INGI1113: Projet Minix

Christoph Paasch, Fabien Duchêne

10 avril 2012

Table des matières

1 Introduction

Ce projet a pour but d'ajouter à Minix [2] un système de contrôle d'utilisation de ressources. Ceci est essentiel pour une utilisation équitable des ressources ou une utilisation sur un système limité. Dans ce projet, nous allons donc implémenter deux appels systèmes POSIX : `setrlimit` et `getrlimit` qui permettent respectivement de fixer et de récupérer la limitation sur une ressource.

Les objectifs poursuivis dans ce projet sont multiples. A la fin de ce projet vous devrez être capables de :

1. Comprendre le fonctionnement d'un système d'exploitation simple.
2. Comprendre l'exécution d'un appel système dans cet OS.
3. Déployer et utiliser un nouvel appel système quelconque.

2 Livre de Minix

La réalisation du projet demande de bien comprendre l'architecture du système, il est donc nécessaire, **avant de modifier le code**, d'avoir lu les sections du livre [2] décrites ci-dessous. Notez que cette liste est à la fois incomplète et trop complète : à vous d'identifier les parties qui s'avèreront cruciales pour votre projet.

A noter également : la quantité de lecture est assez importante, il faut donc prévoir commencer la lecture au plus tôt.

- Section 1.4 : Appels sytème.
- Section 2.5 : Architecture générale de Minix3, primitives de *message passing* (2.5.3). (Il est très important de bien comprendre la figure 2.29 p. 113).
- Section 2.6 : L'implémentation des processus dans Minix et conventions de codage (nous vous demandons de les respecter), format des messages (figure 2.34 page 143), privilèges attribués aux différents processus en matière de *message passing*.
- Section 2.7 : La tâche système.
- Section 4.7 : Le processus manager.
- Section 4.8 : L'implémentation du processus manager.

Il est utile de regarder (sans le modifier dans un premier temps) le code de Minix au fur et à mesure de la lecture. Notez que certaines indications du livre peuvent différer de la version de Minix installée dans les salles, car celle-ci est plus récente.

3 Utiliser Minix dans les salles

Qemu [1] est utilisé pour la virtualisation. Celui-ci utilise un disque dur virtuel qui contient le système Minix, équipé de vi, emacs, diff, gcc, etc. Cette image prenant trop de place (±140MB), elle est stockée en lecture seule dans un dossier partagé, accessible depuis toutes les machines des salles. Un système de *copy-on-write* (fichier .cow) est utilisé afin de garder localement les différences par rapport à cette image de base, limitant dès lors grandement l'espace requis.

Pour ce qui est du transfert de données vers l'environnement Minix, seu un échange unidirectionnel est possible : depuis la machine hôte vers la machine virtuelle. Vous ne devez donc pas modifier de fichier directement dans Minix. Des scripts ont été déployés expressément pour faciliter l'échange de donnés.

Il est à noter que Minix n'implémente pas les claviers belges. Il a donc été configuré pour supporter une disposition de clavier français. Les différences par rapport au clavier belge peuvent être trouvées à la figure 1.

3.1 Utilisation de Minix dans les Salles

Minix est disponible sur les machines des salles Siemens et Intel via Qemu un outil de virtualisation. Pour se faire un Makefile a été développé afin de vous faciliter la tâche et est disponible dans la section "Documents et liens/Projet_Minix".

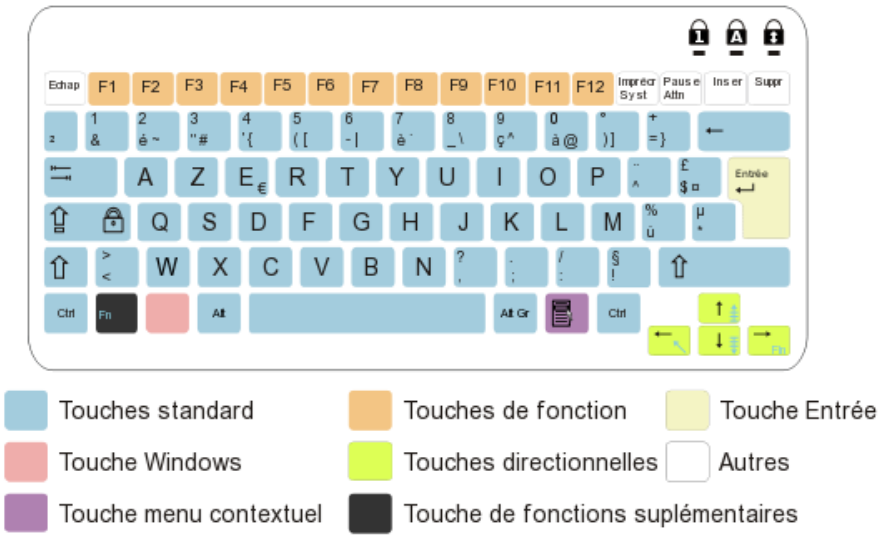

FIGURE 1 – Disposition azerty française pour PC portable (cfr. Wikipedia).

