors que les versions 64 bits peuvent supporter jusqu'à 256 cœurs de processeur.**[5]**
"***win 7" , possède plusieurs caractéristiques propre a la gestion des multicores qui sont :***

[4] En gérant le concepte NUMA , windows 7 voit des cœurs comme des nœuds fonctionnels[*1], gère la discussion entre les nœuds, et permet le partitionnement ou l'allocation de la mémoire entre les cœurs contrairement aux autres versions qui voit les cores comme des processeurs séparé.

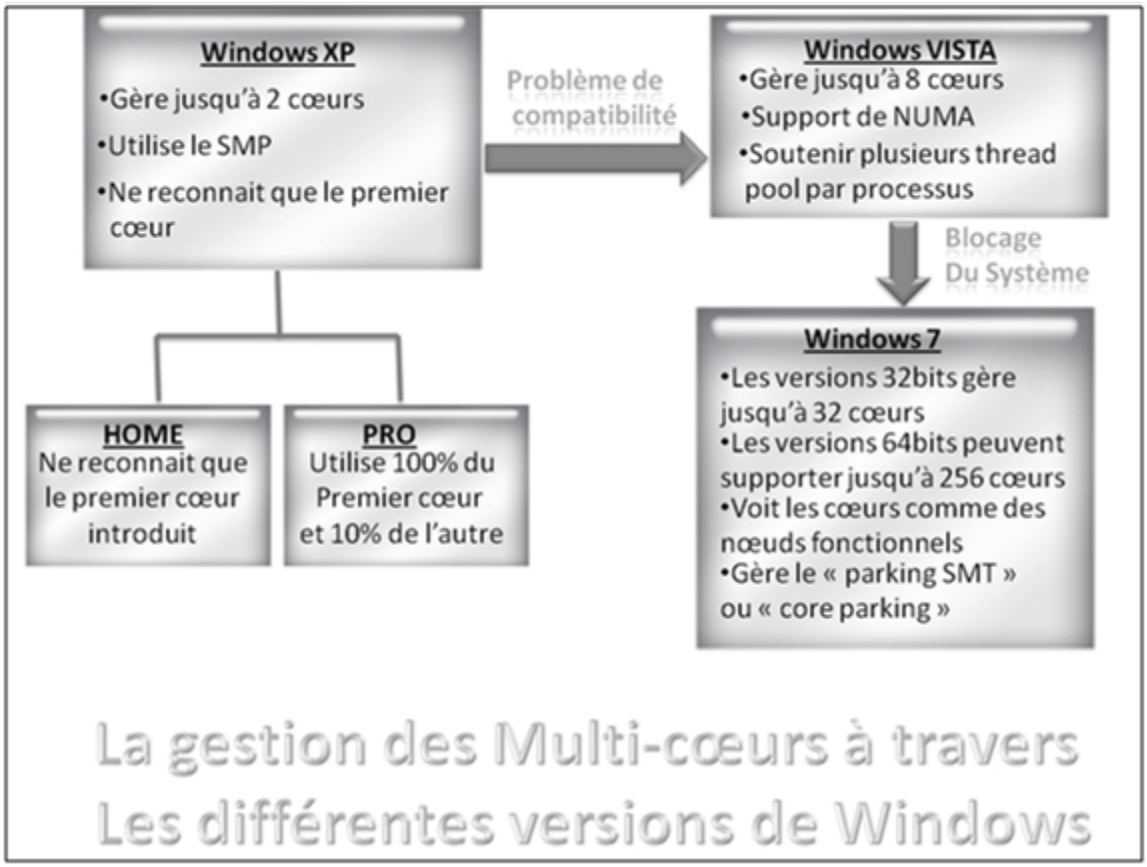
Windows 7 gère aussi le "parking SMT" ou "core parking" Les deux sont utilisés de façon interchangeable. Il s'agit d'une fonction développée en partenariat avec Intel, qui prend en charge la technologie Hyper-Threading (HT) dans les nouveaux Core i5/i7 et permet en plus de repartir les threads entre les coeurs .

