Année scolaire : **2004-2005**
Date : **26/11/2004**

Module ISI 423 : **Système d'exploitation et son support architectural**

TP MINIX : Découverte de MINIX et gestion de la mémoire.

1 INTRODUCTION

Solaris MINIX (smx) est une version du système d'exploitation MINIX (type *NIX) qui tourne à l'intérieur d'un processus du système UNIX Solaris des stations SUN. Le processus Solaris joue le rôle d'une machine virtuelle (en fait MINIX utilise un deuxième processus pour des détails très techniques, mais oublions). Pour l'utilisateur, MINIX est un système d'exploitation normal. Du fait que vous avez accès à l'intérieur de MINIX. et que vous disposez des droits du super utilisateur root, vous serez à la fois utilisateur normal, administrateur système et ingénieur système UNIX. En particulier, il vous sera possible de modifier les fichiers sources du système MINIX et de le recompiler sous Solaris.

2 INSTALLATION

Vous avez le choix entre une installation sur votre compte ***2A*** ou en local dans ***/var/tmp***. Avant d'installer minix, faites `df -k` pour voir le taux d'occupation des disques à tout hasard. Minix a besoin de 6 Méga octets.

Pour installer MINIX (mettre en place quelques variables utiles)
- sur le disque **2A**, vous devez taper : `SETUP MINIX`
- sur le disque **/var/tmp**, vous devez taper : `SETUP MINIX3`

Attention, /var/**tmp** est local (non sauvegardé) sur chaque station. Mais son avantage est qu'il y a de la place en général...

L'option `-v` du SETUP, vous permet de voir les variables que cette commande met en place.

Dans le cas de **/var/tmp**, il faut qu'un répertoire à votre nom existe sous **/var/tmp** ; pour cela, tapez la commande :

```
mkdir /var/tmp/`whoami`
```

Une fois ceci fait, cd $MINIX_HOME vous permettra d'aller directement à ce répertoire (emplacement où seront les sources et l'exécutable de votre MINIX).

Faire ensuite :

`SETUP SUNPRO3` (compilateur de Sun très vieux mais gratuit encore utile pour compiler le noyau de minix lors des TPs...)

`install_minix` (petit programme qui copie les fichiers utiles)

install_minix place une arborescence de fichiers dans le répertoire minix. Pour en savoir plus sur les SETUP et le programme *install_minix*, reportez-vous à l'annexe 2.

Pour éviter de taper les SETUP précédents à chaque fois qu'on se connecte on pourrait les rajouter dans son ~/.cshrc.PERSO.

3 UTILISATION DE MINIX

3.1 Lancer MINIX

Pour lancer MINIX, utilisez la commande `minix` (auparavant créez un sous-répertoire de travail chez vous, en dehors du répertoire minix, et allez dans ce répertoire. Ainsi c'est dans ce répertoire que seront écrits les fichiers par les commandes sunread/sunwrite qu'on verra par la suite). La commande `minix` lancera une session de MINIX (ayant pour nom minix) dans la fenêtre texte où vous taperez cette commande (sur la machine où vous êtes connecté). Ne lancez pas MINIX deux fois en même temps. Vous pouvez, par contre, ouvrir un deuxième terminal (type xterm) sur la même session MINIX en tapant `mlogin minix` , sans oublier de faire le SETUP auparavant, dans une autre fenêtre de la même machine (où minix est le nom de votre session minix, cf. annexe 3).

3.2 Quitter MINIX

Après avoir un peu regardé ce nouvel OS, vous pourrez le quitter en tapant `shutdown` depuis MINIX (une fois connecté en tant que root). `shutdown` ferme toutes les fenêtres MINIX ouvertes. La fermeture d'une fenêtre MINIX s'effectue à l'aide de la séquence de touches 'quit' en maintenant la touche Ctrl enfoncée (c'est à dire Ctrl Q, Ctrl U, Ctrl I et Ctrl T ! Incroyable, non ?).

Attention : ne pas fermer une fenêtre en utilisant la commande quit du menu du coin en haut à gauche de la fenêtre (surtout si c'est la dernière fenêtre MINIX). C'est la commande `shutdown` qui ferme proprement le système MINIX).

En cas de problèmes, vous pouvez "tuer MINIX" de deux façons : à partir de Solaris en utilisant l'instruction `kill` (à éviter) et à l'aide des touches 'Ctrl-7' (préférable) qui vous affichera quelques renseignements sur le système avant de le terminer proprement. Il faut noter que si une fenêtre MINIX est bloquée, vous avez toujours accès au système à l'aide de la commande `mlogin`.

Attention, rien ne vous empêche de détruire votre système (puisque vous avez tous les droits) ; en cas de problème, vous pourrez toujours remettre votre système dans son état de base (le disque monté à la racine) en tapant sous Solaris :

```
cp $MINIX/disks/root $MINIX_HOME/minix/disks/root
```

3.3 Après une séance de travail

En cas de "surcharge" de disques, on vous conseille d'effacer le répertoire minix (rm -rf minix) chez vous. Cela permettra à chacun de pouvoir travailler.

