Le projet MidiShare / Open Source

Dominique Fober, Yann Orlarey, Stéphane Letz

Le projet MidiShare / Open Source

Dominique Fober, Stéphane Letz, Yann Orlarey
Grame 9, rue du Garet 69001 LYON
[fober, letz, orlarey]@grame.fr

***Résumé**: MidiShare est un système d'exploitation musical temps-réel, dédié aux applications MIDI. Originellement conçu en 1989, il a été récompensé depuis du prix Apple Trophy (1989), du prix Paris-Cité (1990) et plus récemment du Max d'Or au concours international du logiciel musical de Bourges (1999). Parmi les éléments remarquables du système figurent sa disponibilité sur les principales plates-formes matérielles, son puissant système de communication inter-applications ainsi que ses performances temps réel. Il faut également mentionner la simplicité d'utilisation et de programmation de MidiShare qui ont certainement contribué à en faire un système supporté par un nombre grandissant de projets.*
L'ampleur actuelle du projet ainsi que l'évolution du contexte dans lequel MidiShare est amené à opérer nous ont conduit à en faire un projet "Open Source" et à repenser l'architecture même du noyau, de telle sorte qu'elle permette une meilleure prise en compte des problèmes de portabilité ainsi qu'une plus grande souplesse dans le design de composants tels que les drivers. Ce sont ces évolutions qui sont présentées, notamment à travers une implémentation récente de MidiShare pour GNU/Linux, réalisée sur la base de cette nouvelle architecture.

1 Historique

MidiShare a été conçu il y a une dizaine d'années, en réponse à des problèmes communément rencontrés dans le développement d'applications musicales temps-réel [Orlarey, Lequay, 1989]. Les principales motivations de ce travail sont issues des limitations courantes des systèmes d'exploitation et de leur inadéquation aux besoins des applications musicales : ces systèmes ne permettaient pas le partage des ressources critiques de la machine, ils n'étaient également pas adaptés à la prise en compte du temps et du temps réel, notamment dans la communication inter-applications.
C'est en 1987, sous forme d'une librairie pour l'écriture d'applications musicales, que germent les idées fondatrices de MidiShare. Cette librairie prendra tout d'abord la forme d'un noyau logiciel sous le nom de MAK (pour Midi Applications Kernel) avant de devenir la première version publique pour Macintosh en 1989. Cette version fera rapidement l'objet d'un portage pour les ordinateurs Atari. Quelques années plus tard, ce sont les versions pour Windows 3.1 (1995) puis Windows 95 (1996) qui verront le jour, suivies d'une version thunkée permettant sous Windows d'accéder au noyau 16 bits à partir d'applications 32 bits (1997).
A partir de 1995, MidiShare devient librement accessible aux développeurs d'applications. Le kit de développement est public sur Internet ; une mailing liste MidiShare est mise en place. Profitant de la dynamique du réseau, le nombre de projets supportant MidiShare est alors croissant.
Dans le même temps, MidiShare fera l'objet d'un portage sur un système embarqué et d'un certain nombre de travaux dérivés, prenant en compte l'organisation de la communication inter-applications [Orlarey, 1991], la transmission temps réel sur Ethernet [Fober, 1994], ou encore la problématique de la macro-construction d'applications [Fober & al. 1998]. Le développement de ces travaux ainsi que des projets clients justifieront l'édition du premier CD-ROM MidiShare, disponible depuis début 1999.
Avec le développement des applications multimédia, il eut été naturel de voir les systèmes d'exploitation

évoluer vers une meilleure prise en compte des besoins de ces applications. Au contraire, aujourd'hui tous les systèmes tendent à intégrer des caractéristiques originellement développées sur Unix (telles qu'espaces mémoires protégés, multi-tâches préemptif), qui contribuent grandement à l'efficacité et à la fiabilité de ces systèmes, mais qui compliquent radicalement l'implémentation d'applications temps-réel et la communication en temps réel entre ces applications. Les motivations qui furent à l'origine de MidiShare ainsi que le besoin d'une telle architecture sont donc toujours d'actualité. Le nombre de plate-formes actuellement supportées, leurs évolutions ainsi que des demandes de plus en plus fréquentes de portage sur de nouvelles plate-formes nous ont conduit à revoir l'organisation du projet MidiShare pour en faire un projet "Open Source[1] " et à recentrer l'architecture du noyau autour de ses services essentiels.

