Conception et implémentation d'applications multimédias en Solaris 2

Philippe Owezarski, Michel Diaz

LAAS - CNRS
7, Avenue du colonel Roche
F-31077, Toulouse cedex 4
e-mail : {owe, diaz}@laas.fr

RÉSUMÉ. L'avènement des applications multimédias est dû aux progrès récents dans de nombreux domaines comme les circuits intégrés VLSI pour traiter les données audio et vidéo, les systèmes de télécommunication, mais aussi les systèmes opératoires avec les nouvelles librairies de threads et fonctionnalités temps-réel. Cet article présente donc la problématique associée à la présentation de données multimédias par des systèmes informatiques, qui doivent être très rapides de part la quantité d'informations à traiter, mais aussi qui doivent respecter les contraintes temporelles des objets multimédias. Pour cela, cet article étudie les différents mécanismes proposés par le système Sun Solaris 2, comme les threads et les différentes classes d'ordonnancement, et propose une méthodologie à suivre pour la conception d'applications multimédias multithreads temps-réel. En particulier, cette méthodologie est appliquée à l'outil de visioconférence synchronisée PNSVS dont les caractéristiques, ainsi qu'une évaluation sont proposées.

ABSTRACT. The explosion of multimedia applications is due to recent progresses in many domains like VLSI design for audio and video data computing, communication systems, and also operating systems with their new threads libraries and real-time functionnalities. This paper thus presents the problematic associated to the presentation of multimedia data by computer systems, that have to be very fast because of the amount of information to compute, but also that have to fulfil multimedia objects temporal constraints. For that, this paper studies the mechanisms provided by the Sun Solaris 2 system, like threads and the different system scheduling classes, and proposes some guidelines for the design of real-time multithreaded multimedia applications. As an example, these guidelines are applied for the design of the PNSVS synchronized videoconference tool which characteristics and evaluation are given in the paper.

MOTS-CLÉS : Synchronisation multimédia, modélisation, threads, ordonnanceur temps-réel, Solaris 2, visioconférence

KEY WORDS : Multimedia Synchronization, modeling, threads, real-time scheduler, Solaris 2, videoconference

1. Introduction

Les progrès effectués ces dernières années dans le cadre des technologies multimédias, des systèmes de communication haut débit et des systèmes opératoires ont conduit à l'avènement des nouvelles applications multimédias distribuées coopératives. En particulier, le mécanisme des threads, généralisé aujourd'hui dans les systèmes opératoires, a beaucoup contribué à cet essort ; en effet, les données multimédias ont la particularité d'avoir des qualités de service, en termes de fiabilité, d'ordonnancement, de quantité d'information, extrèmement variables, et surtout, et c'est une nouveauté pour l'informatique, extrèmement variables en termes de contraintes temporelles, appelées aussi contraintes de synchronisation multimédia ; ces contraintes temporelles peuvent notamment être très difficiles à assurer pour un flux vidéo à 25 images/s pour lequel une image doit être présentée toutes les 40 ms, ce qui engendre une granularité de traitement très fine.

Les données multimédias sont donc des données qui nécessitent beaucoup de ressources (de part la taille des informations manipulées) et dont les contraintes temporelles doivent être respectées au mieux : ces contraintes ne sont pas impératives dans le sens où un certain niveau de gigue ou de dérive (variations temporelles qui seront décrites plus avant) peut être supporté ; ces données se satisfont en fait de supports assurant le respect de contraintes temps réel souples. Toutefois, étant donné le grand nombres de flux en série et/ou en parallèle, et la finesse de la granularité des synchronisations à l'intérieur (intra-flux) et entre les flux (inter-flux) d'un objet multimédia, il est impossible de se satisfaire de l'ancien concept de processus (existant notamment sous UNIX) dont la lourdeur du contexte interdit cette finesse de granularité de commutation, de parallélisme et d'ordonnancement.

De plus, les différents médias qui composent un objet multimédia sont très différents (audio, vidéo, textes, graphiques, etc.) et sont souvent manipulés par des processeurs spécialisés (cartes vidéo, cartes graphiques, cartes audio etc.) ; à cause du niveau de parallélisme qui existent entre ces médias, il faut également pouvoir invoquer de façon parallèle ces différents périphériques multimédias, ce qui n'est pas possible avec les processus lourds ; une fois de plus, les threads semblent apporter une solution adaptée à ce problème.

L'objectif de cet article est donc de proposer une méthodologie d'utilisation des librairies de threads, qui tienne compte de l'environnement système. En effet, les noyaux des systèmes opératoires sont aujourd'hui tous multithreadés pour accroître leurs débits, mais l'association entre les threads du noyau et les threads des librairies utilisateurs n'est pas toujours triviale. En particulier, les applications multimédias sont gourmandes en mécanismes système, et cette association est cruciale à la conception même de ces applications. En particulier, les mécanismes système mis en œuvre au niveau du noyau perturbent considérablement les mécanismes implémentés

dans les librairies de threads, notamment les mécanismes d'ordonnancement, de gestion des priorités et de synchronisation logiques. Aussi, cet article proposera une méthodologie d'utilisation des threads en tenant compte des mécanismes système pour les applications multimédias. Comme l'association faite entre les librairies de threads[1] et le noyau est propre à chaque constructeur, cet article s'intéressera exclusivement aux mécanismes du système Solaris 2. De plus, il prendra comme exemple les mécanismes de synchronisation multimédia d'une application de visioconférence appelée PNSVS : Petri Net Synchronized Videoconference System.

Cet article a donc le plan suivant : tout d'abord, la section 2 présente les contraintes de synchronisation temporelles multimédias en insistant sur l'asynchronisme des systèmes distribués servant de support aux applications ; elle propose aussi de modéliser ces contraintes en utilisant un formalisme basé sur une extension temporelle des réseaux de Petri - le modèle TSPN (Time Streams Petri Net) - dont les pouvoirs de modélisation et d'expression élévés permettent de modéliser de façon simple et complète tous les scénarios de synchronisation multimédia. Ensuite, la section 3 présente les principes de l'architecture multithreads de Solaris 2, en insistant sur l'ordonnancement des threads de la librairie utilisateur, l'ordonnancement des threads noyau, l'association faite par le système opératoire entre ces deux niveaux, et les différentes classes d'ordonnancement proposées par le système. La section 4 propose ensuite une méthodologie de conception des architectures multithreads, qui permette d'utiliser au mieux les ressources et les mécanismes du système pour garantir les contraintes de synchronisation des applications multimédias. Cette méthodologie est appliquée au cas particulier du logiciel de visioconférence PNSVS, dont une évaluation temporelle est proposée (section 5). Enfin, quelques conclusions terminent cet article.