ATTENTION : cela signifie qu'il faut sauvegarder chez vous, en dehors de minix, tous les fichiers que vous avez dû modifier si vous effacez ce répertoire. En particulier il faut le faire systématiquement si vous avez travaillé par exempl e dans **/var/tmp**.

3.4 Sous smx (small unix)

Quand vous aurez lancé une session MINIX, on vous demandera de vous connecter (login). La première fois, tapez root (il n'y a pas de mot de passe).

3.4.1 Root et mot de passe

Vous êtes alors l'administrateur de votre système d'exploitation (il est conseillé d'associer à root un mot de passe par la commande passwd).

A partir de ce moment, vous avez à votre disposition deux systèmes d'exploitation : Solaris d'une part et MINIX d'autre part, les fichiers de l'un sont invisibles de l'autre. Les commandes `sunread` et `sunwrite` permettent de transférer des fichiers d'un système à l'autre (annexe 3).

3.4.2 Processus MINIX et processus Solaris

Sous Solaris, faites la commande `ps -aef`. Combien de processus exécutant MINIX trouvez-vous? Faites ps -axl sous smx. Que notez-vous? Quels sont les ancêtres de ces processus minix?

3.4.3 Ouvrir plusieurs fenêtres sur MINIX

Ouvrez une deuxième fenêtre sous Solaris et tapez la commande `mlogin minix`.[1] Refaites `ps -aef` sous Solaris. Y-a-t-il des changements? Refaites ps -axl dans cette nouvelle fenêtre smx. Il y a bien un seul minix en fonctionnement avec 2 fenêtres interprétant les commandes shell. Quel est le nombre maximum de fenêtres que vous pouvez avoir? Avec l'option `-u` de `ps` vous pouvez avoir la liste des processus que possède un usager donné.

```
ps -u rannou
```

Attention :
Il arrive souvent que des élèves se déconnectent de Solaris alors qu'ils possèdent encore des processus qui tournent. C'est le cas en particulier lorsque les élèves sont mal sortis de MINIX (ou qu'ils oublient de quitter les éditeurs de textes)
Ces processus vont ralentir la station pour les collègues qui suivent., ceux-ci ne possédant pas les droits pour les détruire.

3.4.4 Rajouter un utilisateur.

Vous disposez pour cela de la commande adduser. Faites man adduser pour connaître son fonctionnement. Regardez le fichier */etc/passwd* qui contient la liste des utilisateurs et */etc/shadow* pour les mots de passe (fichier illisible pour les simples utilisateurs). Cette commande nécessite que le répertoire contenant le répertoire de l'utilisateur existe déjà.

Vous pouvez dans un premier temps utiliser un groupe existant (other par exemple). Une fois le nouvel utilisateur introduit, connectez vous sous cet utilisateur (dans la même fenêtre ou mieux dans une autre). Associez à ce nouvel utilisateur un mot de passe.

3.4.5 Créer un nouveau groupe et rajouter un utilisateur de groupe.

Pour cela, rajoutez une ligne au fichier /etc/group.

[1] Cette commande se trouve dans $MINIX_HOME/minix/bin

Quand vous travaillez sous MINIX, vous avez le choix entre utiliser un des éditeurs de texte de MINIX

- ed,
- elvis (une sorte de vi)
- uemacs (une sorte d'emacs),

ou travailler avec un éditeur sous Solaris (emacs, textedit, ...).

Dans ce dernier cas, comme les fichiers MINIX ne sont pas visibles sous Solaris, vous devrez faire usage des commandes MINIX `sunread` (pour un transfert Solaris vers MINIX d'un fichier), ou `sunwrite` (pour un transfert MINIX vers Solaris). Voir annexe 3 et la commande `man` sous MINIX.

Attention, pour ces deux commandes, un nom relatif de fichier Solaris désigne le répertoire où vous étiez lorsque vous avez lancé la commande `minix`. C'est pourquoi on vous a fait créer un répertoire *travail* à partir duquel vous avez lancé MINIX par la commande `minix`.

3.4.6 Faire advent.

Le but ici est de jouer en cours au jeu d'aventure **advent**. Profitez-en c'est la seule fois où ce sera autorisé ! Mais après tout MINIX est lui même un système d'exploitation jouet...

4 ALLOCATION DE MÉMOIRE

4.1 Comment modifier le système et reconstruire le système

Les sources du système qui vous concernent sont placées dans 4 répertoires sous Solaris dans $MINIX_HOME/src.

- ***fs*** le système de gestion de fichier
- ***init***
- ***kernel*** le noyau du système MINIX
- ***mm*** *la gestion de la mémoire* .

Vous avez un imprimé de quelques pages qui décrit sommairement à quoi correspondent tous les fichiers *source*. Vous n'avez pas encore besoin de savoir beaucoup de choses.

Quand vous aurez modifié, l'un quelconque des fichiers *source*, il suffit d'aller dans le répertoire $MINIX_HOME/minix/src/tools et de faire make image pour construire un minix exécutable. Félicitation ! Vous venez peut etre de compiler le premier système d'exploitation de votre vie...