2 MidiShare / Open Source

Afin de permettre le développement collaboratif du noyau MidiShare, son code source est librement disponible sous license GNU / LGPL[2] (Library General Public License) depuis 1999. Le source des librairies MidiShare (MidiFile et Player) est également publié sous une license dérivée de la "License Artistic".

L'ensemble de ces sources est géré par un serveur CVS dont les références sont les suivantes :

serveur :	cvs.grame.fr	
authentification :	pserver:password	
user:	anonymous	
passwd:	anonymous	
racines:	MidiShare :	/src/midishare
	Librairies :	/src/lib

Les informations relatives au projet sont accessibles à l'adresse suivante :

http://www.grame.fr/MidiShare/SCPP

Deux mailing listes dédiées au développement du noyau et des librairies sont également accessibles via Majordomo[3] à l'adresse majordomo@grame.fr

3 La nouvelle architecture du noyau MidiShare

L'architecture de MidiShare a été présentée plusieurs fois déjà [Orlarey, Lequay, 1989], [Orlarey, 1990], [Fober & al. 1995]. Pour faciliter son portage, elle a été recentrée sur les services essentiels du noyau et comprend les composants suivants :

- un ordonnanceur, qui a pour responsabilité de délivrer les événements et les tâches à leurs dates d'échéance
- un gestionnaire du temps, qui maintient la date courante du système.
- un gestionnaire de la communication, qui distribue les événements reçus de l'ordonnanceur aux applications clientes et aux drivers.
- un gestionnaire des tâches, qui a pour responsabilité d'activer les tâches délivrées par l'ordonnanceur.
- un gestionnaire de mémoire, spécialement conçu pour fonctionner en temps réel, sous interruption.
- un gestionnaire de pilotes : nouveau gestionnaire qui remplace les pilotes MIDI précédents pour fournir un mécanisme plus général permettant de brancher dynamiquement de nouveaux pilotes dans le noyau.

Pour plus de détails sur cette architecture, vous pouvez vous référer à la documentation développeur [Grame, 1990] ou encore au guide de développement du noyau MidiShare [Grame, 1999].

---

[1] http://www.opensource.org/

[2] http://www.gnu.ai.mit.edu/copyleft/lgpl.html

[3] http://www.greatcircle.com/

4 Portabilité

Plusieurs sections dans l'implémentation du noyau sont dépendantes plate-forme et concernent généralement des services critiques fournis par le système d'exploitation hôte. Parmi ces services figurent :

- la gestion des interruptions
- la gestion de la mémoire
- les basculements de contexte entre le noyau MidiShare et les processus clients
- la synchronisation des processus.

Les systèmes d'exploitation actuels fournissent généralement des fonctions de haut niveau permettant d'effectuer ces tâches. Ces fonctions ne sont cependant pas conçues pour opérer dans un contexte temps réel. En particulier, les problèmes d'efficacité et de précision sont critiques pour un bon fonctionnement du noyau MidiShare. L'implementation de MidiShare dans un contexte donné soulève donc les problèmes détaillés ci-dessous.

4.1 Le processus MidiShare

Le processus MidiShare représente l'entité logicielle à qui appartiennent les ressources nécessaires au bon fontionnement du noyau. Il assure notamment la permanence de ces ressources tout au long de la période d'activité de MidiShare. Il ne fait pas nécessairement référence à la notion commune de processus ; son implémentation est généralement dictée par le système d'exploitation hôte. Les problèmes d'efficacité et de précision doivent ici être particulièrement pris en compte : si par exemple, le processus MidiShare est implémenté comme un processus standard du système d'exploitation hôte et que les coûts de basculement de contexte sont élevés, le noyau résultant sera certainement inefficace puisqu'il devra supporter ces coûts à chaque interruption soit toutes les millisecondes.

