Emulateur Logiciel MPC/1
Manuel d'Installation

version 3.0

Université Paris 6 - MASI
Alexandre Fenyö, Frédéric Potter, Pierre David

18 décembre 1996

Table des matières

Table des figures

Chapitre 1

Introduction

Ce manuel décrit les procédures de mise en place de l'émulateur de la machine parallèle MPC.

1.1 Architecture logicielle

L'architecture logicielle des services de communication de la machine MPC est composée de modules noyau chargeables dynamiquement[1], de démons Unix, ainsi que d'une bibliothèque de programmation.

Les différences de conception entre la machine MPC et sa version émulateur sont les suivantes :

– Les démons Unix nécessaires à la configuration de la machine possèdent en plus une fonction de routage : sur chaque noeud, ils extraient les entrées de LPE (Liste des Pages à Emettre) et effectuent les échanges de mémoire physique requis.

– La couche PUT émule la gestion d'une LPE et coopère avec les démons Unix pour la simulation des interruptions en émission comme en réception.

– La couche de plus bas niveau (pilote HSL) coopère avec les démons Unix pour les échanges de mémoire physique à mémoire physique.

– La couche USERACCESS communique avec la couche PUT dans le cadre de l'émulateur, il y a donc passage en mode noyau, alors que dans la machine MPC, cette couche communique directement avec le composant PCI-DDC sans passage en mode noyau. Il n'y a, par contre, pas de différence pour l'utilisateur.

1. loadable kernel module

La figure 1.1 présente les différentes couches logicielles en jeu dans l'émulateur MPC.

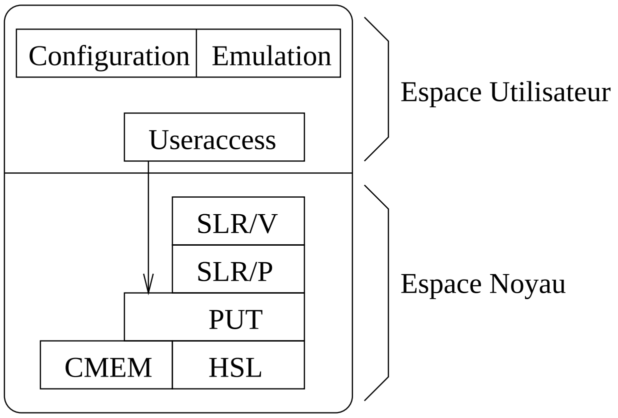

FIG. 1.1 - *Architecture logicielle*

Chapitre 2

L'émulateur de la machine MPC émule les fonctionnalités du circuit PCI-DDC. Seuls les messages courts et les messages composés ne sont pas supportés. Les couches HSLDRIVER et PUT sont les seuls éléments qui vont être réimplémentés pour dialoguer avec la carte d'interface Fast-HSL, les autres couches sont indentiques qu'il s'agisse de fonctionner sur l'émulateur ou avec la carte Fast-HSL.

La figure 2.1 présente les relations entre les différents modules et démons dans le cadre de l'utilisation de la machine MPC et la figure 2.2 présente les relations entre ces mêmes modules dans le cadre de l'utilisation de l'émulateur.