2. Les contraintes de synchronisation multimédias

2.1. Les systèmes distribués asynchrones

L'étude et les travaux décrits dans cet article ont été effectués dans le cadre des systèmes distribués asynchrones. Ce choix s'est imposé pour trois raisons :

• Tout d'abord, les systèmes distribués asynchrones représentent à quelques exceptions près la totalité des systèmes opérationnels existants. Les réseaux locaux, les réseaux d'interconnexion comme Internet et les systèmes opératoires qui s'exécutent sur les stations de travail (Unix, DOS, …) sont asynchrones. De plus, la tendance actuelle (imposée par les constructeurs) ne fait que faire croître l'asynchronisme des systèmes pour gagner en performance, et l'asynchronisme semble donc être l'avenir dans le domaine des machines de bureau : l'asynchronisme

---

[1] les mécanismes des librairies sont communs à tous les constructeurs notamment avec la norme POSIX.

est aujourd'hui le point de passage obligé vers les hauts débits. Aussi, nous avons souhaité faire une étude dans un cadre le plus général possible pour donner des mécanismes de synchronisation qui pourront être généralisés dans un maximum d'applications sur un maximum de sites présents et futurs.

• Ensuite, la solution au problème de la synchronisation multimédia dans un environnement synchrone est un cas particulier de celle en environnement asynchrone (c'est le cas où l'asynchronisme est nul), et nous apportons donc une solution la plus générale possible.

• Enfin, avec l'intégration dans UNIX de classes d'ordonnancement temps réel (comme dans Solaris 2), il est possible de réaliser des applications temps réel, ce qui n'était pas le cas au début de la décennie lorsque les travaux décrits dans [COU94] [JEF94b] et [JEF92] ont débuté, et avaient alors déclaré UNIX inapte pour supporter des applications ayant des contraintes temps réel [VOG95]. Ces travaux préconisaient d'utiliser des systèmes temps réel qui étaient les seuls susceptibles de respecter les contraintes temporelles des applications. Il faut noter ici que la théorie sur les systèmes temps réel est une chose, mais que la réalité en est une autre [KAT94]. Dès que l'on essaie d'implémenter un système opératoire temps réel, quasiment les mêmes problèmes d'asynchronisme qu'avec un système classique (comme Solaris 2) se posent :

  • les entrées / sorties sont asynchrones ;
  • il peut y avoir des problèmes d'inversion de priorités lors de la synchronisation de processus ;
  • le noyau doit être entièrement préemptible ;
  • les tâches système indispensables peuvent perturber l'ordonnancement temporel des applications ;
  • etc. ...

En fait, pour obtenir un comportement temps réel avec un système temps réel, il faut se limiter à des processus de calcul pur qui ne font pas d'entrées / sorties [BAK94]. Cependant, ceci est incompatible avec les applications distribuées multimédias qui font des accès très nombreux aux cartes audio et vidéo et aux cartes réseaux. Les conditions d'utilisation d'un système temps réel sont donc trop restrictives pour des applications distribuées multimédias. En fait, utiliser des systèmes classiques (UNIX ou POSIX) revêt beaucoup d'intérêt, par leur nombre et leur disponibilité, et ne pose pas plus de problèmes que lors de l'utilisation d'un système temps réel [ADE94].

De plus, le choix des systèmes asynchrones s'est imposé car [KAN94] a montré que les systèmes synchrones à grande échelle sont impossibles à mettre en oeuvre. Ainsi, pour l'exemple de la visioconférence, qui nous intéresse tout particulièrement dans cet article, et qui est une application distribuée, et potentiellement longue distance, il faut donc passer par des systèmes asynchrones.

Néanmoins, l'utilisation d'un support asynchrone pour traiter des données multimédias isochrones introduit de nombreux problèmes. Le problème essentiel réside dans la variabilité (aussi appelée gigue) des temps de traitements des différentes

opérations, non bornés supérieurement. Cette variabilité (asynchronisme) apparaît à trois niveaux dans les systèmes distribués multimédias asynchrones :

• les supports et protocoles de communication. Par exemple, les modes d'accès aux principaux réseaux sont non déterministes (dépendant de la charge du réseau par exemple), et aucun délai de transit maximum n'est garanti. Ce phénomène d'asynchronisme est encore plus marqué lorsqu'un réseau public longue distance comme le réseau Internet est utilisé.

• le système opératoire (par exemple Unix) dont la variabilité dans les temps de traitement est due :

- aux mécanismes d'ordonnancement temps partagé du système qui tendent à privilégier les processus interactifs et le débit moyen du système d'exploitation, au détriment du respect de certaines contraintes temps réel des processus. De plus, la notion de processus "lourd" telle qu'elle est utilisée dans Unix induit une surcharge en matière d'ordonnancement incompatible avec le haut degré de parallélisme et les hauts débits associés aux flux multimédias ;
- au caractère non préemptif ou localement préemptif des noyaux qui induit un temps de latence non déterministe pour le traitement des interruptions ;
- au changement des pages mémoires et de nombreuses autres tâches système qui s'exécutent à des moments non déterministes et avec des priorités supérieures aux tâches des utilisateurs, et qui en cela gênent la qualité temporelle de ces dernières ;

• le matériel, et en particulier les cartes multimédias (audio et vidéo). En effet, la nouvelle génération de microprocesseurs intègre des caches d'instructions et de données multi-niveaux qui génèrent des temps d'accès à la mémoire et de commutation de contexte non déterministes. De même, la gestion de la mémoire virtuelle introduit une variabilité des temps d'accès à la mémoire. Cette variabilité est renforcée dans les architectures multiprocesseurs par les mécanismes de réactualisation des caches et les contentions pour l'accès au bus. De plus, les systèmes de gestion d'interruption hiérarchisés induisent des temps de latence non déterministes. Les disques, les processeurs spécialisés de compression / décompression et les périphériques multimédias sont ainsi autant de sources d'asynchronisme.

2.2. Les contraintes temporelles des applications multimédias

Etant données les contraintes et les propriétés d'asynchronisme du système distribué servant de support à la conception des applications multimédias, la problématique associée à la synchronisation multimédia consiste à assurer à la fois les contraintes de synchronisation intra et inter-flux. Garantir les contraintes de synchronisation intra-flux consiste à contrôler la gigue (c-à-d la variabilité dans les temps de présentation) sur tous les objets audio et vidéo pour qu'elle reste en dessous d'une valeur maximale acceptable. Cette valeur de gigue dépend de la nature même du média : par exemple une variation de 10 ms sur la présentation d'une image d'un flux vidéo n'a pas d'incidence sur la qualité de la présentation (cette distorsion est imperceptible pour un utilisateur humain). Garantir la synchronisation inter-flux

consiste à contrôler la dérive qui peut apparaître entre des flux parallèles (dus à l'effet cumulatif de la gigue) et à la garder en dessous d'une valeur maximale (par exemple contrôler la dérive audio / vidéo dans le cas de la synchronisation des lèvres dans une application de visioconférence).

2.3. Le modèle TSPN

Pour réaliser des applications multimédias synchronisées, il faut donc considérer tous les problèmes dus à la variabilité temporelle des temps de traitement (gigue, dérive) et les relier aux propriétés inhérentes à chaque objet multimédia (par exemple, relier la gigue d'un système opératoire à la gigue maximale tolérable par chaque objet multimédia). Pour définir ces propriétés de synchronisation sur les objets multimédias, l'auteur d'une application multimédia doit disposer d'un modèle permettant d'exprimer les contraintes de synchronisation de cette application. Plusieurs études ont déjà été menées dans le domaine et plusieurs modèles ont été proposés. En particulier, certains de ces modèles utilisent des approches formelles basées sur les réseaux de Petri temporisés. Le modèle OCPN (Object Composition Petri Net) [LIT90] ne considère que des temps de traitement nominaux, et ne résoud donc pas le problème de la variabilité des temps de traitement des systèmes asynchrones ; les TPN de Merlin (Time Petri Net) [BER91] ne permettent pas une modélisation facile des contraintes de synchronisation inter-flux, et le modèle ATPN (Arc Time Petri Net) [WAL83] ne propose aucune solution lors de désynchronisations inter-flux. [DIA93] présente ces modèles, et montre qu'ils ne répondent pas à la problématique posée par la synchronisation multimédia en environnement asynchrone.