4.2 Gestion de la mémoire.

La mémoire est allouée par le processus MidiShare. Celui-ci est libre de choisir le mode d'allocation de la mémoire interne du noyau. Il en va différement de la mémoire allouée aux événements MidiShare : ces événements sont à la base de la communication inter-applications ; deux stratégies différentes permettent d'implémenter le partage des informations transportées par ces événements :

- l'allocation de ces événements dans une zone de mémoire partagée, ce qui permet au noyau MidiShare et à tous ses clients d'accéder à ces événements grâce à de simples échanges de pointeurs. Cette solution à l'avantage d'être très efficace mais constitue un "trou de sécurité" puisqu'elle permet à tout client de corrompre la mémoire du noyau ou d'un autre client.
- la gestion d'un espace mémoire événementiel propre à chaque client et l'implémentation de mécanismes permettant au noyau MidiShare d'accéder à cette mémoire en lecture et en écriture. Cette solution, proche des mécanismes de mémoire protégée, permet d'assurer une meilleure fiabilité du noyau mais est beaucoup plus coûteuse puisqu'elle nécessite la recopie d'espaces mémoire pour assurer les mécanismes de la communication.

4.3 Tâches et alarmes temps réel.

MidiShare fournit les mécanismes de tâches et alarmes temps réel à ses applications clientes. Les tâches temps réel sont stockées dans des événements de type "Process", confiés à l'ordonnanceur et activés sous interruption par le processus MidiShare à leur date d'échéance. Les alarmes sont installées par les applications clientes et activées dans les cas suivants :

- lors de changement global de contexte de MidiShare : il s'agit alors d'une "alarme d'application", qui peut être activée à tout moment, sous interruption ou au niveau utilisateur.
- lors de la réception de nouveaux événements : il s'agit alors d'une "alarme de réception", qui sera toujours activée sous interruption.

Les mécanismes permettant d'activer une alarme ou une tâches clientes sont dépendants plate-forme. Quelques systèmes d'exploitation permettent à un processus d'appeler directement du code client ; la plupart des systèmes interdisent cependant ces appels en séparant les espaces d'adressage de chaque application. Dans ce cas, l'activation d'une tâche consiste généralement à réveiller le thread correspondant

ce qui soulève alors les problèmes de précision, de priorité et de coût des basculements de contexte.

4.4 Synchronisation de processus

Les accès concurrent à des sections critiques du code de MidiShare soulève le problème de la synchronisation des processus. Les systèmes d'exploitation préemptifs fournissent les mécanismes nécessaires à la gestion de sémaphores, ils ne sont cependant pas conçus pour opérer dans un contexte temps réel. En particulier ils posent à la fois des problèmes d'efficacité et d'inversion de priorité. L'implémentation actuelle met en oeuvre des "objets partagés sans verrou" [Anderson & al. 1995] dont le principe consiste non pas à interdire l'accès à des objets en cours de modification, mais à reprendre la procédure de modification tant qu'elle n'a pu être réalisée avec succès. Ce principe permet de conserver la priorité des threads accédant aux objets partagés. De plus, la plupart des micro-processeurs actuels fournissent dans leur jeu d'instruction, des opérateurs permettant par le biais de réservations sur des zones mémoire, d'implémenter les procédures de modification adéquates avec une très grande efficacité.

5 MidiShare pour GNU/Linux

Une implémentation de MidiShare pour GNU/Linux a été réalisée en 1999 sur la base de sa nouvelle architecture. Nous allons détailler les solutions mises en oeuvre pour répondre aux problèmes de portabilité énoncés ci-dessus, ainsi que celles originales qui permettent une meilleure intégration du noyau dans le système d'exploitation hôte.

5.1 Le module MidiShare

L'implémentation de MidiShare constitue un module du système Linux. Les applications clientes de MidiShare accèdent à ses services par le biais d'une librairie partagée qui, dans l'espace utilisateur de ces applications, fournit les méthodes d'accès au module. Une fois chargé en mémoire, le module MidiShare est vu comme une extension du système d'exploitation : en ce sens, le processus MidiShare est pris en charge par le noyau Linux lui-même, ce qui évite tout basculement de contexte entre le système d'exploitation et le noyau MidiShare.