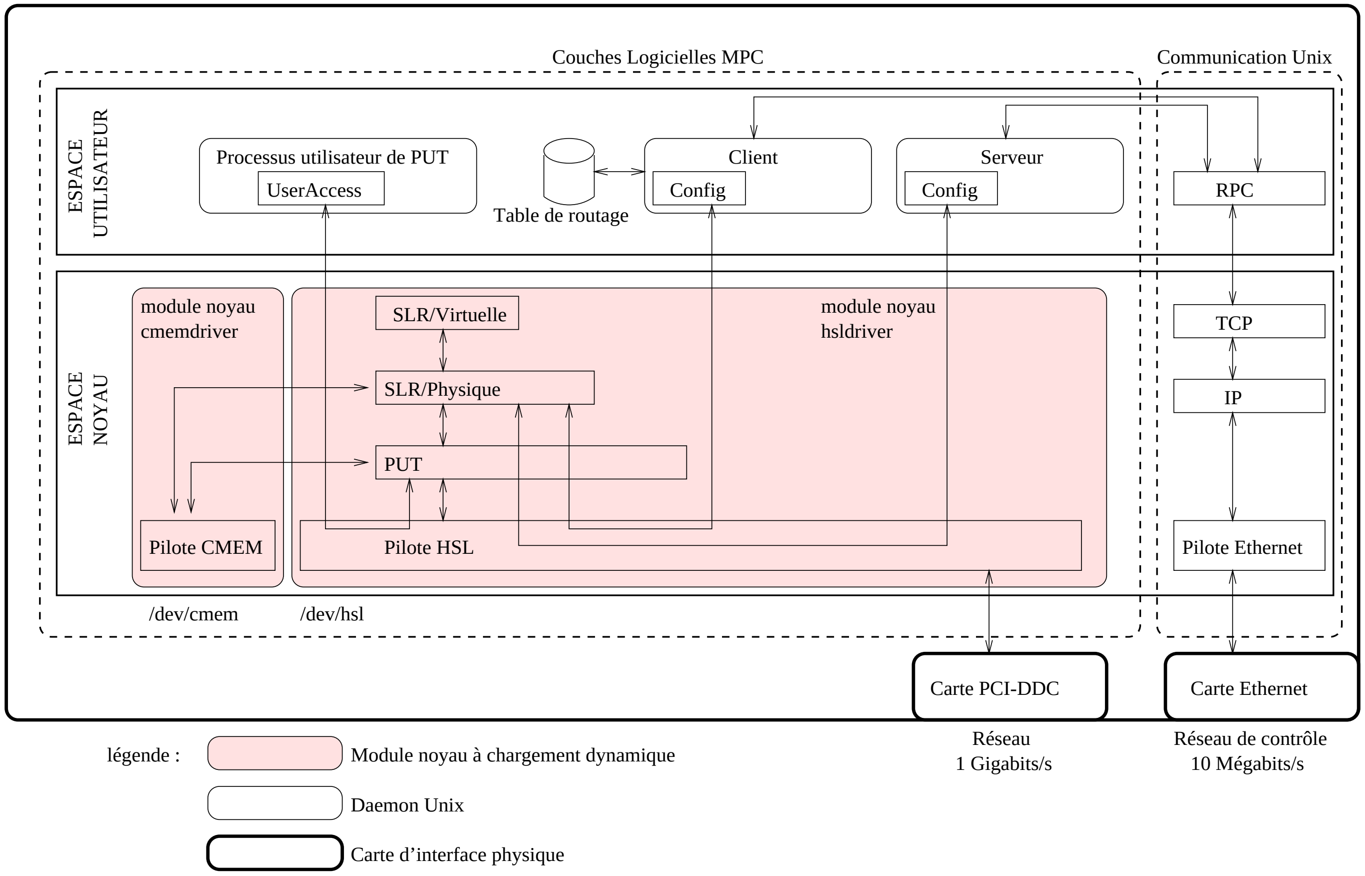

FIG. 2.1 - *Interactions entre modules sur la machine MPC*

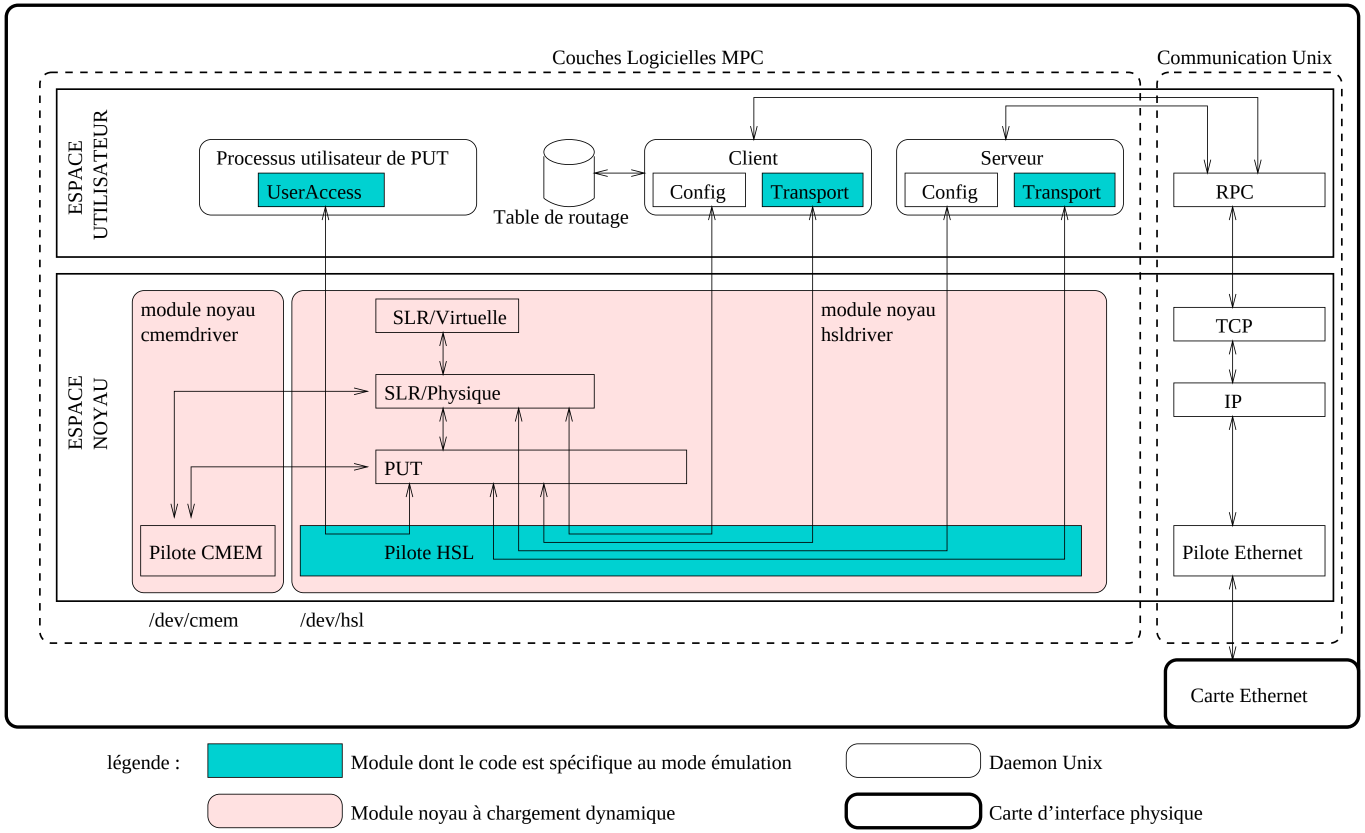

Fig. 2.2 - *Interactions entre modules sur l'émulateur*

Chapitre 3

Installation

Ce chapitre décrit la procédure d'installation des modules, démons et de la bibliothèque dont la distribution est composée. Pour installer correctement ces différents éléments, il suffit de suivre les opérations pas-à-pas, et dans l'ordre adopté par le chapitre.

3.1 Prérequis

La distribution version 3.0 des services logiciels de communication MPC mode émulateur nécessite le système d'exploitation FreeBSD-2.1. De plus, l'utilitaire `gmake`[1] est requis si l'utilisateur se procure la version source. Enfin, si l'utilisateur désire apporter des modifications, il lui faut posséder les utilitaires GNU-Automake[2] et GNU-Autoconf[3].

3.2 Obtention de la distribution

Pour obtenir la distribution logicielle, **il faut faire une demande par *email* auprès de mpc@mpc.ibp.fr**.