Dans des essais de Vista, Microsoft a découvert que beaucoup des blocages du système provenaient de multiples applications qui tentent d'accéder à Graphics Device Interface GDI [*2].

Dans Vista, une application unique pourrait verrouiller l'ensemble du système alors qu'il attendait un autre thread ou application à accéder à l'interface GDI et de redessiner le système Le noyau de Windows 7 dispose d'une allocation de mémoire GDI nouvelle et améliorée. Microsoft a donner au GDI une nouvelle architecture interne en l'associant a un thread. Il ajoute un peu de surcharge, mais il protège le système de blocage lorsque plusieurs applications multiples se battre pour le GDI. Avec les Multi-core sous Windows 7 plusieurs applications peuvent accéder à la GDI en même temps.**[6]**

voici un schéma récapitulatif de la gestion des Multi-coeurs par les 3 versions de Windows (figure 2.4):

via DAHMANI selma



-figure 2.4-

[*1] ***des groupe de coeurs ou chaque groupe a sa propre mémoire partagé***
[*2] *la représentation d'objets graphiques ainsi que pour leur transmission aux périphériques de sorti*

III. Inconvenients du Systéme de gestion de windows

Dans **Windows** et les systèmes d' exploitation les plus courants (Linux , Mac Os) c'est un **noyau unique[*0]** qui prend en charge **tous** les processeurs disponibles , dans le cas d'une architecture multi-processeur ainsi que l'ensemble des coeurs lorsqu'on a une architecture avec des processeurs multi-coeur , aussi cet unique noyau gère l'ensemble de la mémoire associé a ces deux architectures . Et a mesure que le nombre de processeurs ou/et coeurs augmente la gestion de la mémoire partagé et son maintien dans état cohérent devient de plus en plus complexe .Donc , les performances de l'ordinateur n'augmentent pas linéairement avec l'ajout de nouveau processeurs et/ou coeurs de processeurs .

Un autre problème se pose : Windows est un **système centralisé[1]** , ce qui rend les coeurs **dépendants** entre eux (**il gère le tout comme un seul entité**), car si un coeur processeur est suspendu , tout les processus sont inaccessibles jusqu'à ce que ce dernier reprend son activité , et c'est le cas aussi pour les périphériques car si un contrôleur de bus est mis hors tension tous les périphériques sur ce bus deviennent inaccessible .

[*0] : le code qui rassemble les fonctions les plus importantes et critiques du système d'exploitation

IV. Barrelfish la solution miracle !

Barrelfish , est un nouveau projet d'OS avec une stratégie d'implèmentation complètement différente des autres OS de Windows .Il a été mis au point en 2007 , en collaboration avec les chercheurs du laboratoire universitaire ETH de Zurich et le centre de recherche de Microsoft de Cambridge .Ce projet est encore en cours de développement.

Propriétés du Barrelfish:

Noyau :

Appelé CPU driver , ce dernier tourne sur chacun des coeurs (un noyau par coeur) , La communication entre ces coeurs est effectuée de manière **asynchrone** en utilisant le passage de message .

Le passage de message est inspiré du modèle client -serveur dans les systèmes distribués .Dans ce modèle, la communication se fait directement par envoie de message sans l'utilisation de mémoire partagé .

Moniteurs :