4.2 Allocation de la mémoire

Pour commencer, vous allez regarder le répertoire *mm* (memory manager ou gestionnaire de mémoire). Il s'occupe notamment d'allouer et de désallouer la mémoire pour les processus en maintenant une table de l'espace mémoire disponible. Ceci est réalisé dans le fichier *alloc.c*.

MINIX n'utilise ni les techniques de pagination ni celles de segmentation. Lorsque l'on crée un processus sous MINIX, le gestionnaire de mémoire recherche un bloc de taille

suffisante dans l'espace mémoire accordé à MINIX par le système Solaris. Le module alloc.c gère une liste des blocs libres (hole_head).

L'unité de mémoire est le click (le click vaut un certain nombre d'octets).

Le fichier alloc.c contient 4 points d'entrées :

- phys_clicks alloc_mem (phys_clicks *clicks*). Alloue *clicks* unités de mémoire et rend comme résultat l'adresse du bloc.
- void free_mem (phys_clicks *base*, phys_clicks *clicks*). Libère un bloc de *clicks* unités situé à l'adresse *base*.
- void mem_init (total, free). Effectue l'initalisation. A la sortie de cette fonction, la liste des blocs libres est constituée. On ne dispose pas forcément d'un seul bloc.
- phys_clicks max-holes (). Donne la taille du plus grand bloc.

L'invariant de ce module stipule :

- qu'il n'y a pas deux blocs libres contigus. A l'initialisation et à chaque fois qu'un bloc est libéré, on effectue la fusion (fonction interne merge) des blocs contigus.
- que la somme espace disponible - espace alloué reste constant.

4.3 Regarder ce qui se passe

Dans le fichier *alloc.c*, il y a plusieurs commandes d'impression (printf) commentées ; si vous décommentez ces lignes, et que vous compilez MINIX (make image), vous verrez s'afficher de nombreux messages concernant l'allocation de mémoire (il est conseillé de ne pas utiliser la fenêtre principale qui sera saturée de messages, mais plutôt de faire un mlogin, il est aussi conseillé de ne pas décommenter tout d'un seul coup).

Une fois connecté, vous pouvez voir l'allocation de mémoire pour un petit programme (`mm.test1` par exemple) qui ne fait pas grand chose. Vous pouvez ensuite utiliser autre chose (`mm.test2` par exemple) pour voir la différence (vous avez les sources commentées de ces deux programmes en annexe). Vous pouvez aussi voir le travail du gestionnaire de mémoire pour de simples commandes telles que `ls`, `ps` ou `tcsh`.

Je vous rappelle que l'utilisation de '`&`' à la fin d'une commande la lance en tâche de fond et vous permet donc de lancer plusieurs programmes à la fois (pour les enlever, utilisez les instructions `ps` et `kill`).

4.4 Pour bloquer le système

Toujours dans le même fichier, vous avez une variable NR_HOLES qui indique en combien de morceaux la mémoire libre peut se découper ; vous pouvez la modifier (en dessous de 6 votre système ne devrait plus se lancer, et vers 8-9 il devrait poser des problèmes dans certaines conditions d'utilisation, regardez).

Attention, quand vous modifiez des valeurs, veillez à les remettre à leur valeur par défaut, votre système risquerait de ne pas toujours apprécier.

4.5 Etude de la gestion de la mémoire *(si nous avions plus de temps...)*

Vous avez, en annexe 1, la description de plusieurs algorithmes de choix pour l'allocation de la mémoire ; en modifiant la fonction d'allocation de la mémoire dans le fichier *alloc.c*, vous essaierez ces divers algorithmes et les comparerez du point de vue rapidité (vous pourrez utiliser les programmes de test à votre disposition associés à la fonction `time` par exemple) connaissant leur efficacité respective.

Vous pouvez aussi accéder à l'URL suivante pour une description d'algorithmes, en particulier on y décrit le principe d'un algorithme utilisé dans les systèmes actuels : le buddy system.:

http://www.cs.wisc.edu/%7ecs537-1/memory.html#core_algorithms

Quand la fonction alloc_mem est incapable de trouver un bloc libre, cela ne veut pas dire qu'il n'y a pas d'espace libre. La proportion d'espace libre sur l'espace total peut même être très importante. Vous pouvoir imprimer cette proportion si vous le souhaitez. Cela pose la question : pourquoi ne pas chercher à déplacer les blocs occupés (processus) quand il reste de la place, mais de façon trop fragmentée?

A vous d'effectuer une simulation de ces algorithmes pour les comparer, vous pouvez "extraire" ces algorithmes de MINIX pour avoir plus de souplesse pour effectuer les tests.

Les binômes qui ont ce travail sur l'allocation mémoire à réaliser devront faire une présentation de l'algorithmes : buddy.

Annexe 1

Algorithmes d'allocation de la mémoire

first fit : il prend le premier emplacement assez grand (c'est celui utilisé par MINIX et c'est le plus simple).

next fit : il recommence à chercher un emplacement libre là où il s'était arrêté la fois précédente (assez rapide dans un premier temps, mais favorable à la fragmentation de la mémoire).

best fit : il cherche le plus petit bloc qui soit assez grand pour l'allocation (assez lent mais un certain gain sur l'occupation de la mémoire).

worst fit : il prend systématiquement le plus grand emplacement disponible (a priori pas le plus intéressant, car pas très rapide et pas trop efficace).