Un identifiant et un mot-de-passe vous seront alors fournis, afin que vous puissiez récupérer l'ensemble des sources sur le réseau Internet, par différents protocoles :

- FTP anonyme :
  - serveur : `mpc.ibp.fr`

1. GNU-Make : disponible sur ftp://ftp.ibp.fr/gnu/make-3.74.tar.gz
2. GNU-Automake : disponible sur ftp://ftp.ibp.fr/gnu/automake-1.0.tar.gz
3. GNU-Autoconf : disponible sur ftp://ftp.ipb.fr/gnu/autoconf-2.10.tar.gz

  – identification : contacter mpc@mpc.ibp.fr
  – mot-de-passe : contacter mpc@mpc.ibp.fr
  – répertoire : `/pub/emulateur`
  – fichier : `emulateur-3.0.tar.gz` (distribution source)
  – fichier : `emulateur-3.0.binaires.tar.gz` (distribution précompilée)

– FTAM[4] :

  – serveur : `mpc.ibp.fr`
  – identification : contacter mpc@mpc.ibp.fr
  – mot-de-passe : contacter mpc@mpc.ibp.fr
  – répertoire : `/pub/emulateur`
  – fichier : `emulateur-3.0.tar.gz` (distribution source)
  – fichier : `emulateur-3.0.binaires.tar.gz` (distribution précompilée)
  – type d'encodage : `set realstore unix`

De nombreux documents techniques sur le projet MPC sont accessibles par le World Wide Web, à l'URL `http://mpc.ibp.fr`.