3.1.1 Initialisation du répertoire de travail

Une fois le Makefile téléchargé, il faut initialiser le dossier. L'initialisation consiste en trois étapes :

1. La création des disque dur virtuels `minix_local.cow` et `additional_disk.img`. Le premier correspond au disque principal qui est une version Copy-on-Write du disque principal. Le deuxième (qui apparaitera comme `/dev/c0d1` dans Minix) servira à créer un nouveau système de fichier mfs afin de tester le bon fonctionnement de vos appels systèmes.
2. La copie des sources de Minix dans le dossier `./src`. Les sources que vous devez modifier sont copiées, le reste est référencé via un lien symbolique.
3. La création d'un dossier `./test` qui contiendra, vos `scripts/codes` qui permettront de tester le bon fonctionnement de vos appels systèmes.

L'ensemble de ces étapes est réalisé en exécutant :

```
davidof:~/minix user$ make init
Creation du disque virtuel: minix_local.cow... done.
Creation du disque virtuel: additional_disk.img... done.
Creation du dossier: src... done.
Creation du lien symbolique: src_orig... done.
Creation du dossier: test... done.
davidof:~/minix user$ ls
additional_disk.img  Makefile  minix_local.cow  src  src_orig  test
```

Note : Le Makefile fourni les cibles suivantes :

```
davidof:~/minix user$ make
Utilisez 'make <target>' où <target> est :
  init         pour initialiser le répertoire du projet
  run          pour executer la machine virtuelle (en console)
  run_x11      pour executer la machine virtuelle (en fenêtre)
  patch        pour générer le patch
  dist         pour générer une archive comprenant le patch, le
               rapport et le dossier test
  clean        supprime l'archive et le patch
  mrproper     supprime les disques virtuels
```

Si vous voulez, par exemple, régénérer le disque dur virtuel `additional_disk.img`. Il vous suffit de le supprimer et d'ensuite appeler make init. Il en va de même pour tous les fichiers/dossiers qui se trouvent dans ce répertoire.

3.1.2 Exécution de la machine virtuelle

Une fois le dossier initialisé vous pouvez simplement modifier les sources de Minix dans le dossier `./src`. Une fois les sources modifiées vous devez démarrer la machine virtuelle contenant Minix. Pour ce faire :

```
davidof:~/minix user$ make run # ou make run_x11
...
Hit a key as follows:

    1  Start MINIX 3
    3  Start Custom MINIX 3
1
```

Au démarrage vous devez taper 1 afin de démarrer le noyau MINIX 3. Une fois la machine virtuelle lancée, vous pouvez vous loguer en tapant comme login root. A partir de ce moment vous devez configurer la machine virtuelle, afin qu'elle puisse communiquer avec l'hôte (à n'exécuter qu'une seule fois !) :

```
# echo "export HOST_USERNAME=<votre_login_des_salle>" >> .profile
# echo "export HOST_MINIXPATH=<chemin_vers_projet>" >> .profile
# . .profile
```

Pour mettre à jour à la fois les sources de Minix et le dossier test de la machine virtuelle :

```
# ./update_minix
```

Ce script utilise `rsync` sur `ssh` pour synchroniser les deux répertoires avec la machine hôte.

Une fois la machine mise à jour, vous pouvez re-compiler le système Minix :

```
# cd /usr/src/tools
# make libraries hdboot
```

Pour plus d'information sur comment recompiler le système :
`http://wiki.minix3.org/en/DevelopersGuide/RebuildingSystem`

Une fois que le système a été recompilé, vous devez redémarrer la machine :

```
# shutdown
...
MINIX will now be shut down ...
d0p0s0> menu
Hit a key as follows:

    1  Start MINIX 3
    3  Start Custom MINIX 3
3
```

La machine vas dès lors redémarrer sur votre nouveau noyau (Custom MINIX 3). Pour stopper proprement la machine, une fois que vous avez terminé, vous devez taper `off` à la place de `menu`.