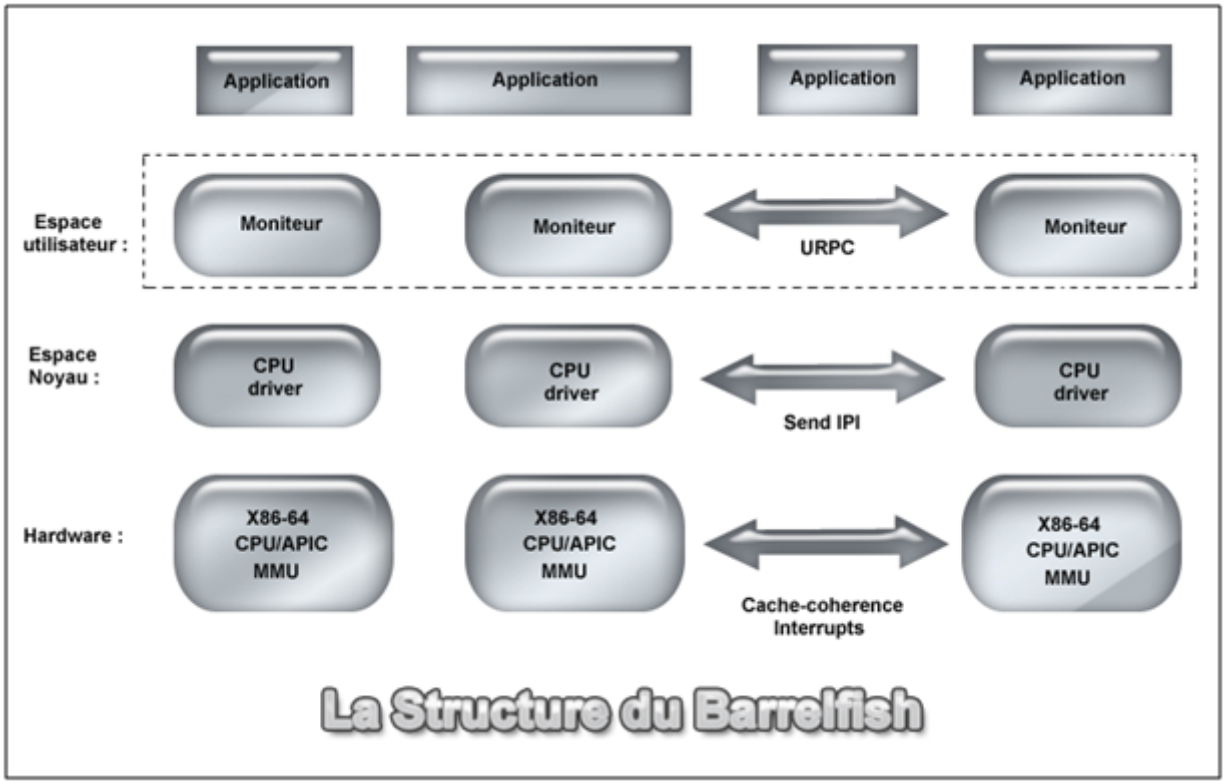
Les moniteurs sont une couche située au-dessus du noyau . Ils sont dupliqués sur chacun des coeurs et sont responsables notamment de l'allocation mémoire ou encore de l'ordonnancement(gestion des processus).

Applications :

Selon le modèle de Barrelfish, une application peut être distribuée sur plusieurs coeurs (éventuellement de types différents).

On peut noter aussi que le Barrelfish est open source **[*0]** .

[*0] *:**La désignation open source** s'applique aux logiciels dont la licence respecte des critères précisément établis par l'Open Source Initiative, c'est-à-dire la possibilité de libre redistribution, d'accès au code source et de travaux dérivés .**Définition web***



-figure 3.1-

Pourquoi Le Barrelfish ?

Le Barrelfish considère le système d'exploitation comme un système distribué :[8]

la propriété principale d'un système distribué est le fait qu'il peut décomposer un projet de calcul globale**[*1]** en plusieurs parties et exécuter les unes indépendamment des autres en plus d'un **middleware** **[*2]** qui coordonne leur activités , et c'est cette **indépendance** qui différencie les *systèmes distribués* des *systèmes centralisés* et les rends meilleurs particulièrement grâce a l'utilisation du **message passing** qui permet d'éliminer les conflits liés a la gestion de la mémoire partagé .

Barrelfish joue la carte du **parallélisme** jusqu'au bout car c'est Un système d'exploitation **MultiKernel** (MULTI- NOYAU) *où* chaque noyau s'occupe de la gestion d'un des coeurs processeur , et ces noyaux ne communiquent pas entre eux , donc on peut envisager Le support de l'hétérogénéité des composants par le Barrelfish , qui peut gérer différents types de processeurs sur une même machine : **par exemple** , il peut gérer un *processeur classique* avec un *processeur* ***GPU[*3].***
Ce système multi_noyeau assure donc une gestion plus **harmonieuse** des différents processus exécutées sur les différents coeurs , contrairement aux systèmes d'exploitation avec **un seul noyau** .**[7]**

Conclusion:

Malgré les maints efforts faits dans l'espoir d'avoir une meilleur gestion des processeurs multi_coeurs , Les différents projets de recherche dans ce domaine restent dans le stade expérimentale, et on besoin d'être perfectionnées .Sans oublier que pour avoir une gestion optimale, il faut aussi développer du code parallèle en plus de l'optimisation des systèmes d'exploitation.