Une des particularités de Linux est de permettre le chargement dynamique de modules, autorisant ainsi l'extension du système à chaud, sans qu'il soit nécessaire de le recompiler. MidiShare bénéficie pleinement de cette souplesse offerte par Linux.

5.2 Gestion de la mémoire

La gestion de la mémoire événementielle de MidiShare se fait dans le respect des espaces mémoire protégés fournis par le système : chaque application gère son propre espace mémoire dans lequel elle alloue les événements MidiShare qui lui sont nécessaires. Ce sont les mécanismes du système d'exploitation qui protègent cet espace mémoire en interdisant son accès à tout processus étranger à l'application. MidiShare fournit dans l'espace utilisateur, sous forme de librairie partagée, les fonctions qui permettent de gérer cette mémoire, y compris les méthodes de synchronisation pour gérer les accès concurrent. MidiShare implémente également dans l'espace du noyau, les méthodes permettant de lire et d'écrire les événements situés dans l'espace d'adressage de ses clients, afin de fournir les mécanismes de communication inter-applications.

5.3 Threads temps réel

Les mécanismes de protection mémoire existant entre les espaces client et noyau interdisent au module kernel MidiShare d'appeler directement du code situé dans l'espace mémoire des clients. Il n'est donc pas possible d'implémenter les tâches ou les alarmes avec des appels directs comme dans des sytèmes à espace mémoire non protégé (Mac OS par exemple). Ces appels sont donc exécutés dans le contexte du client par un thread lui appartenant, ce thread étant géré de manière transparente par le module MidiShare : il est initialisé à l'ouverture de l'application cliente (lors du MidiOpen) et sera détruit à sa fermeture (lors du

MidiClose).
A chaque occurence d'un événement tâche ou alarmes (application ou réception d'événements), le module MidiShare réveille le thread correspondant afin qu'il puisse prendre en charge l'exécution des fonctions correspondantes dans leur contexte d'origine. L'implémentation des mécanismes de suspension et de réveil des threads sont réalisés grâce aux fonctions fournies par Linux : sleep_on et wake_up :

- un appel du thread client à la fonction sleep_on suspend ce thread et provoque l'appel de l'ordonnanceur du système qui élit un autre thread pour exécution.
- la fonction wake_up permet à MidiShare de réveiller un thread suspendu : il change alors d'état et devient à nouveau éligible par le système. Il sera réactivé au prochain appel de l'ordonnanceur.

Ce mécanisme est évidemment moins efficace qu'un appel direct au code des fonctions. Il est en particulier sensible à la stratégie d'ordonnancement des threads utilisé par le noyau Linux. Pour obtenir un temps de latence le plus faible possible (temps entre la notification faite par la noyau et l'éxécution proprement dite du code des alarmes ou des tâches), il faut garantir qu'un thread devenu éligible soit le premier à être activé par l'ordonnanceur du système lors de son prochain appel. Pour cela, le thread est déclaré lors de sa création comme un thread *temps réel*. Il sera exécuté prioritairement à tout autre thread non-temps réel.

5.4 Synchronisation

5.4.1 Principes

Les mécanismes de synchronisation décris en 4.4 sont implémentés avec une pile LIFO et les opérations associées. La pile est réalisée par une liste chainée de cellules qui n'ont pour seule contrainte que d'avoir un pointeur en tout début de structure. Les opérations d'empilement et de dépilement sont lock-free. Elles permettent à plusieurs processus, tournant sur un ou plusieurs processeurs, d'opérer simultanement sur une pile partagée, sans avoir recours à un mécanisme d'exclusion mutuelle. La propriété de lock-free est obtenue en utilisant l'opération CMPXCHG8B, disponible à partir du Pentium, qui permet la comparaison et l'échange atomique de 8 octets. Le coût des transactions est donc réduit au minimum. Cette implémentation est également compatible avec les architectures multi-processeurs.