Annexe 2

SETUP et programmes d'installation

- SETUP MINIX : ce SETUP définira les variables d'environnement MX_INCL (répertoire include), MX_LIB (les librairies propres à MINIX), MINIX (le chemin des sources globales de MINIX), MINIX_HOME (où se trouve votre répertoire minix) ; il modifiera aussi votre variable PATH pour avoir accès aux utilitaires MINIX disponibles sous Solaris. Voici donc le résumé de ce SETUP :

```
setenv MINIX          ************
setenv MX_LIB         $MINIX/lib
setenv MINIX_HOME     $HOME
setenv MX_INCL        $MINIX_HOME/minix/include
setenv PATH           $PATH:$MINIX/bin
```

Vous pouvez changer le répertoire où MINIX sera installé chez vous en modifiant les variables MINIX_HOME et MX_INCL.

- SETUP SUNPRO3 : utilisé pour compiler le noyau de MINIX (compilateur cc).

- install_minix : programme en shell qui installera chez vous l'arborescence de MINIX en copiant le minimum de fichiers et en faisant donc un maximum de liens. Il crée aussi un fichier *minix.config* type dans le répertoire *minix*.

Annexe 3

Programmes de communication avec Solaris

- minix :

minix [-d] [-force] [-m (none|half|full)] [config-file]

Ce programme permet de lancer le système d'exploitation MINIX à l'intérieur d'un processus UNIX.

Vous pouvez paramétrer ce système en modifiant le fichier *minix.config* qui doit être installé chez vous en même temps que le reste des sources pouvant vous être utiles (Utilisez le programme *install_minix* à cette fin).

- mlogin :

mlogin hostname

Ce programme permet de se connecter a un système MINIX déjà lancé sur la machine. Vous devez lui passer en argument le nom de la session MINIX indiqué dans le fichier *minix.config* (faites mlogin minix). Vous pouvez ainsi avoir plusieurs consoles MINIX à la fois.

Pour quitter cette console, il faut taper 'quit' en tenant la touche Ctrl enfoncée.

- sunread :

sunread file_Solaris > file_smx

Lit un fichier Solaris (chemin relatif à partir du répertoire de lancement de minix) et l'écrit dans un fichier sous MINIX ; les fichiers en C doivent être compilés avec `mcc` pour pouvoir tourner sous MINIX.

- sunwrite :

sunwrite file_Solaris < file_smx

Fait le contraire de sunread.

Annexe 4

Programmes de test

Un certain nombre de programmes de tests vous sont fournis. Ces programmes sont peu satisfaisants On vous demande cependant d'en déduire des compratifs.

Il existe 2 versions de chaque programme : une qui se termine et une qui entre dans une boucle infinie pour que le processus conserve toutes ses données (en rajoutant un b au nom).

- mm.test1.c :

```
#include <stdio.h>
int main (int argc, char **argv)
{
```

```
/* Petit programme qui alloue juste la quantite de memoire demandee *
 * On doit entrer en parametre la quantite de memoire desiree       */
  char *tableau;
  int taille;
  int i;
  if (argc!=2)
  {
    fprintf (stderr, "\nUsage : %s taille\n", argv[0]);
    exit (0);
  }

  taille=atoi(argv[1]);
  tableau=(char *)calloc(taille*1024,1);
  if (tableau!=NULL)
     fprintf
(stderr,"\nJ'ai alloue %d octets de memoire\n",     taille*1024);
}
```

- mm.test2.c :

```
#include <stdio.h>
main()
{
/* Ce programme recupere le maximum de memoire jusqu'a ce qu'il ne  puisse
plus. A n'utiliser que pour faire des tests.                          */
    int i,somme;
    char *tab;
    int size = 1024;
/* taille de depart de l'allocation de base */
    somme=0;
    for (;size>1;) {
        if ((tab=(char *)calloc(size,1)) == NULL)
           size /= 2;
/*plus de place, on cherche a allouer 2  fois moins*/
        else {
           somme=somme+size;
           fprintf
(stderr,"\nCool, je peux allouer %d octets...,"
            ,"Donc %d en tout.",size,somme);
           size *=2;
/*Y'a de la place, on en veut encore plus*/
        }
    }
}
```

- mm.test3.c : alloue de la mémoire puis en désalloue une partie; vous pouvez lui passer des paramètres indiquant la taille des morceaux de mémoire et le pourcentage de mémoire qui sera libéré.

TP MINIX : Driver de disque.

2. QUE VA-T-ON VOIR ICI ?

Vous avez pu découvrir le système MINIX dans le TP précédent ; maintenanr, nous allons nous interesser à un autre problème des systèmes d'exploitation : les drivers (pilotes d'entrées-sorties).

Au fait à quoi cela sert d'écrire un contrôleur de périphérique ?

2.1. Principe

Le driver sert d'interface entre le système d'exploitation et un périphérique quelconque. Sous MINIX, les périphériques sont gérés en 2 étapes :

- La première est indépendante du périphérique. Elle se situe dans le gestionnaire de fichiers fs et dans le "kernel" (fichier device.*).
- L'autre partie en est totalement dépendante. Elle est dans le kernel (fichier sun*). C'est le driver ou pilote d'entrée-sortie.