3.3 Compilation de la distribution

Après avoir créé un compte de login `mpc`, il faut décompresser et désarchiver la distribution :

```
% gzip -dc emulateur-3.0.tar.gz | ( cd ~mpc/; tar xf - )
% cd ~mpc/mpc-3.0/
%
```

Les fonctions des différents répertoires et fichiers de la distribution sont indiquées sur la figure 3.1.

La distribution se conforme au standard de portabilité GNU : elle est construite à partir d'Autoconf et d'Automake.

4. File Tranfer Access Managment, transfert de fichiers normalisé ISO, couche transport TSAP sur TCP - RFC-1006

répertoire	fichier	fonction
mpc-3.0/ (racine de la distribution)	Makefile.am	Makefile format GNU Automake
	Makefile.in	template de Makefile pour GNU Autoconf
	configure	Script d'autoconfiguration et de génération des Makefiles
	configure.in	Template de configure pour Autoconf
	install-sh	Script nécessaire à l'installation
	mkinstalldirs	Script nécessaire à l'installation
	ChangeLog	Liste chronologique des modifications
	INSTALL	Manuel d'utilisation de configure
	hsl.conf	Exemple de fichier de configuration
mpc-3.0/docs/ (documentations sur l'interface logicielle)	Makefile.am	Makefile format GNU Automake
	Makefile.in	template de Makefile pour GNU Autoconf
	pcihsl.texi	source Texinfo de la documentation technique
	texinfo.tex	macros Texinfo
	specif.c	morceau de la documentation
	*.{tki,fig,eps}	morceaux de la documentation
	*.tex	documentations utilisateurs
mpc-3.0/modules/ (sources des modules CMEM et HSLDRIVER)	Makefile.am	Makefile format GNU Automake
	Makefile.in	template de Makefile pour GNU Autoconf
	data.h	définitions générales
	ddslrpv.{c,h}	interface SLR/V
	ddslrpp.{c,h}	interface SLR/P
	put.{c,h}	interface PUT
	driver.{c,h}	Pilote HSL
	cmem.{c,h}	Pilote CMEM
mpc-3.0/src/ (sources des démons de configuration et d'émulation, et bibliothèque Useraccess)	Makefile.am	Makefile format GNU Automake
	Makefile.in	template de Makefile pour GNU Autoconf
	hslclient.{c,h}	Client RPC
	hslserver.{c,h}	Serveur RPC
	useraccess.{c,h}	bibliothèque d'accès à PUT en mode utilisateur
	rpctransport.x	structures de données RPC pour rpcgen
	virt2phys.c	utilitaire de déréférenciation physique
	readphys.c	utilitaire de lecture physique
	writephys.c	utilitaire d'écriture physique
mpc-3.0/examples/	addlpe.c	exemple d'utilisation de PUT
	receivelpe.c	
	testrecv2.c	exemple d'utilisation de SLR/P
	testsend2.c	
	testrecvv.c	exemple d'utilisation de SLR/V
	testsendv.c	
mpc-3.0/archive/modload/	sources de modload modifiées	

FIG. 3.1 - *Contenu de la distribution logicielle source*

Il faut donc lancer le script `configure` qui va générer les différents Makefiles en fonction de l'environnement du système.

```
% ./configure
creating cache ./config.cache
checking whether make sets ${MAKE}... yes
checking for gcc... gcc
checking whether we are using GNU C... yes
checking whether gcc accepts -g... yes
checking how to run the C preprocessor... gcc -E
checking whether gcc needs -traditional... no
checking for a BSD compatible install... /usr/local/bin/install -c
checking whether ln -s works... yes
checking for ranlib... ranlib
checking for cat... /bin/cat
checking for ar... /usr/bin/ar
checking for makedepend... /usr/X11R6/bin/makedepend
checking for chmod... /bin/chmod
checking for modload... /sbin/modload
checking for modunload... /sbin/modunload
checking for modstat... /usr/bin/modstat
checking for mknod... /sbin/mknod
checking for rm... /bin/rm
checking for ls... /bin/ls
checking for id... /usr/bin/id
checking for test... /bin/test
checking for cut... /usr/bin/cut
checking for sed... /usr/bin/sed
checking for grep... /usr/bin/grep
checking for rpcgen... /usr/bin/rpcgen
checking whether cross-compiling... no
checking for ANSI C header files... yes
updating cache ./config.cache
creating ./config.status
creating Makefile
creating src/Makefile
creating modules/Makefile
creating examples/Makefile
creating docs/Makefile
%
```

