Fast Traces

FXT

Manuel d'utilisation

Vincent Danjean

29 octobre 2014

Table des matières

1 Présentation **2**

2 Installation **3**
- 2.1 Récupérer les logiciels . . . 3
- 2.2 FKT . . . 3
- 2.3 FUT et les utilitaires . . . 6

3 Prise de traces **8**
- 3.1 FKT . . . 8
  - 3.1.1 Instrumentation du noyau . . . 8
  - 3.1.2 Récolte des traces noyaux avec `fkt_record` . . . 8
- 3.2 FUT et PM^2 . . . 9
  - 3.2.1 Instrumentation des programmes . . . 9
  - 3.2.2 Récoltes des traces avec PM^2 . . . 10

4 Exploitation des traces **11**
- 4.1 Analyse des traces récoltées . . . 11
- 4.2 Visualisation graphique . . . 11

5 Perspectives **15**

Références **16**

Chapitre 1

Présentation

Aujourd'hui, il est très difficile d'observer et de comprendre finement les performances des applications reposant sur des supports d'exécution multithreadés, en particulier lorsque la plateforme de threads utilisée est complexe (ordonnancement multi-niveaux). F$_{\text{X}}^{\text{T}}$ est un environnement permettant d'observer précisément le comportement des applications multithreadées donnant ainsi accès à des informations telles que le nombre de cycles consommés par une fonction donnée ou l'ordonnancement exact des threads utilisés.

F$_{\text{X}}^{\text{T}}$ est composé de deux parties indépendantes (FKT et FUT) pour la récolte de traces ainsi que d'outils pour l'analyse et l'exploitation des traces récoltés.

FKT *Fast Kernel Trace* permet de récolter des événements relatifs au noyau Linux grâce à un ensemble de macros insérées dans le code source du noyau. Cette récolte est réalisée de manière très efficace et non intrusive.

FUT *Fast User Trace*, très similaire à FKT, permet de faire la même chose (récolte d'événements) en mode utilisateur.

On peut les utiliser indépendemment l'un de l'autre.

Les outils Un ensemble d'outils permettent de manipuler et d'exploiter les traces générées par FKT et FUT. On peut ainsi les parcourir mais aussi les fusionner pour générer une unique *supertrace* décrivant pour chaque événement sur quel processeur, dans quel thread noyau et dans quel thread utilisateur il s'est exécuté. Il est alors possible de générer une trace au format du logiciel Pajé afin de visualiser ces traces de manière graphique et interactive.

Pour de plus amples informations sur le fonctionnement interne de FKT et FUT, le lecteur se réfèrera à [1].

Chapitre 2

Installation

2.1 Récupérer les logiciels

FXT est disponible sur le site http://savannah.nongnu.org/projects/fkt. Pour bénéficier pleinement de FXT avec des programmes utilisant des processus légers utilisateurs, vous aurez également besoin d'une version récente de MARCEL disponible sur le site http://runtime.bordeaux.inria.fr/marcel/.

L'installation de FKT nécessite de patcher le noyau. L'installation de FUT et de la libfxt ne nécessite pas de patcher le noyau.

2.2 FKT

Les étapes nécessaires pour installer FKT dans un noyau LINUX sont décrites ici. On supposera par la suite que les fichiers du projets FXT ont été rapatriées dans un répertoire appelé `FxT/` ci-dessous.

Note : à ce jour, seule l'architecture `x86` contient les routines assembleurs de FXT. Un travail de portage serait nécessaire pour utiliser FXT sur d'autres architectures.

Note (bis) : Pour une machine dédiée où un programme utilisateur n'utilisera pas plus de threads noyaux que de processeurs (par exemple, un programme utilisant la bibliothèque MARCEL), FXT n'est pas réellement nécessaire pour observer le comportement des threads.

1. Récupérer les sources d'un noyau Linux

Le noyau original 2.6.10 est parfait. Une version patchée devrait également convenir, au moins pour la partie générique. Les patches de recueil d'événements pourront ne pas s'appliquer directement ; il faudra alors les corriger manuellement.

La partie générique devrait s'appliquer sur n'importe quel noyau 2.6 récent, mais les patches de recueil d'événements doivent parfois être modifiés. Dans ce cas, vérifiez toujours qu'une nouvelle version de FXT n'est pas disponible avec les patches adaptés à votre noyau.

Le répertoire contenant les sources du noyau LINUX sera appelé `linux/` par la suite.