Ainsi, pour rajouter un gestionnaire, il suffit de rajouter la partie dépendante en respectant les règles dictées par la partie indépendante.

2.2. Spécificité de MINIX

Le problème qui se pose à nous est dû au fait que MINIX est construit de façon particulière. Nous avons une version de MINIX qui tourne par dessus un autre OS. Nous n'avons pas affaire à de vrais périphériques, de vrais interruptions matérielles, etc. Les périphériques sont supportés par Solaris. C'est par exemple le répertoire MINIX/disks sous Solaris qui supporte les 3 partitions disque du système smx (à savoir les pages du man, les binaires (usr/bin) et la partition root.

Dans ces circonstances, pour réaliser un driver smx, il faut savoir communiquer avec Solaris en respectant des règles de communication avec Solaris.

3. AVANT DE VOUS LANCER DANS LE TRAVAIL

3.1. Un fichier spécial, qu'est-ce ? Notion de majeur et de mineur

a) Lancer MINIX et aller voir du côté du répertoire /dev de minix. Si vous faites ls -l, vous verrez des fichiers spéciaux, correspondant à des périphériques. Il y a deux sortes de fichiers spéciaux, les fichiers de mode bloc (la ligne débute par le caractère b) et les fichiers de mode caractère (la ligne débute par le caractère c). Dans le mode caractère, le processus utilisateur interagit directement avec le pilote (caractère par caractère). Dans le mode bloc, il y un intermédiaire dans le système qui gère des tampons. C'est cet intermédiaire qui prendra la décision de vider les tampons (par exemple vers le disque) ou de les remplir.

Profitez pour faire un df et de regarder le minix.config dans votre répertoire minix du Sun.

b) Faites de même sous Solaris. Aller voir /dev. En fait, dans Solaris, les fichiers spéciaux ne sont pas directement dans /dev, mais dans /devices. Vous verrez si vous cherchez bien que certains

périphériques (les disques par exemple) fournissent deux modes d'accès (mode bloc et mode caractère ou raw).

Comment créer un fichier spécial. C'est la commande mknod.

c) Tout fichier spécial possède 2 nombres, l'un dit majeur et l'autre dit mineur. Ces 2 nombres apparaissent lorsque vous faites la commande ls.

Le majeur permet de désigner le driver. Le majeur sera utilisé dans le noyau comme index (aiguillage) dans un tableau de structure de type driver, cette structure contenant des pointeurs vers les fonctions interfaces du driver (voir dans le fichier sunfloppy.c[2] l'objet f_dtab). Tous les drivers possèdent le même ensemble de fonctions. Il suffit de brancher un driver pour travailler avec lui. A vous de découvrir comment.

Un driver (de numéro majeur M) peut en fait piloter plusieurs périphériques désigner par le mineur (m). Il est passé en paramètre à certaines fonctions de l'interface du driver.

Epatez votre binôme en faisant de la magie :

- taper dans une fenêtre minix la commande tty pour avoir le device associé à la fenêtre, /dev/tty*xy*. Dans une autre fenêtre minix taper
  ls > /dev/tty*xy*
  commentez...
- Plus violent :
  cat /dev/hdx0 > /dev/kmem
  Philosophez sur le sens de la vie et les besoins de protection...

3.2. Commandes utiles de navigation

On vous rappelle l'existence des 2 commandes de recherche :

- d'un fichier

```
find . -name alloc.c -print
```

- d'une chaîne dans un ensemble de fichiers

```
grep -n sys_call *
```

3.3. Structure de smx

Smx comporte 3 parties :

- kernel au plus bas qui s'occupe entre autres de la gestion des interruptions, qui contient les pilotes de périphériques, les outils de communications inter-processus (par messages)
- et, au-dessus du kernel, fs (la gestion des fichiers)
- et mm (la gestion de la mémoire).

3.4. Organisation d'une partie (kernel, mm ou fs).

Elles sont bâties sur le même modèle. Regardez les fichiers kernel.h, fs.h, et mm.h des répertoires mm, fs et kernel de src. Ils contiennent des directives include. Vous y trouverez les include "locaux" suivants :

- const.h (les constantes de la partie)
- type.h (les types de la partie)
- glo.h (les variables globales de la partie)
- proto.h (les fonctions de l'interface de la partie)

[2]Ce fichier n'a plus rien à voir avec les disquettes car on n'est plus en 1980. Il s'agit maintenant d'un fichier contrôlant les disques en général.

Vous trouverez aussi des include "globaux" mentionnant des fichiers.h se trouvant dans MINIX/include (en particulier des types et fonctions communes aux 3 parties).

4. LE TRAVAIL QUI NOUS INTÉRESSE

Nous allons nous inspirer dans notre travail du seul vrai gestionnaire de périphérique existant *sunfloppy.c*. Donc, allez voir comment est construit un driver.

4.1 Interface du pilote avec le reste du système

La partie dépendante du périphérique se trouve dans sunfloppy.c ; la partie commune à tout driver dans driver.c.