Il faut maintenant compiler les différentes sources :

```
% gmake
for subdir in src modules examples docs; do             \
  target=`echo all-recursive | sed s/-recursive//`; \
  echo making $target in $subdir;       \
  (cd $subdir && gmake $target) \
   || case "" in *k*) fail=yes;; *) exit 1;; esac; \
done && test -z "$fail"
making all in src
gmake[1]: Entering directory `/tmp/mpc-3.0-Alpha/src'
gcc -c -DSTDC_HEADERS=1  -I. -I.    -g -O useraccess.c
rm -f libuseraccess.a
ar cru libuseraccess.a useraccess.o
ranlib libuseraccess.a
/usr/bin/rpcgen -c rpctransport.x > rpctransport_xdr.h
/usr/bin/rpcgen rpctransport.x
gcc -c -DSTDC_HEADERS=1  -I. -I.    -g -O hslclient.c
gcc -c -DSTDC_HEADERS=1  -I. -I.    -g -O rpctransport_clnt.c
gcc  -o hslclient hslclient.o rpctransport_clnt.o
gcc -c -DSTDC_HEADERS=1  -I. -I.    -g -O hslserver.c
gcc -c -DSTDC_HEADERS=1  -I. -I.    -g -O rpctransport_svc.c
gcc  -o hslserver hslserver.o rpctransport_svc.o
gcc -c -DSTDC_HEADERS=1  -I. -I.    -g -O malloc.c
gcc  -o malloc malloc.o
gcc -c -DSTDC_HEADERS=1  -I. -I.    -g -O virt2phys.c
gcc  -o virt2phys virt2phys.o
gcc -c -DSTDC_HEADERS=1  -I. -I.    -g -O readphys.c
gcc  -o readphys readphys.o
gcc -c -DSTDC_HEADERS=1  -I. -I.    -g -O writephys.c
gcc  -o writephys writephys.o
gmake[1]: Leaving directory `/tmp/mpc-3.0-Alpha/src'
making all in modules
gmake[1]: Entering directory `/tmp/mpc-3.0-Alpha/modules'
gcc -c -DSTDC_HEADERS=1 -I. -DKERNEL -Wall -DDEBUG_HSL -DMODNAME=\"hsldriver\" -O driver.c
gcc -c -DSTDC_HEADERS=1 -I. -DKERNEL -Wall -DDEBUG_HSL -DMODNAME=\"hsldriver\" -O put.c
gcc -c -DSTDC_HEADERS=1 -I. -DKERNEL -Wall -DDEBUG_HSL -DMODNAME=\"hsldriver\" -O ddslrpp.c
gcc -c -DSTDC_HEADERS=1 -I. -DKERNEL -Wall -DDEBUG_HSL -DMODNAME=\"hsldriver\" -O ddslrpv.c
ld -r -o hsldriver driver.o put.o ddslrpp.o ddslrpv.o
gmake[1]: Leaving directory `/tmp/mpc-3.0-Alpha/modules'
making all in examples
gmake[1]: Entering directory `/tmp/mpc-3.0-Alpha/examples'
gcc -c -DSTDC_HEADERS=1  -I. -I.    -g -O addlpe.c
gcc  -o addlpe addlpe.o
gcc -c -DSTDC_HEADERS=1  -I. -I.    -g -O receivelpe.c
gcc  -o receivelpe receivelpe.o
gcc -c -DSTDC_HEADERS=1  -I. -I.    -g -O testsend2.c
gcc  -o testsend2 testsend2.o
gcc -c -DSTDC_HEADERS=1  -I. -I.    -g -O testrecv2.c
gcc  -o testrecv2 testrecv2.o
gcc -c -DSTDC_HEADERS=1  -I. -I.    -g -O testsendv.c
gcc  -o testsendv testsendv.o
gcc -c -DSTDC_HEADERS=1  -I. -I.    -g -O testrecvv.c
gcc  -o testrecvv testrecvv.o
gcc  -o testuseraccess testuseraccess.o ../src/libuseraccess.a
gmake[1]: Leaving directory `/tmp/mpc-3.0-Alpha/examples'
making all in docs
gmake[1]: Entering directory `/tmp/mpc-3.0-Alpha/docs'
/bin/cat specif.c | /usr/bin/sed 's/\([{}]\)/@\1/g' > specif.c.texi
makeinfo -I. pcihsl.texi -o ./pcihsl.info
Making info file `./pcihsl.info' from `pcihsl.texi'.
gmake[1]: Leaving directory `/tmp/mpc-3.0-Alpha/docs'
%
```