Ces limitations observées au niveau des modèles existants ont conduit à proposer un nouveau modèle (basé sur les réseaux de Petri[2]), appelé TSPN (Time Streams Petri Net), offrant un bon pouvoir d'expression et un bon pouvoir de modélisation.

Ainsi, les TSPN utilisent des intervalles temporels sur les arcs sortant des places, ce qui permet à la fois de tenir compte du non déterminisme temporel des systèmes distribués asynchrones et de la variabilité des temps de présentation des objets multimédias. Les intervalles temporels dans un TSPN sont des triplets (x^s, n^s, y^s) appelés intervalles de validité temporelle, où x^s, n^s et y^s sont respectivement les temps du traitement de présentation minimal, nominal et maximal. Les durées nominales sont utiles pour calculer la dérive temporelle sur les arcs (par rapport à la durée nominale).

Les dérives temporelles entre flux multimédias peuvent être contrôlées de façon très précise grâce à 9 sémantiques de transition différentes, et grâce à la position des intervalles temporels sur les arcs sortant des places (permettant de traiter chaque flux de façon indépendante). En utilisant ces règles de transition, il est possible de spécifier des mécanismes de synchronisation conduits par le processus le plus en

[2]Pour l'aspect graphique des réseaux de Petri qui permettent de mieux visualiser les caractéristiques de synchronisation d'un document multimédia.

avance (synchronisations de type "ou"), par le processus le plus en retard (synchronisations de type "et") ou par un processus statiquement choisi (synchronisation de type "maître"). Ces sémantiques de synchronisation permettent de définir les instants de synchronisation à partir d'un arc choisi statiquement ou dynamiquement. Pour plus de détails, [SEN96] donne une définition formelle du modèle et de ces différentes sémantiques de tir.

2.4. Exemple d'utilisation du modèle TSPN

Le modèle TSPN constitue un modèle parfaitement adapté à la modélisation des contraintes de synchronisation de flux multimédias, en environnement asynchrone. Grâce à ses pouvoirs de modélisation et d'expression élevés, ce modèle permet de modéliser facilement des scénarios de synchronisation de complexité forte. De plus, à partir de la description en TSPN d'un scénario de synchronisation, il est possible de vérifier sa validité temporelle. Le point essentiel de cet article consiste à étudier comment il est possible à partir d'un TSPN de décrire le comportement d'une couche de synchronisation, et en particulier d'obtenir l'ordonnancement temporel des processus (threads) de présentation multimédia, de façon à ce que pour tous les flux de données, les contraintes de synchronisation intra et inter-flux soient garanties.

Dans la visioconférence, un flux audio et un flux vidéo doivent être synchronisés. Un certain nombre de paramètres de qualité de service peuvent être représentés par un TSPN [OWE96]. Ainsi, la figure 1 modélise une application de visioconférence dont les paramètres de synchronisation sont :

- débit : 10 images par seconde. Le temps de présentation nominal d'un objet est donc de 100ms ;
- gigue acceptable sur les objets audio et vidéo : 10 ms [JEF94a]. Il s'en suit que les intervalles de validité temporelle sont [90, 100, 110] ;
- le son est le média le plus important : les synchronisations inter-flux sont de type "et-maître" sur le son. En effet, le son étant prépondérant par rapport à la vidéo, il se pose comme le flux maître et ses contraintes temporelles doivent être forcément respectées. De plus, l'attribut "et" de cette règle de synchronisation essaie autant que possible d'attendre et de respecter les contraintes sur le flux vidéo pour réduire le nombre de discontinuités ;
- la qualité de la synchronisation doit être excellente, c'est à dire que la dérive inter-flux ne doit pas excéder 100 ms [JEF94a], 100 ms étant la limite en deçà de laquelle le décalage audio vidéo n'est pas perceptible par l'homme. Aussi la période de synchronisation inter-flux correspond à la présentation de 5 images. En effet, la dérive maximale sur 5 objets audio ou vidéo est de 50 ms. La dérive entre les deux flux est donc au maximum de 100 ms.

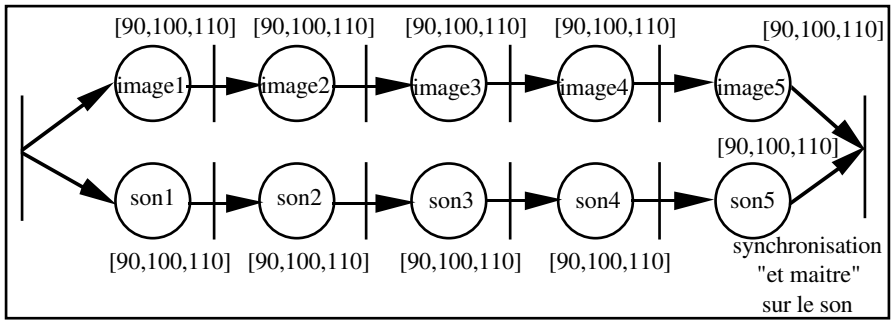

Figure 1. Exemple de TSPN pour la visioconférence à 10 images par seconde

3. Architecture multithreads de Solaris 2

L'objectif des travaux décrits dans cet article est donc de concevoir et développer des mécanismes permettant de respecter les contraintes de synchronisation intra et inter-flux des applications multimédias (contraintes modélisées par un TSPN). Le problème consiste donc à inférer un ordonnancement des threads de présentation des différents flux des applications qui respecte les contraintes temporelles de présentation des différents objets multimédias. Pour cela, cette partie va présenter les principes d'ordonnancement des threads sur le matériel choisi (Sun Solaris 2), les mécanismes de synchronisation logique inter-threads, ainsi que les mécanismes temporels susceptibles de permettre le respect des contraintes de synchronisation des objets multimédias.

3.1. Solaris 2 : une architecture à 2 niveaux

L'architecture d'ordonnancement de solaris 2 est une architecture à 2 niveaux (Figure 2) : un niveau dans l'espace d'adressage du noyau, et un niveau dans l'espace d'adressage de l'utilisateur [POW91] [STE92] [EYK92] [SUN94a] [KLE96].

Au niveau utilisateur, la librairie des threads met en œuvre des mécanismes d'ordonnancement et de synchronisation logique des threads. Pour pouvoir s'exécuter, ils utilisent des processus légers (LWP : LightWeight Processes) qui permettent de faire le lien entre l'espace utilisateur et l'espace noyau. Ces LWP sont en fait des unités centrales virtuelles qui représentent le niveau de concurrence idéal pour une application (un processus Unix).

Au niveau système, les LWP sont attachés à des threads noyau (lien 1 - 1 entre un LWP et un thread noyau). Les threads noyau sont ensuite ordonnancés par l'ordonnanceur système sur les processeurs physiques de la machine.