5.4.2 Structures de données

Deux structures de données sont utilisées : *lifo* et *cell*. Une cellule *cell*, pouvant être empilée et dépilée, est n'importe quelle structure de donnée sous réserve de commencer par un pointeur disponible pour le chainage. La structure de donnée *lifo*, définissant la pile proprement dite comporte, outre un pointeur sur la cellule du sommet de la pile, deux compteurs ic (input counter) et oc (output counter) qui vont permettre de connaitre le nombre d'éléments dans la pile (n=ic-oc), tout en permettant d'identifier l'état de la pile pour CMPXCHG8B.

```
typedef struct cell {
        struct cell* link;      /*+ cellule suivante de la pile +*/
       /* données...*/
} cell;

typedef struct lifo {
        volatile unsigned long  ic;     /*+ input (push) count  +*/
        volatile          cell* top;    /*+ tête de la pile     +*/
        volatile unsigned long  oc;     /*+ output (pop) count  +*/
} lifo;
```

5.4.3 Opérations

Quatre opérations sont définies : lfinit, lfpop, lfpush, lfsize.

lfinit permet d'initialiser une pile. La sémantique de l'opération est la suivante:

```
lfinit() -> [0, ◊, 0] où ◊ représente la liste vide
```

lfpush ajoute de manière atomique, une cellule au sommet de la pile. La sémantique de l'opération est la suivante:

```
lfpush([ic, a:b:...: ◊, oc], n) -> [ic+1, n:a:b:...: ◊, oc]
```

lfpop réalise l'opération inverse. Elle dépile et retourne le premier élément de la pile (si celle ci n'est pas vide). Sa sémantique est définie par les deux règles suivantes:

```
lfpop([ic, a:b:c:...: ◊, oc]) -> a
avec pour effet suivant sur la pile : { [ic, b:c:...: ◊, oc+1] }
            lfpop([ic, ◊, oc]) -> ◊
la pile reste inchangée
```

lfsize essaie de calculer la taille de la pile. Le résultat peut être faux si des opérations sont réalisées entre les lectures des compteurs *oc* et *ic*. La lecture de *oc* avant *ic* garantit de ne jamais sous estimer la taille de la pile.

5.5 Architecture de composants

La nouvelle architecture de MidiShare est une architecture de composants dans le sens où les pilotes de périphériques ne sont plus partie intégrante du noyau mais où celui-ci fournit les mécanismes permettant de brancher dynamiquement ces pilotes dans le noyau.

5.5.1 Les pilotes de périphériques

Un pilote de périphérique est considéré comme étant une application cliente particulière : il bénéficie de toute l'API de MidiShare (alarmes de réception, tâches temps réel, mécanismes d'émission...) pour implémenter ses services et communiquer avec le noyau, avec pour seule différence qu'il s'enregistre comme étant un pilote et non une application. De manière transparente, le noyau prend en charge la distribution des événements MidiShare aux pilotes correspondants et inversement, l'acheminement des événements reçus des pilotes jusqu'aux applications clientes. Un composant particulier, le gestionnaire de pilotes, permet d'associer un numéro de port logique à un pilote sous forme de connexion dynamiquement configurable par l'utilisateur. Le graphe de connexion des ports détermine dans un sens, la stratégie de distribution des événements aux pilotes et dans le sens inverse, l'attribution d'un numéro de port à un événement reçu d'un pilote.

5.5.2 Branchement des composants

MidiShare utilise le système de modules kernel de Linux pour charger dynamiquement ses pilotes de périphériques. Un fichier de configuration, "MidiShare.conf", décrit les pilotes que doit utiliser le noyau ; ceux-ci sont ensuite chargés à la volée lors de la procédure de réveil de MidiShare. Il est de la responsabilité du module de s'enregistrer auprès du noyau comme pilote une fois chargé en mémoire. L'accès aux fonctions de MidiShare se fait grâce à une édition dynamique de liens réalisée par le noyau Linux au chargement du module. MidiShare se contente d'exporter son API ainsi que le pointeur sur son environnement global, de telle sorte que les pilotes chargés sous forme de modules deviennent partie intégrante du code du noyau MidiShare.
Ce mécanisme de module pourrait être utilisé de manière plus générale, pour étendre les capacités du noyau au delà de la seule extension à des pilotes de périphériques.