Nous devons compiler `modload`, dont les sources se trouvent dans le répertoire `~mpc/mpc-3.0/archive/modload`, car la version standard de FreeBSD ne suffit pas à l'utilisation qu'on veut en faire. En effet, le module CMEMDRIVER, gestionnaire de mémoire physique, a besoin de réserver de la RAM au moment de l'amorçage d'Unix, alors que le répertoire `/usr` n'est pas encore monté. Cela pose un problème à l'utilisation du `modload` livré dans la distribution de FreeBSD : ce dernier utilise `/usr/bin/ld`, qui n'est pas disponible à ce moment du boot. Le `modload` de la distribution MPC appelle quant-à-lui `/etc/mpc/ld`.

```
% cd archive/modload/
% make LDFLAGS=-static
cc -O   -c modload.c
cc -O   -static -o modload modload.o
%
```

3.4 Installation des démons Unix et de la bibliothèque USERACCESS

La cible `install` des Makefiles générés précédemment permet d'installer les démons Unix, ainsi que la bibliothèque USERACCESS.

Il faut donc entrer la commande `gmake install` :

```
% su
# cd ~mpc/mpc-3.0/
# gmake install
...
# ls -lt /usr/local/bin/ | head -6
total 91204
-rwxr-xr-x  1 root  bin      42283 Jul 19 00:30 hslclient*
-rwxr-xr-x  1 root  bin      39577 Jul 19 00:30 hslserver*
-rwxr-xr-x  1 root  bin      19539 Jul 19 00:30 writephys*
-rwxr-xr-x  1 root  bin      19525 Jul 19 00:30 readphys*
-rwxr-xr-x  1 root  bin      19593 Jul 19 00:30 virt2phys*
# ls -l /usr/local/lib/libuseraccess.a
-rw-r--r--  1 root  bin      12916 Jul 19 00:30 /usr/local/lib/libuseraccess.a
# ls -lt /usr/local/include/ | head -7
total 452
-rw-r--r--  1 root  bin        376 Jul 19 00:30 useraccess.h
-rw-r--r--  1 root  bin       2761 Jul 19 00:30 put.h
-rw-r--r--  1 root  bin       2419 Jul 19 00:30 driver.h
-rw-r--r--  1 root  bin       1163 Jul 19 00:30 ddslrpv.h
-rw-r--r--  1 root  bin       1030 Jul 19 00:30 ddslrpp.h
-rw-r--r--  1 root  bin       5500 Jul 19 00:30 data.h
# ls -l /usr/local/lib/mpc/
total 376
-rwxr-xr-x  1 root  bin      36621 Jul 19 00:30 cmemdriver*
-rwxr-xr-x  1 root  bin     143333 Jul 19 00:30 hsldriver*
# ^D
%
```

3.5 Installation du module CMEMDRIVER

Le module CMEMDRIVER doit être chargé dynamiquement dans le noyau au début de l'amorçage de la machine, pour qu'il puisse trouver suffisamment de mémoire physique contigüe libre pour couvrir ses besoins. Notons qu'avec l'installation telle que nous la proposons (au début du fichier de commande `/etc/rc`), il est capable d'allouer jusqu'à 25 Mo contigüs sur une machine disposant de 32 Mo de RAM, et jusqu'à 50 Mo contigüs sur une machine disposant de 64 Mo de RAM.

Pour installer le module CMEMDRIVER, il faut créer un répertoire `/etc/mpc` qui contienne `ld`, `modload`, et le module noyau à chargement dynamique `cmemdriver.o` :

```
% su
# cd ~mpc/mpc-3.0/
# mkdir /etc/mpc
# cp archive/modload/modload /etc/mpc/
# cp modules/cmemdriver /etc/mpc/cmemdriver.o
# chmod 755 /etc/mpc/modload
# chmod 644 /etc/mpc/cmemdriver.o
# chown bin.bin /etc/mpc/modload
# chown root.kmem /etc/mpc/cmemdriver.o
# ^D
%
```

Il faut maintenant éditer le fichier de commandes au démarrage : `/etc/rc`. Juste après la commande de montage read-write du répertoire `/`, il faut rajouter la ligne :

```
/etc/mpc/modload -e cmeminit -o /etc/mpc/cmemdriver.sym \
                                /etc/mpc/cmemdriver.o
```

Le fichier `/etc/rc` ressemble donc à :

```
...
mount -u -o rw /
if [ $? != 0 ]; then
        echo "Filesystem mount failed, startup aborted"
        exit 1
fi

/etc/mpc/modload -e cmeminit -o /etc/mpc/cmemdriver.sym /etc/mpc/cmemdriver.o

umount -a >/dev/null 2>&1
...
```

Redémarrez maintenant la machine.