2. Installer la partie générique de FKT

Placez-vous dans le répertoire contenant FxT.

```
cd FxT/
```

Modifiez `Makefile.config` pour indiquer où se situe votre répertoire contenant les sources du noyau Linux, ainsi que la version des patches à essayer (indiquez la version exacte de votre noyau si elle est disponible, la plus proche sinon). Les versions de patches disponibles sont les numéros de version des répertoires `linux-patches-...` dans le répertoire de FxT.

Installez les fichiers génériques (indépendant de la version du noyau) à l'aide de la commande suivante :

```
make install-fkt
```

Ceci copiera les fichiers FKT, *i.e.* les fichiers `arch/i386/kernel/fkt_header.S`, `include/linux/fkt.h`, `kernel/fkt.c` et `kernel/fkt-mod.c`. Tous ces fichiers sont indépendants de la version du noyau.

Il faut ensuite modifier (patcher) quelques fichiers du noyau. Cela s'effectue avec la commande :

```
make install-sys
```

Si le noyau utilisé correspond exactement à celui déclaré dans `Makefile.config` (même numéro de version, aucun patch supplémentaire), alors cette étape doit se dérouler sans aucun problème. Dans ce cas contraire, il sera peut-être nécessaire corriger l'application des patches en les appliquant à la main. La figure 2.1 peut aider à ce travail dans ce cas.

3. Insertion de prises de traces supplémentaires (facultatif)

Si l'on désire instrumenter plus en détail le noyau, il existe déjà des ensembles d'instrumentation prêtes à l'emploi. *Cette étape est complètement facultative.* Il est inutile d'appliquer ces patches si les événements enregistrés ne sont pas intéressants pour vous.

Ces ensembles d'instrumentation ne sont disponibles que pour le noyau 2.4.21 à l'heure actuel. Si vous utilisez un autre noyau, vous aurez probablement à gérer des conflits manuellement lors de l'application de ces patches.

Pour appliquer ces ensembles d'instrumentation, il faut donc positionner `KERN_VERSION` à 2.4.21 dans `Makefile.config`, puis utiliser une ou plusieurs des commandes suivantes :

```
make install-fs
```

ajoute des prises de traces pour les opérations du système de fichier : `lseek`, `read`, `write`, `ext2_getblk`, `ll_rw_blk`

```
make install-net
```

ajoute des prises de traces dans la pile TCP/IP. Différentes couches peuvent être sélectionnées grâce au masque d'enregistrement (*c.f.* **??** à la page **??**)

```
make install-netdriver-3c59x or acenic or tulip
```

`arch/i386/config.in`
`arch/i386/Kconfig`
`arch/i386/defconfig`
`arch/i386/kernel/Makefile`
`kernel/Makefile`
: ajoute FKT au système de compilation. En cas de problème, l'application du patch est très simple à réaliser manuellement.

`include/linux/pagemap.h`
`mm/filemap.c`
`fs/read_write.c`
: ajoute quelques fonctions utilitaires. Pour les anciens noyaux 2.4, `sendfile` est également modifiée pour appeler FKT comme il se doit. Facile à appliquer manuellement.

`include/linux/mm.h`
`include/linux/page-flags.h`
`mm/page_alloc.c`
: définit un nouveau drapeau pour les pages FKT afin d'avoir un compteur de référence non null si elle sont `Reserved`.

`include/linux/fs.h`
`include/linux/buffer_head.h`
`fs/buffer.c`
: ajoute quelques fonctions utilitaires, facile à appliquer manuellement. Quelques prises de traces sont également insérées, mais elles ne sont utiles que pour le déboggage de l'écriture des traces FKT.

`kernel/sched.c`
`fs/exec.c`
`include/linux/sched.h`
`kernel/pid.c`
: ajoute des prises de traces nécessaires aux outils d'analyse pour suivre les fonctions `fork()`, `exec()`, `switch_to()` et `wait4()`.

`arch/i386/kernel/entry.S`
: ajoute des prises de traces pour tous les appels systèmes ainsi que quelques appels systèmes pour enregistrer des événements sur demande des programmes utilisateurs. Ces modifications doivent être fusionnées en faisant bien attention en cas de problèmes.

`include/asm-i386/unistd.h`
: ajoute les appels systèmes cité au-dessus

`include/asm-i386/hw_irq.h`
: ajoute des prises de traces pour les IRQs. À traiter avec attention en cas de problèmes.

`include/linux/interrupt.h`
: ajoute une `softirq` pour FKT. Facile à appliquer manuellement.

`kernel/timer.c`
: ajoute des prises de traces pour nanosleep. Peut être ignoré en cas de problèmes.