5.6 Intégration dans le système de fichiers

Sous Linux, les applications MIDI standard accèdent aux entrées/sorties MIDI par l'intermédiaire d'un device utilisable comme un fichier : /dev/midi*xx*. Pour émettre des messages MIDI, ces applications écrivent un flot MIDI standard (avec octet de statut) dans ce fichier et à l'inverse, pour recevoir des messages elles lisent un flot MIDI dans ce fichier.
Pour rendre MidiShare compatible avec cette utilisation du système de fichier, il suffit d'implémenter le comportement décrit ci-dessus dans le device MidiShare, ce qui permet alors d'établir une passerelle entre les application MIDI standard et les applications MidiShare.
L'ouverture du device provoque la création d'un nouveau client MidiShare. L'application MidiShare sera associée au descripteur de fichier passé en paramètre de la fonction *open*. Le client interne est connecté par défaut aux entrées sorties physiques de manière à émuler le comportement du device MIDI.
Les octets MIDI écrits sur le device sont convertis en événements MidiShare datés à la date courante au moment de leur écriture. Lorsque l'application interne reçoit des événements, ceux ci seront linéarisés sous

la forme d'un flot MIDI standard pour être lus par l'application utilisant le device. La conversion sera faite lors de l'appel de la fonction *read*.

5.7 Limites actuelles de l'implémentation Linux

- La tâche du temps du noyau est insérée dans la liste des tâches à éxecuter par l'ordonnanceur du système à chaque interruption. Elle est éxécutée avec une fréquence de 100 Hz sur Linux/PC (au lieu de 1000Hz pour les autre implémentations). Ainsi à chaque interruption, l'ordonnanceur de MidiShare est appelé 10 fois.
- Le temps de latence du thread temps réel associé à chaque client est fonction de la charge du système.

Références

[Anderson & al. 1995] James H. Anderson, Srikanth Ramamurthy, Kevin Jeffay. *Real-Time Computing with Lock-Free Shared Objects*. Proceedings of the 16th IEEE Real-Time Systems Symposium, Pisa, Italy, 1995, pp.28-37

[Fober, 1994] Fober D. *Real time, musical data flow on Ethernet and MidiShare software architecture.* Proceedings of the ICMC, 1994, ICMA, San Francisco, pp. 447-450

[Fober & al. 1995] Fober D., Letz S., Orlarey Y. - *MidiShare, un système d'exploitation musical pour la communication et la collaboration* - Actes des Journées d'Informatique Musicale JIM95, Paris, pp.91-100

[Fober & al. 1998] Fober D., Carron T., Letz S., Orlarey Y. - *Cristallisation d'applications musicales par collaboration* - Actes des Journées d'Informatique Musicale JIM98, Marseille, pp.A2-1, A2-7

[Fober & al. 1999] Fober D., Letz S., Orlarey Y. - *MidiShare joins the Open Sources Softwares* - Proceedings of the ICMC, 1999, ICMA, San Francisco., pp.311-313

[Grame 1990] *MidiShare Developer Documentation*. Grame 1990, Lyon

[Grame 1999] *MidiShare Kernel Development Guide*. Grame 1999, Lyon

[Orlarey, Lequay, 1989] Orlarey Y., Lequay H. *MidiShare : a Real Time multi-tasks software module for Midi applications.* Proceedings of the ICMC, 1989, ICMA, San Francisco, pp.234-237

[Orlarey, 1990] Orlarey Y. *An Efficient Scheduling Algorithm for Real-Time Musical Systems*. Proceedings of the ICMC, 1990, ICMA, San Francisco, pp.194-198

[Orlarey, 1991] Orlarey Y. *Hierarchical Real Time Inter application Communications*. Proceedings of the ICMC, 1991, ICMA, San Francisco, pp.408-415