[*1] car toutes les instructions d'1 programme sont transformées en opérations d'addition et …
[*2] ie:*logiciel tiers qui crée un réseau d'échange d'informations entre différentes applications informatiques*
[*3] ie: *processeur graphique : il est présent sur la carte graphique ce qui la rend indépendante du processeur classique*

Résumé

Dans le but d'avoir une parallélisation réel lors de l'exécution des programmes, et une puissance de calcul supérieure avec une fréquence d'horloge moins élevé , les micro-processeurs de type **multi-coeurs** furent introduit et il a fallu repenser les Systèmes d'exploitation qui les gèrent .

parmi ces Systèmes d'exploitation , nous distinguons Windows . chaque Système d'exploitation y compris Windows possède deux principaux éléments de gestion :
le **Scheduler** et **le gestionnaire de la mémoire cache** .

Dans **Windows** , **le Scheduling** concerne les **threads** (processus légers), utilisant l'algorithme appelé **ROUND ROBIN** . Ce dernier est basé sur le principe de priorités entre les threads , ces priorités sont rangés de 0 a 31, et des tranches de temps sont alloués aux différents threads.

Et concernant la gestion de la mémoire cache, un **Système de maintien de de la cohérence des données** a été mis au point Lorsqu'un cœur accède à une donnée stockée dans le cache associé à un autre cœur.

Tout le long de ses versions , **Windows** a effectué divers changement pour assurer une meilleure gestion de ces **multi coeurs** .Passant du support d'un seul coeur(Windows XP) jusqu'au support de plusieurs coeurs et supportant plusieurs architectures , comme la **NUMA , SMP**(Windows7) .

mais ceci na pas été suffisant , a cause des conflits lors de la gestion de la mémoire partagé, car tous les coeurs se disputent l'accès a cette dernière , sans oublier que c'est un noyau unique qui gère les différents coeurs du micro-processeur .

c'est la que fut introduit un os de nouvelle génération appelé **barrelfish** , un système très inspiré des systèmes distribués ,apportant des solutions aux problèmes de gestion de la mémoire partagé en utilisant la technique du **message passing** , donc la communication entre les différents coeur se fait directement par message , sans partage de mémoire .
Barrelfish est aussi un système **multi-noyau** , ou chaque noyau s'occupe de la gestion d'un des coeurs d'où une facilité de gestion avec moins de risque d'erreurs .
Ce système est encore en stade expérimentale et beaucoup d'améliorations sont a prévoir du coté de la compatibilité des applications sur cet OS .

Bibliographie:

[1]http://fr.wikipedia.org/wiki/Optimisation_des_performances_des_architectures_multi-c%C5%93urs
[2] http://www.dailytech.com/Microsoft+Unveils+Barrelfish+a+New+Multicore+OS/article16344.htm
[3] http://www.scribd.com/doc/4838281/Operating-System-Scheduling-on-multicore-architectures
[4] http://itexpertvoice.com/home/multi-core-support-in-windows-7/
[5] http://www.clubic.com/actualite-166638-windows-7-optimisations-multi-thread-coeurs-cores.html
[6] http://www.blog.ma-config.com/
[7] http://www.espace-microsoft.com/fr/actualites/19351-a-recherche-un-os-taille-le-multi-c-ur.html
[8] http://pro.01net.com/editorial/506580/a-la-recherche-dun-os-taille-pour-le-multicoeur/
[9] http://www.clubic.com/actualite-301904-barrelfish-microsoft-os-research-multikernel.html
[10] http://www.pcinpact.com/news/53229-microsoft-barrelfish-multi-kernel.htm
[11] http://research.microsoft.com/en-us/projects/barrelfish/