FIGURE 2.1 – Résumé des modifications apportées aux sources

ajoute des prises de traces dans le pilote de la carte réseau correspondante

```
make install-scsigen
```

ajoute des prises de traces dans le pilote SCSI generique et le pilote CD SCSI

4. Compilation du noyau

Une fois adapté, le noyau LINUX doit être compilé en validant le support pour FKT.

Allez dans le répertoire `linux` et configurez votre noyau :

```
cd linux/
make config (ou make oldconfig ou make xconfig)
```

Vous devez valider le support (éventuellement en module) pour FKT (`CONFIG_FKT`). `CONFIG_FKT_TIME_ONLY` permet de réduire la taille des traces en n'enregistrant pas les paramètres, mais ce mode est à réserver à ceux qui veulent profiler le noyau (et non pas profiler des applications en espace utilisateur).

Il faut ensuite compiler son noyau et l'installer. Par exemple, avec :

```
make dep && make clean && make bzImage
make install
```

5. Pour les utilisateurs

Il est également nécessaire de créer le fichier périphérique `/dev/fkt`, par exemple avec la commande :

```
mknod /dev/fkt b 60 0
```

Pour les personnes ayant choisit de compiler FKT en module, elles pourront ajouter à leur fichier `/etc/modules.conf` la ligne :

```
alias block-major-60 fkt-mod
```

de sorte que le module puisse être chargé automatiquement.

Pour restreindre l'utilisation de FxT aux utilisateurs d'un certain groupe `group`, il est possible de modifier les permissions du fichier `/dev/fkt` :

```
chgrp somegroup /dev/fkt
chmod 440 /dev/fkt
```

2.3 FUT et les utilitaires

FUT et les utilitaires de FxT ont besoin de quelques bibliothèques externes. Vous devrez probablement installer, si ce n'est pas déjà fait, le package contenant la bibliothèque `libbfd` (probablement le package `binutils-dev`).

Il suffit ensuite d'utiliser le classique

```
./configure
make
make install
```

Cela construit tous les fichiers nécessaires à l'utilisation de FUT ainsi que les utilitaires qui permettront de récolter et analyser les traces.

Pour plus de facilité d'utilisation, vous pourrez installer les programmes dans un répertoire de votre `PATH` ainsi que les bibliothèques (fichiers `libfxt.*`) dans un répertoire lu par le linker (`/usr/local/lib`, `/usr/lib`, ...) ou présent dans votre variable d'environnement `LD_LIBRARY_PATH`.

Chapitre 3

Prise de traces

FxT permet d'obtenir des traces précises du déroulement de l'exécution de flots d'exécution. Ce chapitre explique brièvement comment placer des mesures dans les application et comment obtenir les traces.

3.1 FKT

FKT permet d'observer le comportement du noyau. Il y a deux objectifs à cela : observer le fonctionnement du noyau lui-même et obtenir des traces utiles pour l'observation de programmes en espace utilisateur.

3.1.1 Instrumentation du noyau

L'installation des patch de FKT (*c.f.* partie 2.2) dans les sources du noyau Linux insère les prises de traces nécessaires à l'observation des programmes en espace utilisateur. À savoir, des traces sont récoltées pour

- chaque changement de contexte ;
- chaque création/destruction d'un flot d'exécution (processus ou thread noyau) ;
- chaque changement d'espace virtuel (`execve`) pour enregistrer le nom des nouveaux processus ;
- chaque transition noyau/utilisateur (appels systèmes, interruption matérielle, etc.).

Il est possible d'instrumenter plus finement le noyau. Quelques patches sont fournis avec la distribution (*c.f.* partie 2.2), mais pour des besoins particuliers il est parfaitement possible d'instrumenter soit-même le noyau en utilisant les mêmes techniques que celles développées ci-après pour les programmes en espace utilisateur.

3.1.2 Récolte des traces noyaux avec `fkt_record`

Pour enregistrer les traces récoltées par le noyau, l'utilitaire `fkt_record` est fourni. Il s'utilise de la manière suivante :

```
fkt_record [options..] [--] [program p1 p2 p3 ...]
```

Si le programme n'est pas donnée, `fkt_record` démarre un fils qui se contentera de consommer du CPU.

Les options possibles sont :

`-f output_file`
: sauve la trace dans le fichier `output_file`. Si cette option n'est pas fournie, `fkt_record` utilise le contenu de la variable d'environnement `TRACE_FILE` et en cas d'absence il crée le fichier *trace_file*

`-k mask`
: masque à utiliser pour la récolte d'événements. Par défaut, `fkt_record` utilise la valeur 1 qui enregistre uniquement [1] les transitions noyau/utilisateur (appels systèmes, irq, etc.)