Note : Il y a évidement moyen de faire tourner Minix sur vos machines personnelles. Cependant, cela est fortement déconseillé. En effet, aucun support ne sera fourni en dehors des machines des salles. Vous risquez également de ne pas respecter le format de la soumission, ce qui vous fera perdre des points inutilement.

N'oubliez pas que vous pouvez sans problème accéder aux machines des salles, même hors de l'UCL, en passant par Sirius et ensuite en vous connectant aux machines :

```
$ ssh -C user@sirius.info.ucl.ac.be # -X si run_x11

sirius:~ user$ ssh volcan10 # volcanXY ou intelXY // -X si run_x11
user@volcan10's password:

davidof:~ user$ cd minix
davidof:~/minix user$ make run
```

3.1.3 Soumettre votre projet

Votre rapport en version pdf doit se trouver dans le répertoire *projet_minix* du SVN et se nommer *rapport.pdf*.

4 Le Pré-Projet – un appel système basique

Afin de découvrir le code de Minix et donc pouvoir répondre aux questions, nous vous proposons d'implémenter ce pré-projet :

```
struct pid_s {
    pid_t me;
    pid_t parent;
};

/**
 * sortie: *pids* Une structure contenant le pid du processus appelant
 *            et le pid du parent.
 * valeur de retour: Renvoi toujours un code de succès.
```

```
 */
int getpidinfo(struct pid_s * pids);
```

L'objectif principal est de pouvoir identifier les fichiers à modifier afin d'y inclure ce nouvel appel système. Une bonne façon de comprendre comment un appel système est réalisé est de tracer son exécution (par ex, regardez ce qui se passe entre l'appel de read() par un processus user-space et son retour de fonction) et d'identifier ce qui se passe (où est défini l'appel système, comment est-il exécuté, comment est-il implémenté).

Note : Il n'y a rien à rendre pour ce pré-projet. Celui-ci a pour but de vous faciliter le travail pour le projet. Une permanence sera organisée après les vacances de pâques pour vous aider avec ce pré-projet.

5 Le Projet – limitation d'utilisation de ressources

Minix n'implémente pas de gestion de consommation de ressource. Il vous est dès lors demandé d'implémenter les deux appel système POSIX : `setrlimit` et `getrlimit`, suivant la définition suivante. Ces deux appels permettent respectivement de changer et de récupérer la limitation d'une ressource pour le processus courant.

5.1 "manpage"

SYNOPSYS

```
#include <sys/resource.h>

int getrlimit(int resource, struct rlimit *rlim);
int setrlimit(int resource, const struct rlimit *rlim);
```

DESCRIPTION

getrlimit() and **setrlimit()** get and set resource limits respectively. Each resource has an associated soft and hard limit, as defined by the *rlimit* structure (the *rlim* argument to both **getrlimit()** and **setrlimit()**) :

```
struct rlimit {
    rlim_t rlim_cur;  /* Soft limit */
    rlim_t rlim_max;  /* Hard limit (ceiling for rlim_cur) */
};
```

The soft limit is the value that the kernel enforces for the corresponding resource. The hard limit acts as a ceiling for the soft limit : an unprivileged process may only set its soft limit to a value in the range from 0 up to the hard limit, and (irreversibly) lower its hard limit. A privileged process (under Minix : one with **SUPER_USER**) may make arbitrary changes to either limit value.

The value **RLIM_INFINITY** denotes no limit on a resource (both in the structure returned by **getrlimit()** and in the structure passed to **setrlimit()**).

resource must be one of :

RLIMIT_CPU CPU time limit in seconds. When the process reaches the soft limit, it is sent a **SIGXCPU** signal. The default action for this signal is to terminate the process. However, the signal can be caught, and the handler can return control to the main program. If the process continues to consume CPU time, it will be sent **SIGXCPU** once per second until the hard limit is reached, at which time it sent **SIGKILL**.

RLIMIT_NICE Specifies a ceiling to which the process's nice value can be raised using **setpriority()** or **nice**. The actual ceiling for the nice value is calculated as *20 - rlim_cur*.

RLIMIT_NPROC The maximum number of simultaneous processes that can be created for the real user ID of the calling process. Upon encountering this limit, **fork()** fails with the error **EAGAIN**.