Pour créer le pseudo-périphérique en mode caractère `/dev/cmem`, il faut regarder le numéro de `major` associé au module CMEMDRIVER, puis utiliser la commande `mknod`. Le `minor` doit être nul.

```
% su
# modstat
Type     Id Off Loadaddr Size Info     Rev Module Name
DEV       0  33 f0aa6000 0013 f0aa702c   1 cmemdriver
# mknod /dev/cmem c 33 0
# chmod 644 /dev/cmem
# chown root.kmem /dev/cmem
# ^D
%
```

Pour vérifier que le module a correctement réservé un espace de mémoire contigüe[5], il faut lire le contenu de `/dev/cmem` :

```
% cat /dev/cmem
/dev/cmem : low level driver for contig. space

physical start :   0xac3000
virtual  start : 0xf4b2e000
length         :    1048576

+------+------------+------------+----------+----------+
| slot | phys start | virt start |     size |     name |
+------+------------+------------+----------+----------+
|    0 |        0x0 |        0x0 |        0 |          |
+------+------------+------------+----------+----------+
|    1 |        0x0 |        0x0 |        0 |          |
+------+------------+------------+----------+----------+
|    2 |        0x0 |        0x0 |        0 |          |
+------+------------+------------+----------+----------+
|    3 |        0x0 |        0x0 |        0 |          |
+------+------------+------------+----------+----------+
%
```

On constate que le module a été capable d'allouer correctement 1 Mo de RAM contigüe. L'installation du gestionnaire de mémoire contigüe est donc maintenant terminée.

3.6 installation du module HSLDRIVER

Le module HSLDRIVER contient le code des couches suivantes :

- SLR/V
- SLR/P
- PUT
- HSLDRIVER (pseudo-périphérique en mode caractère)

Il faut tout d'abord le copier dans le répertoire adéquat :

5. Si l'on n'a pas modifié de paramètre dans les fichiers d'entête avant de compiler, le module réserve 1 Mo par défaut.

```
% su
# cd ~mpc/mpc-3.0/
# cp modules/hsldriver /etc/mpc/hsldriver.o
# chmod 644 /etc/mpc/hsldriver.o
# chown root.kmem /etc/mpc/hsldriver.o
# ^D
%
```

Pour l'installer à la fin de la procédure d'amorçage du système, il faut rajouter, à la fin du fichier `/etc/rc.local`, la ligne suivante :

```
/sbin/modload -A /etc/mpc/cmemdriver.sym \
   -o /etc/mpc/hsldriver.sym /etc/mpc/hsldriver.o
```

On remarquera qu'au lieu d'utiliser directement les symboles du noyau pour charger notre module, on utilise le fichier `/etc/mpc/cmemdriver.sym` crée par `modload` au chargement du module CMEMDRIVER, au début de l'execution de `/etc/rc`, et qui rassemble à la fois les symboles du noyau et ceux de CMEMDRIVER. En effet, HSLDRIVER fait appel à des procédures de CMEMDRIVER, et une édition de liens avec uniquement les symboles du noyau serait insuffisante.

Redémarrez maintenant la machine.

Pour s'assurer que les deux modules CMEMDRIVER et HSLDRIVER se sont correctement chargés, on utilise `modstat` et on vérifie que la couche PUT a réservé auprès de CMEMDRIVER un espace contigu, destiné à la bibliothèque USERACCESS.

```
% modstat
Type     Id Off Loadaddr Size Info     Rev Module Name
DEV       0  33 f0aa6000 0013 f0aa702c   1 cmemdriver
DEV       1  34 f0b55000 002d f0b5a02c   1 hsldriver
% cat /dev/cmem
/dev/cmem : low level driver for contig. space

physical start :   0xac3000
virtual  start : 0xf4b2e000
length         :    1048576

+------+------------+------------+----------+----------+
| slot | phys start | virt start |     size |     name |
+------+------------+------------+----------+----------+
|    0 |   0xac3000 | 0xf4b2e000 |    16384 |      PUT |
+------+------------+------------+----------+----------+
|    1 |        0x0 |        0x0 |        0 |          |
+------+------------+------------+----------+----------+
|    2 |        0x0 |        0x0 |        0 |          |
+------+------------+------------+----------+----------+
|    3 |        0x0 |        0x0 |        0 |          |
+------+------------+------------+----------+----------+
%
```