`-S System.map`
: si ce fichier est indiqué, son contenu est fusionné avec celui de `/proc/kallsyms` ou `/proc/ksyms` en vérifiant qu'il n'y a pas d'incohérence. En effet, les symboles du noyau sont enregistrés au début de la trace pour permettre l'analyse de la trace par la suite

`-s size`
: taille des blocks. Par défaut, la taille des blocks du système de fichier est utilisée. Option à réserver aux experts.

`-p pow`
: alloue 2^{pow} page mémoire pour contenir les traces avant écriture sur disque. Par défaut, 7 pages sont allouées. Il peut être nécessaire d'augmenter cette quantité si on enregistre de grosses quantités de traces en un très bref instant.

`-n`
: aucun programme n'est lancé. La prise de trace s'arrêtera lorsque l'utilisateur interrompra (avec `^ C` par exemple) le programme `fkt_record`.

3.2 FUT et PM^2

FxT fournit des outils pour permettre l'instrumentation des programmes utilisateurs, mais ils ne sont pas aisés à manipuler directement. C'est pourquoi il est conseiller d'utiliser une bibliothèque de processus léger intégrant déjà le support pour FxT. PM^2 et sa bibliothèque de thread Marcel sont parfaitement intégrés avec FxT.

3.2.1 Instrumentation des programmes

Les traces utilisateurs proviennent de l'instrumentation des programmes. PM^2 propose plusieurs manière de le faire.

Instrumentation manuelle : un ensemble de macro est disponible pour prendre des traces automatiquement. Il est possible d'enregistrer tout types d'événements. Les macros nécessitent un code unique (qui identifie le type d'événement) et acceptent un nombre variable de paramètres, actuellement de 0 à 5, mais ce nombre peut être augmenté jusqu'à 254

1. Les changements de contexte sont *toujours* enregistrés par FKT, quelque soit le masque actif.

sans difficultés. L'ensemble de ces macros est disponible dans le fichier `fut.h`.

Instrumentation automatique : en utilisant l'option `-finstrument-function` de `gcc`, les fonctions des fichiers objets compilés ainsi auront automatiquement une trace générée à l'entrée et une à la sortie.

En outre, deux fonctions sont à utiliser dans le programme principal :

```
void profile_activate(int FUT_ENABLE, int fut_mask, int fkt_mask);
void profile_stop();
```

La première permet de démarrer et choisir les masques d'événements pour FKT et FUT, la seconde permet de stopper les traces et de les sauver sur le disque. PM^2 sauve les traces dans les fichiers `/tmp/prof_file_user|kernel_`*`login`*`[_`*`node`*`]`.

Note : le nœud n'est sauvé dans le fichier que si PM^2 est utilisé en réseau (*i.e.* avec MADELEINE).

Note (bis) : PM^2 active lui-même FKT si nécessaire. Il n'y a alors pas besoin d'utiliser le programme `fkt_record`.

3.2.2 Récoltes des traces avec PM^2

Pour récolter les traces avec un programme PM^2, il suffit de le lancer avec une *flavor* contenant le support FxT. Pour cela, au moment du choix des options de la *flavor*, il faut :

- inclure le module `profile`
- prendre l'option `FKT` du module `profile` si l'on veut utiliser FKT en plus de FUT
- choisir l'option `gcc-instrument` pour chacun des modules PM^2 que l'on veut instrumenter automatiquement avec `gcc`
- choisir l'option `profiling` pour chacun des modules PM^2 afin d'utiliser les instrumentations manuelles déjà présentes

L'exécution du programme ainsi compilé générera automatiquement les traces (à condition de bien démarrer et arrêter la prise de traces avec `profile_activate` et `profile_stop`.

Chapitre 4

Exploitation des traces

Une fois les traces recueillies, il est nécessaire de les analyser. FUT fonctionnant sur deux architectures différentes (64bits et 32bits), il est nécessaire de prendre en compte cet aspect pour utiliser les programmes : le format des traces binaires récoltées ne sera pas compatible.

