Université Catholique de Louvain
Ecole Polytechnique de Louvain
Année 2011-2012

Système Informatiques 2 :
Projet MINIX

RAPPORT DE PROJET : LINGI1113

Présenté par : Michael Azzam et Yannick Tivisse

1 Introduction

Dans le cadre de ce projet, il nous été demandé d'implémenter les appels systèmes **getrlimit** et **setrlimit**, qui permettent de récupérer la limitation d'une ressource pour un processus courant ou la modifier. Le rapport se divise en deux parties : Nous expliquons dans un premier temps comment nous avons ajouté ces appels système dans l'OS et finalement nous expliquons les choix architecturaux que nous avons pris pour l'implémentation de ces deux appels.

2 Ajout d'un appel système

La première étape consiste en la création de fonctions dans les librairies utilisateurs qui feront le lien entre l'utilisateur en mode normal et les appels systèmes à proprement parler qui s'exécutent en mode noyau.

Nous avons déclaré les prototypes de ces fonctions dans le fichier **include/sys/resource.h**, et nous définissons également la structure **rlimit**, et les différentes constantes qui nous seront utiles par la suite. Le fichier **include/sys/types.h** a également été modifié pour définir le type **rlim_t**.

Nous avons ajouté les fichiers implémentant les fonctions dans le dossier **lib**. Notamment dans le dossier **lib/libc/posix** où nous avons ajouté les fichiers **_getrlimit.c** et **_setrlimit.c**, qui nous permettent d'envoyer l'appel dans le serveur **PM** ou **FS** en fonction de la valeur de la variable **resource** et grâce aux définitions ajoutées dans **include/minix/callnr.h**.
Les fichiers **getrlimit.S** et **setrlimit.S** ont été ajouté dans **lib/libc/syscall** et servent lors de l'appel système à dévier l'exécution du programme vers le code qui concerne cet appel système. L'exécution sera ici toujours envoyée vers les fichiers que nous avons ajouté précédemment dans la librairie.

Nous avons fini de modifier les librairies et l'exécution est bien envoyée vers le bon serveur dans chaque cas d'exécution. Maintenant, il faut modifier ces serveurs pour qu'ils effectuent le travail désiré.
Nous avons apporté les modifications qui vont suivre aux deux serveurs **PM** et **FS**. Il faut ajouter les prototypes des appels systèmes dans le fichier **servers/pm;fs/proto.h**. Il faut également associer les nombres dédiés à nos appels systèmes avec les fonctions qui les exécute. Ceci se fait en modifiant le fichier **servers/pm;fs/table.c**.

Il ne faut bien entendu ne pas oublier de modifier tous les **Makefile** dans les dossiers où nous avons ajouté des fichiers. Maintenant, il ne reste plus qu'à implémenter nos fonctions dans **servers/pm;fs**

3 Choix architecturaux par ressource

Nous avons fait le choix de donner à **RLIM_INFINITY** la valeur maximale pouvant stockée dans un **unsigned long** étant donné que toutes les limites de ressources sont toujours positives. C'est notamment la raison pour laquelle nous avons choisi de représenter **rlim_t** par un type **unsigned long** et non **signed long**. Ce faisant, nous avons accès à une plus grande plage de valeurs positives pour les limites. Certaines comparaisons seront de ce fait simplifiées

en prenant **RLIM_INFINITY** comme telle.

Nous vérifions que les valeurs données pour la *soft limit* ne dépasse jamais la valeur de la *hard limit*, et qu'elles soient positives. Dans le cas où la nouvelle limite est inférieure à la ressource actuelle, par exemple si la nouvelle limite du nombre de fichier ouvert est en dessous du nombre actuel de fichiers ouvert, nous avons décidé d'abandonner l'exécution de la fonction et de retourner un code d'erreur **EINVAL**. La norme POSIX ne spécifiant pas le comportement dans ce ce type de demande anormale, il nous a semblé préférable d'informer le programmeur de la situation.

Dans le même ordre d'idée, nous avons fait le choix d'interdire les modifications de priorité qui n'auraient pas de sens. Par exemple, pour **RLIMIT_NOFILE**, il n'aurait pas beaucoup de sens d'établir une limite supérieur au nombre maximal de fichiers ouverts (**OPEN_MAX**).

Nous vérifions également que seulement un **SUPER_USER** puisse modifier la hard limit d'uine ressource, comme spécifié.

Nous n'avons pas traité le passage des limites de père à fils étant donné que ceci est déjà géré dans les serveurs **PM** et **FS** dans les fichiers **servers/pm/forkexit.c** et **servers/vfs/misc.c**.