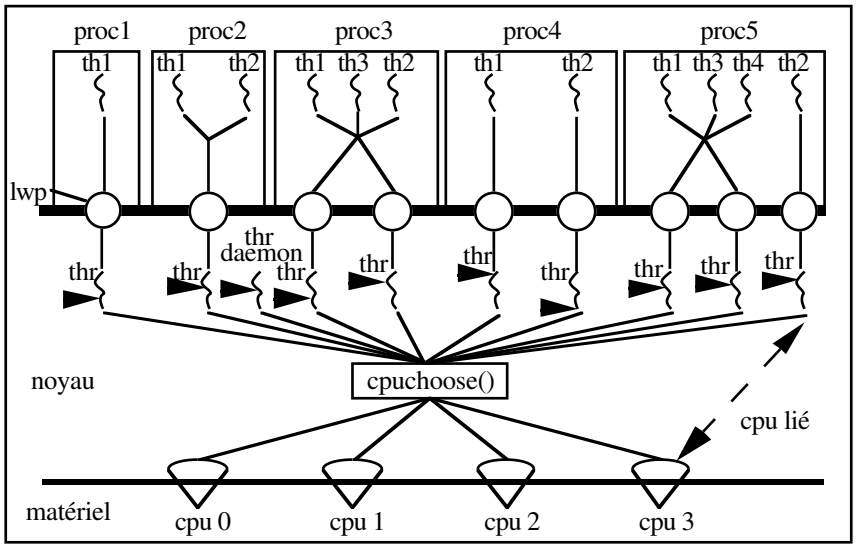

Figure 2. Architecture d'ordonnancement des threads dans Solaris 2

3.1.1. L'ordonnanceur de la librairie des threads

Comme cela a été dit plus haut, la librairie des threads implémente un ordonnanceur des threads. Ce dernier met en œuvre un ordonnancement par niveaux de priorités statiques, c-à-d que le programmeur attribue aux threads des priorités qui ne seront pas modifiées par la librairie ; si les priorités sont modifiées, elles le seront par le programmeur qui demandera ces modifications de manière explicite grâce à des primitives fournies par la librairie des threads et prévues à cet effet. Ainsi, dans un processus Unix, l'ordonnanceur des threads attribue les LWP (processeurs virtuels) aux threads ayant les priorités les plus fortes. Il ne met pas en œuvre de mécanisme de "time-slice", c-à-d qu'un thread peut conserver indéfiniment le LWP qui lui a été attribué, même si d'autres threads ayant le même niveau de priorité l'attendent aussi. En fait, un thread ne peut perdre sa ressource LWP qu'au profit d'un thread de priorité supérieure, ou lorsqu'il est en attente d'une ressource non disponible (mutex, variable condition, etc.) ; la librairie récupère alors le LWP et l'attribue à un autre thread (excepté dans le cas de threads liés, qui sera vu plus loin, et pour lequel les LWP sont parqués).

La librairie des threads implémente également des objets de synchronisation inter-threads : les mutex, variables condition, sémaphores et verrous lecteurs / écrivains. Il faut noter que ces objets de synchronisation fonctionnent aussi sur le principe des priorités statiques de la librairie des threads, c-à-d que le premier thread bloqué qui sera libéré sera celui qui a la plus forte priorité ; si deux threads ont le même niveau de priorité, ce sera le premier arrivé qui sera le premier libéré (gestion FIFO par niveau de priorité).

3.1.2. L'ordonnancement du système

Le noyau de Solaris 2 implémente également un ordonnanceur : l'ordonnanceur traditionnel d'Unix qui dans SunOS 4 ordonnançait les processus Unix. Toutefois, dans Solaris 2 (Version 5 de SunOS), il ordonnance non plus les processus lourds Unix, mais les LWP de ces processus[3]. Les deux classes d'ordonnancement temps-partagé (TS) et système (SYS) traditionnelles des systèmes Unix sont maintenues.

La classe TS, dédiée à l'utilisateur, met en œuvre un ordonnanceur basé sur des priorités dynamiques, c-à-d des priorités qui évoluent en fonction de l'exécution des threads noyau. Les threads noyau élus (c-à-d les LWP qui s'exécutent sur un processeur) sont ceux dont le niveau de priorité est le plus élevé. Toutefois, le niveau de priorité des différents LWP est ajusté dynamiquement : le niveau de priorité d'un LWP élu décroit régulièrement tant qu'il s'exécute. En revanche, un LWP bloqué (sur une variable de synchronisation ou en attente de la ressource CPU) voit sa priorité augmenter. Ainsi, un LWP peut être préempté par un autre LWP dont le niveau de priorité devient supérieur à celui du premier. Ce mécanismes - dont les paramètres sont donnés dans une table d'ordonnancement (TS-DPTBL) - permettent en fait de répartir de façon équitable les ressources CPU aux différents LWP du système.

La classe SYS, dédiée aux tâches système, présente les mêmes caractéristiques d'ordonnancement que la classe TS, mais avec des niveaux de priorités supérieurs ; la classe SYS est prioritaire par rapport à la classe TS afin de permettre aux tâches essentielles su système opératoire de s'exécuter en priorité.

Il est à noter que la complexité des mécanismes d'ordonnancement temps-partagé et système de Solaris 2 rend le comportement temporel des applications complétement imprédictible ; il introduit en fait un très haut degré d'asynchronisme dans le système. De ce fait, il est impossible d'utiliser ces classes d'ordonnancement pour des applications ayant des contraintes de synchronisation temporelles.

3.2. La classe d'ordonnancement temps-réel (RT)

Pour répondre à ce problème, Solaris 2 intègre une classe d'ordonnancement temps-réel qui met en œuvre un ordonnanceur au fonctionnement beaucoup plus simple et beaucoup plus prédictible. La caractéristique principale de cette classe RT reste toutefois son niveau de priorité supérieur à la classe SYS (Figure 3). Pour se préserver du danger que représente le fait d'avoir des utilisateurs dont les applications ont des priorités supérieures à celles du système, Solaris 2 restreint l'accès de la classe RT aux utilisateurs ayant les droits d'exécution du super-utilisateur ; toutefois ceci n'assure en rien le bon fonctionnement de la machine, et la classe RT doit être manipulée avec beaucoup de précautions.

---

[3] En fait, il ordonnance les threads du noyau associés aux LWP, et donc par extension les LWP.

Les règles qui régissent le comportement de l'ordonnanceur temps réel sont basées sur des niveaux de priorités statiques avec "time-slice". En fait, les LWP élus seront ceux qui ont les plus hauts niveaux de priorités, mais dans le cas où ils sont en concurrence avec des LWP de même priorité, ils sont préemptés dès lors que le temps d'exécution qui leur est imparti est écoulé ; le mécanisme d'attribution des processeurs aux LWP par niveau de priorité est basé sur le principe du tourniquet (round-robin). Les paramètres qui régissent le fonctionnement temps réel sont définis dans la table RT-DPTBL.

Toutefois, Solaris 2 avec la classe d'ordonnancement temps-réel et son noyau totalement préemptible (ce dernier aspect étant indispensable pour l'écriture d'applications temporellement contraintes[4]) n'est pas pour autant un système temps-réel. De très nombreux problèmes peuvent apparaître. La partie suivante va d'ailleurs les présenter, et proposer des solutions.