Une tâche système associée au driver appelle la seule fonction publique du fichier *sunfloppy.c* (cette fonction s'appelle driver_task). La fonction driver_task appelle à son tour la fonction driver du fichier *driver.c* en lui passant en paramètre un objet structure contenant des pointeurs sur des fonctions privées de sunfloppy.c (les fonctions interface du driver).

```
static struct driver f_dtab = {
   f_do_name,  /* current device's name */
   f_do_open,  /* open or mount */
   f_do_close, /* nothing on a close */
   f_do_ioctl, /* ioctl */
   f_prepare,  /* prepare for I/O on a given minor device */
   f_schedule, /* do the I/O */
   f_do_finish,      /* schedule does the work, no need to be smart */
   f_do_cleanup      /* nothing's dirty */
};
```

Tout driver possède (à peu de chose près) le même ensemble de fonctions d'interface.

Si vous regarder le fichier memory.c, vous vous apercevrez que le driver qu'elle possède également une fonction memory_task qui appelle la fonction driver_task de driver.c avec également comme paramètre un objet structure contenant des pointeurs de fonctions.

4.2 Si nous avions le temps... Modifier sunprinter.c

C'est ça qu'il faudrait faire si nous avions le temps...

Maintenant, c'est à vous de travailler, vous devez faire un driver pour une imprimante (qui, je le rappelle, n'imprime pas sur une vraie imprimante, mais sur un fichier Solaris "imprimante"). Vous trouverez un fichier sunprinter.c à modifier.

Avant de commencer voyez l'effet des commandes :

- echo "toto" >/dev/lp
- lpr un_fichier_imprimable_de_smx

On vous fournit dans le répertoire test de src une copie du fichier sunread.c où sont mises en œuvre les deux commandes sunread et sunwrite (transfert de fichiers entre Solaris et smx). Vous pouvez vous inspirer, ces deux commande accèdent à des fichiers Solaris.

La fonction SunOS permet d'exécuter à partir de smx des fonctions systèmes de Solaris. Vous devez en particulier savoir ouvrir et fermer un fichier, savoir lire ou écrire dans un fichier Solaris. Sous Solaris, le manuel en ligne sur open, close, etc, ... est à votre disposition. Pour connaître la valeur de certaines drapeaux et modes d'ouverture de fichiers sous Solaris (incompatibilité Solaris et smx), cherchez sous /usr/include fichier fcntl.h.

Vous verrez que la fonction f_prepare utilise des fonctions du kernel de smx très particulières (set_protect, numap, lock et unlock) vous pouvez allez voir ce qu'elles font. Ce n'est pas très nécessaire. Vous pouvez les utiliser aveuglément.

Vous devez aussi regarder la fonction driver_task de driver.c pour voir comment et dans quelle ordre les fonctions d'interface du driver sont appelées.

Vous trouverez aussi dans le répertoire test sous src le fichier lpr.c correspondant à la commande lpr. En détective, vous pouvez partir à la piste du flot de contrôle partant d'un processus exécutant lpr et menant au driver, via la partie fs. Est-ce qu'un spooler est mis en œuvre?

TP MINIX : MINIX. Signaux UNIX. Ordonnancement

2. RAPPELS DE CONNAISSANCE

Ici, nous allons nous intéresser à l'ordonnanceur, une partie du travail du noyau ; celui-ci est géré par la cohabitation du timer (qui signale la fin du temps imparti à un processus) et par le gestionnaire des processus (qui indique le nouveau processus à exécuter).

Le but du TP est aussi de s'immerger dans un gros code écrit en C, d'apprendre à naviguer, regarder comment est faite la programmation objet en C,...

2.1. La structure de minix

Minix est une structure en 3 couches :

- au plus bas, le noyau. Les processus qui s'y trouvent sont appelés tâches.
- au niveau intermédiaire, les parties mm (memory management) et fs (file server) par exemple. Il y dans chaque partie un processus appelé serveur.
- au niveau le plus haut, il y a les processus utilisateurs.

Les processus communiquent par envoi et réception de messages.

2.2. Le travail à effectuer

L'ordonnanceure suit trois niveaux de priorité correspondant aux trois couches ; il utilise le principe de Round Robin (cf. Annexe) pour la couche utilisateur, alors que les autres couches sont exécutées jusqu'à « terminaison » (en réalité, mise en attente d'un message ou d'une interruption).

Le but de ce STP sera de rajouter une couche supérieure (de très faible priorité) pour les processus qui nécessitent beaucoup de temps CPU.

2.3. Promenade dans les sources

2.3.1. C'est quoi un signal sous UNIX?

C'est un moyen utilisé par UNIX pour informer un processus qu'un événement s'est produit.

Il y a deux sortes d'événements :

- les erreurs (division par zéro, tentative non autorisée d'exécuter une instruction machine, tentative non autorisée d'accéder à une certaine zone de mémoire).
- les événements asynchrones (par exemple, l'alarme déclenchée à chaque tic de l'horloge).

Regardez le fichier minix/include/sun/signal.h. Vous trouverez la liste des signaux que minix peut recevoir en provenance de Solaris. Ce n'est pas obligatoirement la même liste que celle des signaux propres au système smx (voir fichier minix/include/signal.h).

