M2

x86-64/Sparc [Dimitri]

L'objectif de ce stage est double.

L'étudiant commencera par le portage de l'exo-noyau Hexo sur le processeur x86 64 bits de type CISC. Hexo et MutekH s'exécute déjà nativement sur l'architecture intel x86 32 bits classique des PCs.

Ce stage implique la prise en main du code du projet MutekH et plus particulièrement de l'hexo noyau Hexo et son portage x86. Le stagiaire devra développer les routines en assembleurs et en C des différents services : démarrage du système, handler d'interruption, changement de contexte, mémoire virtuelle, etc. Ce code pourra s'inspirer fortement du portage 32 bits existant. L'exécution du système sur une machine muni d'un processeur 64 bits permettra de valider cette partie du stage.

Une fois ce premier portage réalisé, l'étudiant aura une bonne connaissance du fonctionnement et de l'arhitecture de l'exo-noyau Hexo. Il pourra alors réaliser le portage sur processeur Sparc V8. Pour cette partie, la validation sera réalisée sur la plate forme de simulation SoCLib. D'autres processeurs de type RISC similaires sont déjà supportés par MutekH et seront une source d'inspiration pour l'implémentation, mais le processeur Sparc présente une réelle particularité qui ouvrent des pistes intéressantes quant à l'optimisation du changement de contexte. Le stagiaire pourra alors donner une certaine dimension recherche à ce stage.

NFS [Nicolas et Joel]

L'objectif de ce stage est le développement d'un driver de système de fichier Network File System (NFS) pour la nouvelle couche de système de fichiers virtuel de MutekH. Ce driver permettra à MutekH d'accéder aux fichiers au travers du réseau et viendra compléter la collection de systèmes de fichiers déjà supportés : Vfat, Iso9660, RamFS...

Le stagiaire devra prendre en main le projet MutekH et plus particulièrement, la pile réseau (libnetwork) et la gestion du système de fichiers (libvfs) pour développer le driver. Un serveur de fichiers NFS sera mis en place, l'accès aux fichiers depuis une application tournant sur MutekH permettra de valider le stage.

Composant réseau SoCLib et le driver MutekH associé ?

L'objectif de ce stage est double. Tout d'abord il s'agit de développer un modèle de périphérique réseau pour la plate forme de prototypage virtuelle SoCLib. Ceci permettra à une simulation tournant sous SocLib? d'accéder au réseau de la machine hôte. Le modèle de simulation utilisera les mécanisme de DMA classiques qui sont déjà employés par d'autres composants. Il exploitera également la technologie TUN/TAP qui permet de créer facilement une interface réseau virtuelle sur la machine hôte. Ceci permettra d'immerger la simulation dans un vrai réseau, les paquets réseau entrant par l'interface réseau du noyau Linux seront reçus par le périphérique SoCLib et inversement.

Le second objectif est le développement du driver associé pour piloter l'interface réseau depuis MutekH qui contient déjà la pile réseau nécessaire et des pilotes pour des périphériques réseaux existants, MutekH étant déjà capable d'accéder au réseau lorsqu'il s'exécute sur certaines plateformes comme les PCs.

Le stage sera validé en exécutant une application sous MutekH capable d'échanger des paquets avec le réseau de la machine hôte.

Libe2fs [Nicolas et Joel]

L'objectif de ce stage est le développement d'un driver pour la famille des systèmes de fichiers Ext2, Ext3 et Ext4. Ces systèmes de fichiers sont ceux de base généralement employés par les systèmes d'exploitation GNU/linux.

Le support de ces systèmes s'appuiera sur la bibliothèque existante Ext2fs qui contient l'algorithmique complexe nécessaire. Ce driver permettra à MutekH d'accéder, par exemple, aux fichiers d'une partition GNU/Linux et viendra compléter la collection de systèmes de fichiers déjà supportés : Vfat, Iso9660, RamFS...

Le stagiaire devra prendre en main le projet MutekH et notamment la gestion du système de fichiers (libvfs) ainsi que la couche d'accès aux périphériques de blocs pour développer le driver. Il devra également prendre en main et adapter la bibliothèque Ext2fs qui est utilisée pour l'accès aux systèmes de fichiers de cette famille sous GNU/Linux.

Coprocesseur Crypto pour SoCLib et le driver MutekH associé [Alexandre et Geoffrey]

L'objectif de ce stage est le développement d'un modèle de composant SoCLib de type crypto-processeur et de son driver de périphérique pour MutekH.

Les crypto-processeurs sont des périphériques qui sont capables de chiffrer et de déchiffrer des données en mémoire par des opérations DMA. Les algorithmes de chiffrement sont un élément clef de la sécurité et s'implémentent particulièrement bien en matériel. Ces crypto-processeurs permettent de soulager les processeurs généralistes pour les opérations cryptographiques et sont de ce fait très employés dans les SoC.

Le stagiaire devra définir le jeu de registre et les automates du nouveau périphérique avant de réaliser son implémentation. Il s'agit d'étudier les mécanismes (DMA, IRQ, ...) à implémenter pour permettre la collaboration entre le logiciel et le matériel qui s'échangeront les données à traiter. Parallèlement, le développement du driver de périphérique pour MutekH et son intégration à la libcrypto qui gère déjà des algorithmes logiciels permettra de tester le modèle de composant.

Le stage sera validé en appliquant une série d'opérations cryptographiques à un jeu de données et en comparant les résultats obtenus entre les implémentations matérielles et logicielles d'un même algorithme. Ce stage n'implique pas nécessairement le développement ou la compréhension profonde des différents algorithmes cryptographiques dont les implémentations existent déjà sous forme de code libre.

Driver de MMU pour ARM9 et boot sur GP32 [Dimitri]

Une MMU(Memory Management Unit) est un composant passerelle entre le processeur et la mémoire centrale. L'espace d'adressage de cette dernière n'est pas forcément identique à celui utilisé par le processeur. L'accès au données nécessite alors une traduction d'adresse, effectué par la MMU. Celle-ci convertit l'adresse demandée par le processeur (adresse dite virtuelle) en une adresse réellement disponible en mémoire (adresse physique).

Suivant les processeurs et les architectures, les MMU diffèrent, les fonctionnalités proposées ne sont pas les mêmes. Par exemple, alors que sur x86 les pages mémoire ont une taille de 4 Kio ou 4 Mio, elles ont une taille de 1 Kio, 4 Kio, 64 Kio ou 1 Mio sur ARM.

La gestion de la mémoire virtuelle est décomposable en deux parties dans Hexo/MutekH:

- un driver, spécifique à chaque MMU, qui effectue des opération bas niveau pour lire/modifier les tables de pages dans Hexo. L'ensemble des drivers de MMU partage une API générique.
- des gestionnaires de page physiques et virtuelles dans MutekH, qui font directement appel au driver.

L'objectif de ce stage est d'ajouter à l'exo-noyau Hexo le support de la mémoire virtuelle sur un processeur ARM doté d'une MMU de référence. Pour cela, l'étudiant devra implémenter le driver pour la MMU du processeur ARM. Ce driver devra évidement respecter l'API citée précédemment. Hexo s'exécute déjà nativement sur une architecture munie d'un processeur ARM et d'une MMU générique du projet SoCLib. Il sera possible de s'inspirer de ce driver pour effectuer le travail demandé.

La validation se fera soit sur une carte de développement ?SAM9-L9260 d'Olimex, soit sur la console de jeux portable ?GamePark32. Elles utilisent toutes les deux un processeur du type ARM 9 avec MMU. La validation consistera à exécuter une application sur MutekH/Hexo utilisant la mémoire virtuelle.