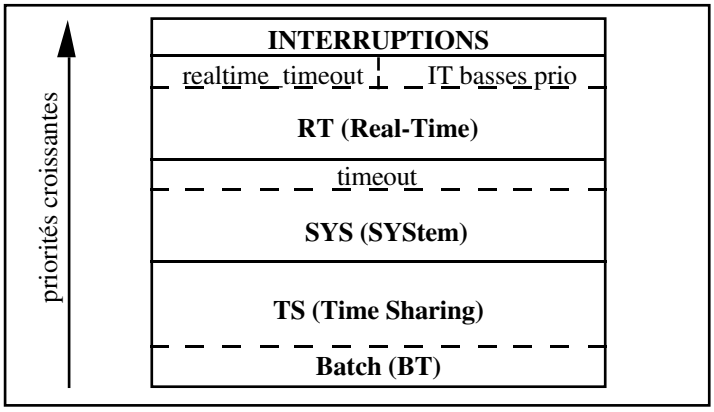

Figure 3. Priorité des classes d'ordonnancement de Solaris 2

4. Utilisation de Solaris 2 pour les applications multimédias

Le système Solaris 2, décrit dans la partie précédente, offre donc, a priori, des mécanismes susceptibles de supporter les contraintes temporelles des applications multimédias : les mécanismes de threads doivent permettre d'obtenir la granularité de synchronisation requise, et la classe d'ordonnancement temps-réel doit apporter les mécanismes permettant de respecter les contraintes temporelles des flux multimédias.

---

[4] La préemptibilité du système Solaris 2 permet d'interrompre à tout moment l'exécution d'une tâche système. Cet aspect est essentiel pour le temps-réel, car si une application ayant des contraintes temporelles strictes doit prendre la main alors qu'une tâche système non préemptible s'exécute, l'application temps-réel devra attendre la fin de la tâche système, et risque de voir ses contraintes temporelles non respectées (la tâche système ayant un temps d'exécution inconnu a priori du fait de l'asynchronisme en classe SYS).

Néanmoins, de part la complexité de l'architecture de Solaris 2 et la variété des services temporels fournis par les différentes classes d'ordonnancement, le contrôle des contraintes de synchronisation de l'application n'est pas trivial ; aussi, cette section va montrer les différents problèmes qui se posent avec Solaris 2, par rapport au respect des contraintes temporelles des applications multimédias, et va proposer des solutions conduisant à une nouvelle architecture applicative utilisant au mieux les services du système opératoire.

4.1. Gestion des deux niveaux d'ordonnancement

Comme présenté plus haut, l'architecture multithreads de Solaris 2 comporte 2 niveaux d'ordonnancement : un ordonnancement dans la librairie des threads et un dans le noyau du système opératoire. Cette architecture à deux niveaux pose un problème de non respect des priorités des threads de la librairie, qui vient de la décorrélation complète existant entre les deux ordonnanceurs ; ainsi, si on considère l'exemple du processus 3 de la figure 2, si le thread 1 a une priorité supérieure au thread 2 qui a lui même une priorité supérieure au thread 3, alors, du fait du niveau de concurrence égal à 2 (2 LWP dans ce processus), l'ordonnanceur va élire les threads 1 et 2 sur les 2 LWP du processus. Cependant, si la machine ne dispose que d'un seul processeur, seul un des LWP va s'exécuter, et rien ne garantit que le thread de priorité la plus forte (thread 1) est associé au LWP de priorité la plus forte par rapport à l'ordonnanceur système. Ainsi, le thread 2 peut très bien s'exécuter avant le thread 1, même si au niveau de la librairie le thread 1 est prioritaire. De plus, même si le thread 1 s'exécute en premier, son LWP va voir sa priorité decroître (excepté s'il s'exécute en classe RT), et à un moment donné, le LWP va être préempté par le LWP associé au thread 2. En fait, avec les deux niveaux d'ordonnancement de Solaris 2, et à cause du niveau de concurrence d'une application (nombre de LWP) qui peut être supérieur au nombre de processeurs, aucun mécanisme système ne permet de respecter l'ordonnancement établi par l'ordonnanceur de la librairie des threads.

Il faudrait donc pouvoir n'utiliser qu'un seul niveau d'ordonnancement. L'idéal serait de pouvoir utiliser le seul ordonnanceur de la librairie des threads car il est plus facile à contrôler - étant donnée sa simplicité - et surtout plus performant [OIK94] car il ne met en jeu aucune ressource système directement.

Cependant, l'ordonnanceur système ne peut pas être désactivé ; aussi, pour ne manipuler qu'un seul niveau d'ordonnancement, il faut annihiler les effets de l'ordonnanceur de la librairie des threads. Pour cela, la solution consiste à créer les threads de l'application liés (à des LWP) [ADE94]. En fait, créer un thread lié consiste en la création d'un thread et d'un LWP associé qui sera exclusivement réservé à l'exécution du thread qui lui est rattaché. Ainsi, grâce à ce lien bijectif, la question de l'ordonnancement des threads sur les LWP ne se pose plus, cet ordonnancement étant complètement défini au moment de la création des threads. Le seul ordonnancement qui intervient alors est celui du système qui répartit les LWP sur les processeurs physiques, comme c'est le cas par exemple pour le processus 4 de la figure 2 où les threads 1 et 2 ne sont pas en concurrence pour l'accession à des

ressources de type LWP. Il faut noter aussi que dans le cas de threads liés, comme on crée un LWP par thread, le niveau de concurrence de l'application est maximal.

D'autre part, le fait d'utiliser des threads liés, et donc d'avoir ce niveau de concurrence maximal pour une application, permet d'éviter les phénomènes de SIGWAITING [SUN94a] [SUN94b]. Ce mécanisme permet d'ajuster le niveau de concurrence de l'application afin d'utiliser au maximum les ressources de la machine. Ainsi, dans le cas du processus 2 de la figure 2, deux threads se partagent un LWP. Si, par exemple, le thread 1 qui s'exécute sur le LWP provoque le blocage du LWP (suite à un appel à une ressource système indisponible), alors la librairie des threads détecte qu'il reste un thread (le 2) qui pourrait s'exécuter, mais qui reste bloqué par manque de ressource LWP ; aussi, la librairie des threads envoie au système un signal SIGWAITING qui engendre la création immédiate d'un nouvel LWP sur lequel le thread 2 pourra s'exécuter, et ainsi éviter de gaspiller des ressources machines. Cependant, ce phénomène pose le problème de l'intérêt même du niveau de concurrence des applications autres que les applications de calcul pur, c-à-d toutes celles qui font des appels système ; en effet, sur le processus 2 de la figure 2, le programmeur de l'application, vue l'architecture des threads qu'il a implantée, souhaitait certainement utiliser la concurrence de 1 pour résoudre les problèmes de parallélisme entre ces deux threads. Or, avec le mécanisme de SIGWAITING, cette concurrence est sans effet, car la librairie des threads va l'amener à son niveau maximum, et pour optimiser l'utilisation des ressources de traitement va ajouter une nouvelle source d'indéterminisme dans le système. La solution d'utiliser des threads liés permet d'éviter ce problème.