Chaque processus possède un masque de signaux indiquant quels sont les signaux qu'il masque, et ceux qu'il peut traiter par un handler spécifique.

Pour rappel, côté Solaris, il y a deux (et seulement deux processus) qui s'occupent du système minix, l'un étant le père de l'autre. C'est le processus fils qui exécute smx. Le père, après avoir masqué tous les signaux, sauf ceux d'erreur, attend simplement que son fils se termine (il fera alors du nettoyage). Les élèves vraiment curieux peuvent regarder le fichier minix/src/Solaris/minix.c pour voir ce que fait Solaris pour lancer Minix.

2.3.2. Associer un traitement à chaque signal.

Regardez maintenant le fichier minix/src/kernel/mpx.c. La fonction init_sun_handlers, appelée au boot de minix, fait en sorte que le processus minix exécute la fonction SunOSsig de mpx.c à chaque prise en compte d'un signal. En fait aucun signal n'est masqué.

La fonction init_sun_handlers, à l'aide de l'appel système Solaris *sigaction*, associe à chaque signal le même handler.

Voici la signature de sigaction :

int sigaction (int sigid, const struct sigaction *act, struct sigaction *oact);

La fonction sigaction a pour effet d'associer un nouveau traitement au signal de numéro sigid et de sauvegarder l'ancien dans oact. En fait, la structure sigaction spécifie non seulement le handler du signal (sa_handler), mais aussi des drapeaux et les signaux qui restent masqués pendant l'exécution du handler (en effet, il y a une sorte d'exclusion mutuelle à assurer).

Il y a apparemment un seul handler pour tous les signaux (SunOSsig). Regardez cette fonction. Il n'y a qu'un seul handler, car il y a des traitements communs à chaque signal.

Que font les appels à entering_kernel et memcpy dans le prélude de sunOSsig, ainsi que les appels à lock_pick_proc et resume dans le postlude?

En fait un aiguilage interne est effectué par sunOSsig vers le handler adéquat (voir l'initialisation de vectors dans init_sun_handlers). Il y a 4 sous-handlers :

- exceptions pour tous les signaux, sauf 3
- s_call pour SIGUSR1
- smx_clock_handler pour SIGALARM
- sunio_interrupt pour SIGIO

Regardez les 2 plus intéressants : smx_clock_handlers et s_call. Que font-ils?

2.3.3. L'horloge.

Sur la piste des fonctions clock_task et clock_handler du fichier minix/src/kernel/clock.c, et de la fonction interrupt de minix/src/kernel/proc.c.

Qui fait quoi?

2.3.4. L'ordonnanceur (scheduler)

Il y a plein de choses intéressantes à découvrir dans ce fichier minix/src/kernel/proc.c. N'est-ce pas là qu'il faut intervenir pour rajouter une seconde classe de processus utilisateurs?

Le scheduler sert à gérer le multitâche ; c'est lui qui simule l'exécution en simultané de toutes les requêtes (système et utilisateur) en répartissant le travail de l'unité centrale entre les processus de smx. Il est confronté à plusieurs problèmes :

- temps de réponse minimal pour un utilisateur (quelqu'un qui demande quelque chose de bénin veut avoir une réponse quasi-immédiate).
- répartition équitable de son temps entre les processus / utilisateurs (un processus ne doit pas être laissé de côté pendant trop longtemps).
- utilisation maximale de l'unité centrale (pas de temps d'inactivité).
- efficacité maximale (maximum de choses faites par unité de temps).

Certaines de ces contraintes étant contradictoires, il n'existe pas de solution unique mais des solutions plus appropriées à certaines applications et à certaines machines. Le choix fait pour minix tient à sa simplicité et à des caractéristiques correctes dans tous les domaines.

3. PASSONS AUX CHOSES SÉRIEUSES

3.1. Repérage sur le terrain

Pour commencer, vous avez dans le répertoire *kernel*, les données sur les listes d'attente (*const.h*), la fréquence du scheduler (`Millisec` et `Sched_Rate`), ainsi que les méthodes de choix du processus suivant.

L'intervention sur la fréquence du scheduler (attention à rester au-dessus de certaines valeurs pour garder la présence du multitâche, cf. Annexe) permet de changer le temps imparti à chaque processus et donc le nombre de fois qu'il sera interrompu; vous pouvez essayer d'utiliser le programme `sched_test1` (sources dans le répertoire *test*) avec les valeurs 10 et 10,000 (dure environ 20-30 secondes), puis d'en mettre 2 en parallèle, puis de refaire la même chose en mettant 20,000ms imparties à chaque processus (afin de voir l'évolution du degré de parallélisme).

3.2. Au boulot

Voilà, vous avez les bases sur le temps partagé, une idée de ce qu'il se passe sous minix, il ne vous reste plus qu'à modifier tout ça pour obtenir une politique d'ordonnancement un peu plus évoluée.

Donc, vous devez rajouter une quatrième couche de priorité plus faible avec des temps d'exécution plus longs (on utilisera aussi la méthode Round Robin pour cette couche). Cette couche pourra être atteinte (et quittée) selon une politique que vous pouvez choisir (une proposition est faite, mais vous pouvez la modifier si cela vous paraît plus intéressant ou plus approprié).