4.1 Analyse des traces récoltées

Dans un premier temps, l'utilitaire `fxt_print` permet de convertir une trace binaire récoltées en une trace au format texte.

```
fxt_print [-f trace_file] [-d] [-o]
```

Les options disponibles sont les suivantes :

`-f trace_file`
: lit la trace dans le fichier `trace_file`. Si cette option n'est pas fournie, `fxt_print` utilise le contenu de la variable d'environnement `TRACE_FILE` et en cas d'absence il lit le fichier *trace_file*

`-d`
: permet de demander à `fxt_print` de ne pas essayer de nommé les événements et les fonctions, mais d'afficher les identifiants (codes) tels qu'ils sont stockés dans la trace (utile surtout pour le débogage pour inspecter précisément le contenu d'une trace)

`-o`
: permet de demander à `fxt_print` d'afficher non plus la date des événements, mais le temps écoulé depuis le premier événement (les dates sont décalées de sortes que le premier événement survienne à la date 0)

Cette conversion de la trace au format binaire en une trace ASCII doit être faite sur la même architecture que la machine qui a récolté les traces.

4.2 Visualisation graphique

Les traces en ASCII peuvent ensuite être visualisées. Pour cela, elles doivent être mises en forme, éventuellement fusionnées (si on utilise FUT et FKT),

triées (les événements ne sont pas nécessairement enregistrés dans le bon ordre), etc. C'est le programme utilitaire `sigmund` qui est responsable de cela :

```
fxt_print [--user user_trace_file] [--kernel kernel_trace_file] [--paje]
```

`-user user_trace_file`
: indique une trace utilisateur (FUT) à traiter. Cette trace doit avoir déjà été convertie en ASCII avec `fxt_print`

`-kernel kernel_trace_file`
: indique une trace noyau (FKT) à traiter. Cette trace doit avoir déjà été convertie en ASCII avec `fxt_print`

`-paje`
: génère une trace complète au format Pajé

`sigmund` trie les traces, les fusionne si on lui en fourni deux, insère les événements d'ordonnancement manquants (prise de la main au début des traces et déordonnancement des threads à la fin), insère des pseudo-événements (création et destruction des threads/LWPs/processeurs/...), etc. Pour l'instant, il n'est pas possible de choisir un sous-ensemble d'actions à effectuer. La prochaine version de `sigmund` le permettra.

Enfin, par défaut, `sigmund` affiche cette super-trace complète. Avec l'option `-paje`, une étape supplémentaire est générée de façon à convertir cette super-trace en un format de trace acceptable par le logiciel de visualisation graphique Pajé.

Sur la figure 4.1, on peut observer un extrait du déroulement d'un programme multithreadé. Sur cet exemple, la machine a quatre processeurs physiques (il s'agit d'un biprocesseur Xeon SMT). L'application observée (nommée `traces`) utilise deux threads noyaux pour ordonnancer ses quatre threads utilisateurs : trois threads sont destinés aux calculs d'un produit de matrice (nommés `calcul1`, `calcul2` et `calcul3`) et le quatrième assure des communications. Les informations sont représentées dans trois zones matérialisant les points de vue thread utilisateur, thread noyau et processeur [1]. Détaillons ces différentes vues :

Informations sur chaque thread noyau :

1. identité du processeur physique utilisé (lorsque ce thread est ordonnancé par le système d'exploitation) ;
2. identité du thread utilisateur qu'il ordonnance (si le thread noyau exécute l'application observée) ;

Informations sur chaque processeur :

1. identité du thread noyau exécuté ;
2. identité du thread utilisateur exécuté (s'il s'agit de l'exécution de l'application observée) ;

Informations sur chaque thread utilisateur :

1. identité du thread noyau propulseur (lorsqu'il est sélectionné par l'ordonnanceur de l'application) ;

1. Si tous les threads noyaux et tous les processeurs du système sont représentés, seuls les threads utilisateurs de l'application espionnée sont observables.