Enfin, la classe d'ordonnancement temps-réel ne peut être utilisée que pour des threads liés. En effet, elle ne peut s'appliquer qu'à des LWP et n'existe qu'au niveau de l'ordonnanceur système. Pour avoir des threads au comportement temps-réel, il faut créer une association bijective (un lien) entre un LWP temps-réel et ce thread [SUN94c].

4.2. Gestion des contraintes temps-réel

Pour pouvoir obtenir des présentations multimédias dont les contraintes temporelles seront respectées, il faut absolument soustraire les threads de l'application au comportement asynchrone du système opératoire. Pour cela, la solution offerte par Solaris 2 consiste à utiliser la classe d'ordonnancement RT du noyau. Cependant, utiliser la classe d'ordonnancement RT est très pénalisant pour les tâches système qui ne peuvent plus s'exécuter lorsque nécessaire et qui sont différées lorsqu'un LWP RT s'exécute, ceci risquant de provoquer un dysfonctionnement du système [SUN94a] [SUN94c]. Par exemple, si la machine est chargée par des tâches temps-réel, les opérations système d'entrée / sortie peuvent ne pas se faire. Il faut donc réduire autant que possible la quantité des opérations faites en temps-réel. Ce point est d'autant plus important dans le cadre des travaux présentés dans cet article

que les applications multimédias réalisent un grand nombre d'appels système : tous les appels aux cartes et librairies multimédias, les accès réseau, etc.

De plus, des changements de classe d'ordonnancement se produisent lors des appels système, et le thread appelant (ou le LWP associé) s'exécute en classe système pendant toute la durée de l'appel système. Ainsi, si un thread RT fait un appel système, il perd son attribut RT et subit l'asynchronisme du système. Étant donné le nombre d'appels système réalisés par les applications multimédias, il faut donc décomposer les tâches de présentation synchronisées en deux : une partie temps-partagé (1) qui réalise les opérations de présentation (c-à-d les appels système), qui n'a donc pas besoin de s'exécuter en temps réel car elle perdrait son attribut RT, et cela ne servirait qu'à charger le système en tâches RT, et (2) une partie temps-réel (l'orchestrateur) qui s'occupe du contrôle temporel du thread de présentation.

La solution proposée est donc de décorréler le comportement des threads de présentation et d'orchestration. Les contraintes sur le thread d'orchestration sont toujours garanties, car ce thread s'exécute en classe RT et ne fait appel qu'à des fonctions appartenant à des librairies[5] et ne fait donc pas d'appels système, ce qui lui permet de toujours conserver sa classe d'ordonnancement RT. Par contre, les contraintes sur le thread de présentation ne sont pas garanties à cause des multiples appels système qu'il réalise. Cependant, le comportement asynchrone du thread de présentation ne pénalise pas l'application car le thread d'orchestration respecte parfaitement les contraintes temps-réel de synchronisation et réalise donc un contrôle temporel précis du comportement du thread de présentation auquel il est associé.

4.3. Gestion des objets de synchronisation logique

Pour pouvoir concevoir des applications multimédias en Solaris 2, et en particulier mettre en œuvre des mécanismes de synchronisation temporelle, il faut pouvoir disposer des objets de synchronisation logiques que représentent les mutex et les variables condition. Toutefois, à cause des caractéristiques de l'architecture des threads qui vient d'être décrite pour les applications multimédias, il faut respecter certaines règles dans la manipulation des objets de synchronisation. En particulier, le problème qui se pose est un problème d'inversion de priorités. En effet, les objets de synchronisation sont définis et implémentés dans la librairie des threads ; aussi, les priorités qu'ils prennent en compte sont les priorités au niveau de l'ordonnanceur de

---

[5] L'appel à une fonction d'une librairie ne modifie pas la classe d'ordonnancement d'un LWP. En effet, le code même de ces fonctions ne contient pas d'appel à des primitives système, même si les actions réalisées dans ces fonctions peuvent ensuite provoquer des actions du sytème opératoire. Par exemple, la création d'un thread lié à un LWP engendre forcément la création d'un LWP et d'un thread noyau qui sont des ressources système. Toutefois, la fonction de la librairie ne fera que positionner des variables de la structure de données de l'ordonnanceur du noyau, et c'est lorsque l'ordonnanceur prendra la main que le LWP et le thread noyau seront créés. Ainsi, aucun changement de classe n'intervient lors de l'exécution d'une fonction d'une librairie.

la librairie. Or, dans la méthodologie proposée, seules les priorités système sont utilisées, et il faut faire très attention à ce qu'elles soient respectées au niveau des objets de synchronisation. En effet, si on suppose un thread de présentation en classe TS et son orchestrateur associé en classe RT, si les deux threads ont à se synchroniser sur une variable condition et si le thread de présentation a une priorité au niveau de la librairie des threads supérieure à celle du thread d'orchestration, alors il sera libéré de la variable condition avant le thread RT. Ceci conduit donc à une inversion de priorité par rapport aux priorités système, ce qui risque de mettre en péril les mécanismes de respect des contraintes temporelles. Pour éviter ce problème, il ne suffit pas de s'occuper des priorités système, et notamment, les priorités au niveau de la librairie des threads devront être adaptées aux priorités système.

5. Exemple de conception et d'implémentation pour l'application de visioconférence PNSVS

Les mécanismes d'utilisation de la librairie des threads et de la classe d'ordonnancement temps-réel ayant été proposés, cette section va maintenant en montrer un exemple concrêt d'application pour l'outil de visioconférence synchronisé PNSVS, dont les contraintes de synchronisation sont représentées par un TSPN (figure 1) [OWE95] [OWE96].

5.1. Principes

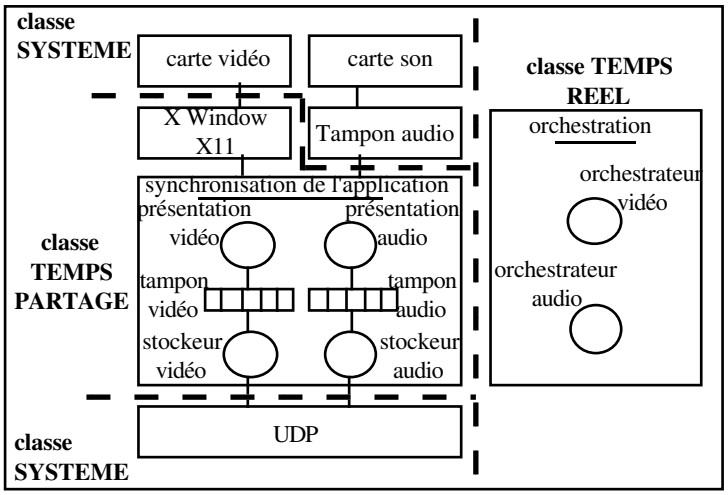

Figure 4. Architecture de synchronisation pour la visioconférence

La figure 4 représente les différents composants du système de visioconférence en associant à chacun d'eux leur classe d'ordonnancement dans le système opératoire. De plus, cette figure montre le découpage de l'application en sous tâches, avec :

- les stockeurs audio et vidéo qui récupèrent les données transitant sur le réseau, et qui les stockent dans les tampons audio et vidéo. Ces tampons servent à stocker temporairement les données arrivant du réseau sans aucune garantie de synchronisation, et à les conserver suffisamment longtemps pour résoudre les problèmes de gigue ;
- les processus de présentation audio et vidéo temps partagé qui réalisent les opérations et les traitements nécessaires aux présentations des sons et des images vidéo ;
- les processus d'orchestration temps-réel qui jouent le scénario de synchronisation.