3.3. Solution :

Les processus qui utilisent entièrement leur temps imparti verront leur priorité baisser ; ceux de faible priorité verront leur priorité augmenter quand ils ne termineront pas leur temps. Le temps de la couche supérieure peut être 10 fois plus grand par exemple.

4. UN DERNIER PETIT EXERCICE

Donc, pour finir, et pour bien réaliser la difficulté de réaliser un système « parfait », nous allons retravailler sur le scheduler. En effet, ce que vous avez du réaliser la fois précédente pose un problème ; un processus peut se retrouver bloqué au niveau de priorité le plus faible sans être jamais exécuté. Pour cela, il suffit qu'il y ait assez de processus au niveau de priorité supérieur, le scheduler étant toujours pris à 100%.

Pour éviter ce genre de choses, on peut rajouter une règle : si un processus reste trop longtemps au niveau de priorité le plus faible sans être jamais exécuté, il passe au niveau plus prioritaire (il n'est pas très bien traité, mais au moins il progresse). Essayez de rajouter cette caractéristique au scheduler.

5. POUR ALLER PLUS LOIN

Il reste de nombreux points sur les systèmes d'exploitation qui n'ont pas été abordés pendant ces STPs, et vous pouvez en aborder certains en étudiant smx. Il y a ci-dessous quelques idées de sujets qui peuvent être intéressants du point de vue de minix.

5.1. Memory Manager

La façon d'allouer la mémoire a été rapidement traitée ; vous pouvez cependant étudier d'autres choses sur l'allocation de la mémoire, que ce soit assez simple (telles que le 'garbage collecting' ou l'allocation par paquets de taille déterminée) ou plus compliqué et demandant plus de travail (notamment le swap).

5.2. Kernel

Nous avons vu quelques éléments de celui-ci, mais les sujets sont très vastes à son sujet ; que ce soit des drivers (rapidement étudiés), du scheduler (il existe de nombreuses autres méthodes ayant chacune ses intérêts et ses applications), mais aussi de la gestion des messages, du parallélisme (et donc de l'interblocage), etc.

5.3. File system

Cette partie n'a pas été abordée (par manque de temps), mais vous pouvez également y regarder (et modifier) la forme même du système de fichiers (notamment table des inodes), les méthodes de gestion des fichiers ouverts, de lecture/écriture (et donc les différents niveaux de cache)...

5.4. Le réseau

Cette partie a été enlevée des sources (pour en diminuer la taille et la complexité), mais vous pouvez tout de même l'étudier, un fichier contenant les sources complètes étant présent.

6. CONCLUSION

Si vous êtes intéressés par ce système, il existe un newsgroup et une mailing-list sur minix (également une sur smx) ; ce système peut s'installer sur tout PC (à partir du 8086), sur Macintosh, sur Atari et sur Amiga (ces deux derniers ne sont peut-être pas trop à jour),...

Annexe 1

Limitation de la fréquence du scheduler

Le scheduler change de processus tous les m temps; ce changement de contexte met un temps n. Il est donc naturel de ne pas trop baisser la valeur de m pour ne pas perdre trop de temps CPU à ces changements de contexte qui ne font pas avancer les requêtes des utilisateurs. Si m est peu différent de n, on aura une efficacité de 0,5 (très moyenne) mais si m>>n, on aura un temps d'attente inacceptable pour l'utilisateur.

Pour ce qui concerne smx (minix sous Solaris), la fréquence de l'horloge simulée a été baissée pour que le travail du système d'exploitation à chaque tick horloge ne nuise pas trop à l'efficacité de celui-ci. Elle est donc fixée à 20Hz (pour la changer, il faudrait recompiler les librairies), ce qui vous interdit de faire descendre la valeur de `Millisec` en dessous de 50 (sinon, le système ne réalise plus de commutation et les processus sont lancés jusqu'à ce qu'ils se terminent).

Annexe 2

Les différentes politiques d'agencement

La méthode la plus simple est celle de Round Robin, elle consiste à donner un temps n à chaque processus à tour de rôle (ils sont placés sur une liste d'attente afin de pouvoir être exécutés quand leur tour vient); il faut préciser que les processus sont retirés de la liste d'attente quand ils se terminent, mais aussi quand ils se mettent en situation d'attente d'un message (que ce soit de la part d'un autre processus ou de la part du système pour une lecture du disque dur par exemple).

De nombreuses autres théories existent, qu'elles soient basées sur les avantages fournis pas certains systèmes prévus à cet effet, ou simplement car les utilisateurs ne veulent pas tous être considérés de la même façon (le super-utilisateur doit-il avoir une priorité supérieure aux autres?). Il existe, pour donner quelques exemples rapidement, des techniques à plusieurs couches (certaines utilisant le swap de la mémoire), à diverses priorités (priorité pouvant être fluctuante en fonction du temps) ou exécutant le plus court en premier (mais comment le savoir avant?). Le plus souvent, les solutions choisies utilisent un compromis de plusieurs principes en fonction du contexte d'utilisation.