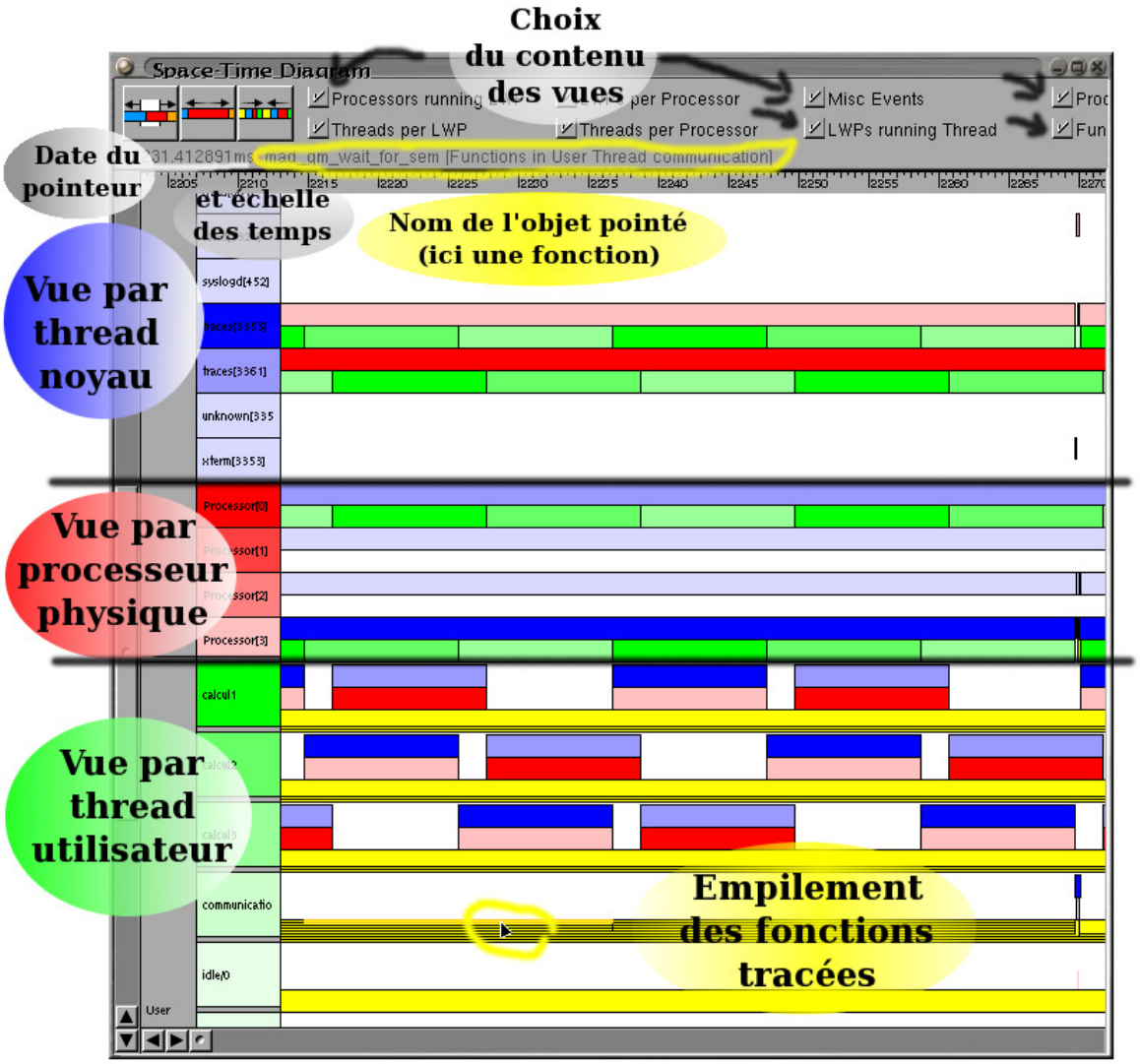

FIGURE 4.1 – Visualisation d'une trace avec PAJÉ

2. identité du processeur physique lorsqu'il est réellement exécuté (il doit alors être aussi ordonnancé par un thread noyau) ;
3. emboîtement des appels de fonction et des événements tracés dans l'application.

Chapitre 5

Perspectives

Le programme `fxt_print` et la bibliothèque `libfxt` vont être étendus pour permettre de lire des fichiers de traces produits sur une machine d'architecture différente.

Le programme `sigmund` va également être étendu afin de permettre de mieux manipuler les flots de traces : filtrages en fonction du type des événements, de dates, de l'émetteur des événements, etc.. Cela permettra de choisir facilement et précisément les informations que l'on souhaite visualiser dans le logiciel Pajé.

Les outils pour instrumenter les applications seront étoffées afin de permettre de décrire facilement, rapidement et de manière générique quelle représentation on souhaite dans Pajé pour chaque type d'événement de l'application.

Bibliographie

[1] Danjean (V.) et Wacrenier (P.-A.), « Mécanismes de traces efficaces pour programmes multithreadés », *TSI*, 2005. http://dept-info.labri.fr/~danjean/publications.html#DanWac05TSI.

[2] Thibault (S.) et Russell (R. D.), « Developing a software tool for precise kernel measurements ». Rapport technique, ENS Lyon, 2003. http://dept-info.labri.fr/~thibault/recherche.html.fr.