5.1.1. Algorithme de synchronisation intra-flux

L'algorithme de synchronisation intra-flux (pour les flux audio et vidéo) doit ordonnancer 3 processus, implémentés à l'aide de threads. L'approche proposée pour la synchronisation apporte une solution au problème de la synchronisation multimédia en environnement asynchrone. Cette approche est diamétralement opposée à celles utilisées dans les systèmes temps-réels [JEF94a] [COU94]. En effet, les systèmes temps-réel garantissent une borne supérieure sur les temps de traitement, et tous les algorithmes temps-réels reposent sur cette propriété. Dans les systèmes distribués asynchrones, cette borne supérieure n'existe pas, et la seule borne existante sur laquelle reposent ces travaux est la borne inférieure. En effet, les processus de présentation peuvent s'exécuter pendant un temps minimum, et calculent tous leurs temps d'attente en fonction de ce temps minimum d'exécution ; comme les temps de traitement seront supérieurs ou égaux à cette borne inférieure, les processus de présentation respecteront forcément le temps minimum de présentation. Par contre, pour respecter le temps de présentation maximum, le processus de présentation doit utiliser un timer temps-réel de priorité maximale afin d'éviter les phénomènes d'asynchronisme (figure 3). En utilisant un tel timer, les temps de traitement des processus d'orchestration sont connus et égaux à leurs valeurs physiques minimales, et il est donc impossible de dépasser le temps de présentation maximal des objets multimédias.

5.1.2. Synchronisation inter-flux

L'algorithme de synchronisation inter-flux est basé sur un rendez-vous entre les processus d'orchestration audio et vidéo, ce rendez-vous devant respecter la sémantique de la règle de tir «et-maître». Le principe de ce contrôle temporel est le même que pour la synchronisation intra-flux, si ce n'est que seul le processus d'orchestration audio utilise un timer temps-réel (puisque seules les contraintes sur le flux audio doivent être obligatoirement respectées). Le processus d'orchestration

audio doit attendre la fin du processus de présentation audio et la demande de rendez-vous du processus d'orchestration vidéo (signifiant que la présentation vidéo est achevée) : il peut donc tirer la transition inter-flux pour débuter une nouvelle période. Toutefois, si le timer expire, le processus d'orchestration audio doit arréter les processus de présentation audio et/ou vidéo et doit ordonner au processus d'orchestration vidéo de passer au premier objet vidéo de la période de synchronisation suivante (ce cas correspond à une accélération du flux).

5.2. Implémentation et évaluation

PNSVS a été implémentée sur des stations Sun (Sun Sparc Station 10, 5 ou 2) avec le système Solaris 2.5. Ces machines sont équipées de cartes vidéo Parallax qui permettent de capturer, afficher, compresser et décompresser des images utilisant le format de compression M-JPEG. La carte audio est la carte Sun standard livrée avec ce type de machines. Enfin, les stations sur lesquelles s'exècute PNSVS communiquent en utilisant le protocole transport UDP. Des tests ont été réalisés sur un réseau Ethernet à 10 Mbps et sur un réseau ATM à 155 Mbps.

PNSVS est une application de visioconférence qui peut traiter 20 images/s (320 x 240 pixels x 24 bits de codage des couleurs) bi-directionnel. Le délai de présentation minimum obtenu est d'environ 400 ms et ne peut guère être réduit à cause des temps de latence audio (environ 250 ms).

Il reste cependant important de vérifier que les mécanismes de synchronisation respectent les contraintes de présentation, et en particulier d'évaluer la qualité de la synchronisation intra et inter-flux. Les mesures ont été réalisées dans le cas d'une visioconférence à 10 images/s dont les contraintes de synchronisation sont modélisées par le TSPN de la figure 1.

La figure 5 montre pour les 100 premières images (le reste de l'expérience étant conforme à ces 10 premières secondes), la gigue qui apparaît sur la présentation de chaque image (la partie audio ayant été désactivée). La gigue mesurée est la différence entre le temps de présentation réel et le temps de présentation nominal de chaque image. Cette courbe montre que la gigue maximale de 10 ms n'est jamais dépassée, mais est toujours négative ; en effet, dans ce cas, il n'y a pas de problème réseau (ni gigue ni perte) et donc les données sont disponibles au moment où le processus de présentation en a besoin. De ce fait, le mécanisme d'anticipation, qui rend possible l'arrêt de la présentation d'un objet dès que son temps minimal de présentation a été atteint, fonctionne systématiquement. Les variations qui apparaissent sont liées à l'ordonnanceur temps réel de Solaris 2.

La figure 6 montre la même expérience appliquée au flux audio de PNSVS (le flux vidéo ayant été désactivé). Comme pour la vidéo, les contraintes de présentation intra-flux sont toujours respectées, mais sont systématiquement positives. En effet, dans ce cas là, le tir de la transition intra-flux du TSPN est conditionné par le signal de fin de consommation des données par le port audio. A l'échelle de temps que nous utilisons, cette consommation est toujours identique et égale à 100 ms. Les

variations qui apparaissent sur la gigue sont liées au temps de prise en compte de cette information par le système.

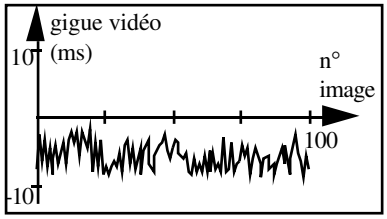

Figure 5. Mesure de la gigue sur les images vidéo de PNSVS

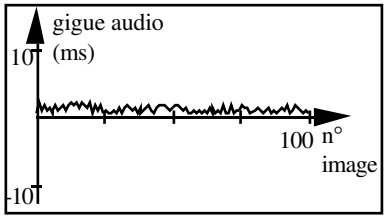

Figure 6. Mesure de la gigue sur les objets audio de PNSVS

La figure 7 montre la courbe représentant la dérive inter-flux en fonction du numéro d'image. Les Figures 5 et 6 ont montré que la gigue vidéo était négative et la gigue audio positive ; aussi, la dérive inter-flux (différence entre les dates de présentation d'un objet audio et de l'image associée) est positive. Cette dérive augmente d'ailleurs régulièrement sur chaque période (ceci est dû à l'effet cumulatif de la gigue), et elle est rattrapée lors de chaque synchronisation inter-flux (tous les 5 objets). Les contraintes de synchronisation inter-flux sont donc parfaitement respectées, puisque la dérive n'excède jamais 50 ms.

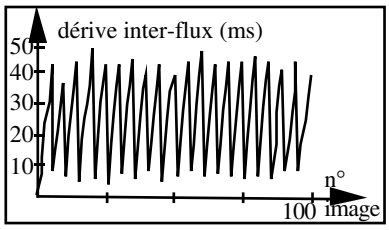

Figure 7. Mesure de la dérive inter-flux dans PNSVS

6. Conclusion

Cet article a proposé une étude des mécanismes multithreads et de l'ordonnancement temps-réel du système Solaris 2, et a proposé une méthodologie montrant comment ces mécanismes peuvent être utilisés pour respecter les contraintes de synchronisation des applications multimédias, et au delà, de toutes les applications temporellement contraintes.