3.1 RLIMIT_NICE

Nous allons gérer cette ressource dans le serveur **PM**, étant donné que **nice** y est implémenté aussi. Etant donné que c'est un limite propre à chaque processus, nous avons ajouté la **struct rlimit mp_nicelimit** dans le fichier **mproc.h** définissant les informations concernant chaque processus en rapport avec le serveur **PM**. L'initialisation de cette structure se fait dans le fichier **main.c**.

Nous avons modifié le fichier **server/pm/misc.c** dans lequel est implémenté la méthode **setpriority**. Nous vérifions que les limites soient comprises entre 0 et 40 ou égales à **RLIM_INFINITY**, et que les priorités modifiées ne dépassent pas la limite imposée. Nous avons choisi 40 comme valeur par défaut de la limite car nous n'avions pas de raison de restreindre l'utilisateur de prime abord.

3.2 RLIMIT_NPROC

Pour gérer la limite de la ressource, nous gardons en mémoire un compteur du nombre de processus courants pour chaque utilisateur, ainsi que la limite.

Pour ce faire, nous avons défini la structure chaînée **nproclimlist** qui contient ces informations dans le fichier **servers/pm/mproc.h**. Il nous a semblé plus intelligent de procéder ainsi plutôt que d'utiliser un tableau de taille fixe, car il n'y a pas de limitation du nombre d'utilisateurs. Cette structure est initialisée dans le fichier **servers/pm/main.c**.

Nous avons modifié les fichiers **servers/pm/forkexit.c** et **servers/pm/getset.c** pour les méthodes **fork**, **exit** et **setuid**. Le compteur est incrémenté lors de l'appel de fork, décrémenté lors de l'appel d'**exit**, et lors de l'appel de **setuid**, nous décrémentons le compteur de l'ancien utilisateur, et incrémentons le compteur du nouvel utilisateur. Lors de l'appel de **setuid**, si l'**uid** apparaît pour la première fois, un nouveau noeud est créé dans la structure chaînée.

3.3 RLIMIT_FSIZE

Nous sommes pour cette ressource dans le serveur **VFS**. Nous avons déclaré la variable **fp_fsizelimit** de type **rlimit** dans le fichier **servers/vfs/fproc.h**, et nous initialisons les deux champs de la structure dans le fichier **servers/vfs/main.c** à la valeur **RLIM_INFINITY**.

Nous modifions les fichiers **servers/vfs/read.c** et **servers/vfs/link.c** pour les méthodes **write** et **truncate**. Pour l'appel de la fonction **write**, nous regardons où se place le curseur dans le fichier pour l'écriture de donnée, et nous y ajoutons le nombre d'octets que le processus va écrire. Nous vérifions ainsi que l'écriture n'engendrerait pas un dépassement de la limite.

Dans le cas ou la taille finale du fichier est trop grande, nous envoyons le signal **SIGXFSZ** au processus appelant via la fonction **kill**, tuant le processus dans le cas où le signal n'est pas attrapé. Nous avons déclaré le signal dans le fichier **include/signal.h** Dans le cas de l'appel de **truncate**, nous pouvons simplement vérifier la taille finale du fichier en allant chercher le dernier paramètre de l'appel qui n'est rien d'autre que la taille finale souhaitée.

3.4 RLIMIT_NOFILE

Cette ressource est liée au serveur **VFS**. Etant donné que cette limite est propre à chaque processus, nous modifions le fichier **servers/vfs/fproc.h** en y ajoutant le champ **rlimit fp_nofilelimit**. Cette limite sera initialisée pour chaque processus dans le fichier **servers/fs/main.c**, que nous avons modifié à cet effet.

Nous avons choisi d'ajouter une variable qui compte le nombre de file descriptor ouverts par le processus. Cela nous a semblé être un solution simple et efficace. Lors d'un appel système, nous vérifions les cas suivant :

- Si le processus crée un nouveau file descriptor, vérifier s'il ne dépasse pas la limite
- Si le processus baisse la limite en dessous du nombre de file descriptor déjà ouverts, nous retournons un code d'erreur EINVAL.
- Si le processus essaye de placer une nouvelle limite en dehors des valeurs autorisées, nous renvoyons une erreur EINVAL.
- Si le processus essaye de monter la limite alors qu'il n'est pas **SUPER_USER**, nous renvoyons le code d'erreur EPERM

.
Pour implémenter ces appels, nous avons modifiés les fichiers où il va falloir contrôler la création de file descriptor. C'est-à-dire :

- **servers/vfs/misc.c** pour les fonction dup et fcntl
- **servers/vfs/open.c** pour les fonction open et close
- **servers/vfs/pipe.c** pour la fonction pipe

.
La valeur par défaut que nous avons choisi pour cette limite est la taille maximale du tableau **fp_filp**, qui est représenté par la macro **OPEN_MAX**, et qui vaut 255.