RLIMIT_FSIZE Specifies the maximum size (in Bytes) of files that the process may create. Attempts to extend a file beyond this limit result in delivery of a **SIGXFSZ** signal. By default, this signal terminates a process, but a process can catch this signal instead, in which case the relevant system call (e.g., **write()**, **truncate()**) fails with the error **EFBIG**.

RLIMIT_NOFILE Specifies a value one greater than the maximum file descriptor number that can be opened by this process. Attempts (**open()**, **pipe()**, **dup()**, etc.) to exceed this limit yield the error **EMFILE**.

RETURN VALUE

A 0 return value indicates that the call succeeded, changing or returning the resource limit. A return value of -1 indicates that an error occurred, and an error code is stored in the global location *errno*.

ERRORS

The **getrlimit()** and **setrlimit()** system calls will fail if :

EFAULT The address specified for *rlim* is invalid.

EINVAL *resource* is invalid.

The **setrlimit()** call will fail if :

EINVAL The specified limit is invalid (e.g., *rlim_max* lower than *rlim_cur*).

EPERM The limit specified would have raised the maximum limit value and the caller is not the super-user.

NOTES

A child process created via **fork()** inherits its parents resource limits. Resource limits are preserved across **execve()**.

5.2 Indications

Nous attirons votre attention sur quelques points particulièrement importants :

- Toutes les limitations de ressources ne sont pas obligatoires, RLIMIT_CPU est considéré comme un bonus du fait de sa complexité à être implémenté.
- Comme Minix est divisé en plusieurs parties (kernel, pm, fs, etc.) et que les ressources ne sont utilisées que dans une de ces partie à la fois, il vous est demandé de ne stocker que les informations utiles dans les parties adéquates.
- De même, vous devez bien faire attention à ce que l'appel système soit correctement dispatché vers la partie adéquate (comme l'est `fork()` par exemple). Exemple, dans le cas de RLIMIT_FSIZE, seul le serveur FS doit être contacté.
- L'information doit être liée à chaque processus. Il vous faudra donc initialiser correctement chaque limite par une valeur par défaut. Faites bien attention à cette valeur par défaut, Minix défini déjà certaine limites "dures" (ex, `OPEN_MAX`, etc.). (*kernel/proc.h*) ont déjà été modifiés, dans le but de limiter le temps de compilation. testera que l'ensemble est bien implémenté. Exemple, setrlimit() → fork() → getrlimit() et vérification que l'on a la même valeur.

5.3 Délivrables – modalités pratiques

A la fin du projet (sur SVN dans le répertoire `projet_minix`, pour le Lundi 14/05 avant 23h59), un rapport de maximum 8 pages décrivant votre solution doit être fourni. Il décrira l'architecture de votre implémentation, justifiera vos choix et donnera toute information utile à la bonne compréhension de votre travail. Il ne devra pas reprendre des éléments de l'énoncé ou des parties du livre de référence.

Vous devrez évidemment aussi soumettre votre code source. Pour cela vous générez un patch avec la commande make :

```
davidof:~/minix user$ make dist
Génération du patch: minix_3.1.8r3_nfrags_defrag_user.patch... done.
Génération de l'archive: projet_minix_user.tar.gz... done.
```

Ceci générera une archive `projet_minix_${USER}.tar.gz` qui contiendra le patch de vos modifications, votre rapport, ainsi que votre dossier test. Verifiez cependant que tout le contenu demandé se trouve dasn votre archive avant de soumettre.

5.4 Evaluation

Les points vous seront attribués sur base :

1. De vos choix architecturaux. Essayez de fournir la meilleure solution possible, tout en vous assurant qu'elle est faisable dans le temps prévu.
2. De la correction du code source.
3. Des commentaires.
4. Du respect des conventions de codage MINIX (voir section 2.6 de [2])
5. De la qualité du programme de test : celui-ci devra tester de la manière la plus claire et complète possible les nouvelles fonctionnalités que vous aurez ajoutées.

6 Documentation

6.1 Où trouver de la documentation

Les sources principales d'informations sont le livre [2] et le code source de Minix, très bien documenté. Lors de ce projet vous serez amenés à lire beaucoup de code C. Cela fait partie intégrante du développement dans des systèmes informatiques de grande taille comme le sont les systèmes d'exploitation.

Références

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

[2] Andrew S. Tanenbaum and Albert S. Woodhull. *Operating Systems Design and Implementation (3rd Edition) (Prentice Hall Software Series)*. Prentice Hall, January 2006.