Ainsi, les mécanismes multithreads permettent d'obtenir la finesse de granularité de synchronisation requise par les flux multimédias continus comme l'audio et la vidéo, et les mécanismes de l'ordonnanceur temps-réel du système permettent de

garantir le respect des contraintes temporelles des objets et flux multimédias. Toutefois, l'utilisation de ces deux fonctionnalités du système opératoire n'est pas triviale ; en particulier, l'article a mis l'accent sur l'importance de l'asynchronisme et du non-déterminisme du système opératoire de Solaris 2 qui viennent perturber le comportement des applications de l'utilisateur. Pour résoudre ces problèmes entre l'utilisateur et le comportement du système, des solutions ont été proposées, comme l'utilisation de threads liés, une utilisation correcte des priorités pour les objets de synchronisation afin d'éviter les inversions de priorités, les décompositions de tâches en sous-tâches pour éviter les changement de classe d'ordonnancement, etc.

Cette méthodologie et ces principes ont ensuite été expérimentés sur une application de visioconférence (PNSVS) qui a été décrite, et une évaluation a montré leur efficacité.

La généralisation des classes d'ordonnancement temps réel et des librairies de threads sur tous les systèmes est donc un apport fonctionnel considérable qui permet à la fois d'accroître les performances du système, mais aussi de permettre le respect de contraintes temporelles. Toutefois, malgré les efforts de normalisation, notamment par le consortium POSIX, le problème de la portabilité des mécanismes proposés se pose ; en effet, le consortium POSIX ne normalise que l'interface des librairies, et notamment de la librairie des threads, l'implémentation des mécanismes étant propre à chaque constructeur. Ainsi, la notion de LWP est une notion propre à Sun qui ne se retrouve chez aucun autre constructeur. A cause de l'influence des mécanismes système sur le comportement des applications multithreads de l'utilisateur, il est à peu près impossible d'écrire une application temporellement contrainte (n'utilisant que les seules fonctions des librairies normalisées POSIX) qui ait le même comportement sur l'ensemble des machines des membres du consortium POSIX. La méthodologie proposée dans cette article est donc exclusivement réservée au matériel Sun, et les expériences menées montrent que le comportement de PNSVS est totalement différent sur une Sun, une HP et une SGI, qui sont les trois machines qui ont été testées.

7. Références

[ADE94] B. Adelberg, H. Garcia-Molina, B. Kao, "Emulating Soft Real Time Scheduling Using Traditional Operating System Schedulers", Proceedings of the Real Time Systems Symposium, San Juan, Puerto Rico, December 7-9, 1994

[BAK94] T.P. Baker, F. Mueller, V. Rustagi, 'Experience with a Prototype of the POSIX Minimal Realtime System Profile", IEEE Newsletter on Real Time Systems, 1994

[BER91] B. Berthomieu, M. Diaz, "Modelling and verification of time dependant systems using time Petri nets", IEEE transactions on software engineering, Vol. 17, No. 3, March 1991

[COU94] G. Coulson, G. S. Blair, P. Robin, "Micro-kernel support for continuous media in distributed systems", Computer Networks and ISDN systems, Vol. 26, 1994

[DIA93] M. Diaz, P. Sénac, "Time stream Petri nets, a model for multimedia streams synchronization", proceedings of MultiMedia modelling'93, Singapore, November 1993

[EYK92] J.R. Eykholt, S.R. Kleiman, S. Barton, R. Faulkner, A. Shivalingiah, M. Smith, D. Stein, J. Voll, M. Weeks, D. Williams, "Beyond Multiprocessing... Multithreading the SunOS Kernel, Summer'92 USENIX, San Antonio, TX, june 8-12, 1992

[JEF92] K. Jeffay, D. L. Stone, F. Donelson Smith, "Kernel support for live digital audio and video", Computer communications, Vol. 15, No. 6, July / August 1992

[JEF94a] K. Jeffay, D.L. Stone, F.D. Smith, "Transport and display mechanisms for multimedia conferencing across packet-switched networks", Computer networks and ISDN systems, Vol. 26, 1994

[JEF94b] K. Jeffay, "On latency management in Time-Shared Operating Systems", IEEE Newsletter on real time systems, 1994

[KAN94] Kang G. Shin, Parameswaran Ramanathan, "Real time computing : a new discipline of computer science engineering", Proceedings of the IEEE, Vol. 82, No. 1, January 1994

[KAT94] D.I. Katcher, K.A. Kettler, J.K. Strosnider, "Modeling DSP Operating Systems for Multimedia Applications", Proceedings of the Real Time Systems Symposium, San Juan, Puerto Rico, December 7-9, 1994

[KLE96] S. Kleiman, D. Shah, B. Smaalders, "Programming with threads", 534 pages, Prentice Hall 1996

[LIT90] T.D.C. Little, A. Ghafoor, "Synchronization and storage models for multimedia objects", IEEE journal on selected areas in communications, Vol. 18, No. 3, april 1990

[OIK94] S. Oikawa, H. Tokuda, "User-Level-Real-Time-Threads", IEEE Newsletter on Real Time Systems, 1994

[OWE95] P. Owezarski, M. Diaz, P. Sénac, "Modélisation et implémentation de mécanismes de synchronisation multimédia dans une application de visioconférence", Actes du colloque francophone sur l'ingénierie des protocoles, (CFIP'95), pp 305-319, Editions Hermès, Rennes, France, 10-12 mai 1995

[OWE96] P. Owezarski, M. Diaz, "Models for enforcing multimedia synchronization in visioconference applications", proceedings of the 3rd MultiMedia Modeling conference - Towards the information superhighway (MMM'96), pp 85-100, World Scientific editor, Toulouse, France, November 12-15, 1996

[POW91] M.L. Powell, S.R. Kleiman, S. Barton, D. Shah, D. Stein, M. Weeks, "SunOS Multi-thread Architecture", USENIX, Dallas, TX,Winter'91

[SEN96] P.Sénac, M. Diaz, A. Léger, P. de Saqui-Sannes, "Modeling Logical and Temporal Synchronization in Hypermedia Systems", IEEE journal on selected areas in communications, Vol. 14, No. 1, January 1996

[STE92] D. Stein, D. Shah, "Implementing Lightweight Threads", Summer'92 USENIX, San Antonio, TX, June 8-12, 1992

[SUN94a] SunSoft, Solaris 2.4 Answerbook, "Multithreaded Programing Guide", 1994

[SUN94b] SunSoft, Solaris 2.4 Answerbook, "System Services Guide : Chapter 4 : Process Scheduler", 1994

[SUN94c] SunSoft, Solaris 2.4 Answerbook, "System Services Guide : Chapter 6 : Realtime Programming and Administration", 1994

[VOG95] A. Vogel, B. Kerhervé, G. Von Bochmann, J. Gecsei, "Distributed Multimedia and QoS : A Survey", IEEE Multimedia, Summer 1995

[WAL83] B. Walter, "Timed Petri-nets for modeling and analysing protocols with time", proceedings of PSTV, III, IFIP, 1983