Nous devons enfin utiliser la commande `mknod` pour créer le pseudo-périphérique en mode caractère `/dev/hsl` ; le `major` est indiqué par `modstat`, le `minor` indique l'index de la carte réseau HSL que l'on veut adresser par ce fichier spécial. Dans le cadre de l'émulateur, il doit être nul.

```
% su
# modstat -n hsldriver
Type     Id Off Loadaddr Size Info     Rev Module Name
DEV       1  34 f0b55000 002d f0b5a02c   1 hsldriver
# mknod /dev/hsl c 34 0
# chmod 644 /dev/hsl
# chown root.operator /dev/hsl
# ^D
%
```

Le module HSLDRIVER est maintenant installé.

3.7 Apport de modifications

Si des modifications sont apportées à cette distribution, et notemment aux templates des Makefiles (`Makefile.am`), il faut alors supprimer le fichier de cache de `configure` (`config.cache`), regénérer les `Makefile.in` à l'aide d'Automake[6], puis générer le script `configure` à l'aide d'Autoconf[7], et enfin générer les Makefiles à partir de `configure`. La figure 3.2 décrit les flux de données ainsi générés.

Il faut donc entrer les commandes suivantes :

```
% cd ~mpc/mpc-3.0/
% rm config.cache
% automake
% autoconf
% ./configure
...
%
```

6. Automake s'appuie sur `configure.in` et les `Makefile.am` pour produire les `Makefile.in`.

7. Autoconf s'appuie sur `configure.in` et les `Makefile.in` pour produire les Makefiles.

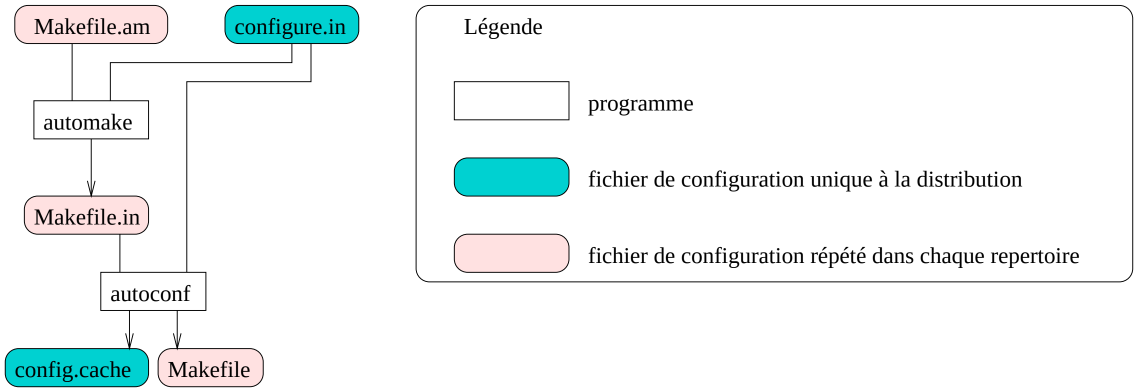

FIG. 3.2 - *Flux de données lors de l'autoconfiguration*

Chapitre 4

Configuration de la machine MPC/1

Ce chapitre décrit le format des fichiers de configuration.

4.1 Les démons de configuration

Deux démons sont nécessaires pour configurer la machine MPC. Ce sont les exécutables `hslclient` et `hslserver`, et ils ont été installés au cours du chapitre précédent dans le répertoire `/usr/local/bin/`.

Ces deux démons sont construits au-dessus de la couche RPC (Remote Procedure Call), et dialoguent entre eux par le réseau de contrôle. Chaque client consulte un fichier de configuration local au noeud, indiquant le nom de domaine de chacun des autres noeuds, et contacte les serveurs qui y résident. On doit donc activer sur chaque noeud les deux démons, dans un ordre quelconque.

La figure 4.1 décrit les liaisons RPC établies du point de vue d'un noeud, sur une machine MPC constituée de quatre noeuds.

4.1.1 Fonctionnement du démon `hslclient`

Le démon `hslclient` procède en quatre phases successives :

1. Phase de connexion : le démon `hslclient` tente de se connecter à tous les serveurs mentionnés dans le fichier de configuration ; il essaie jusqu'au succès, et quitte donc cette phase une fois que tous les démons `hslserver` ont été activés.

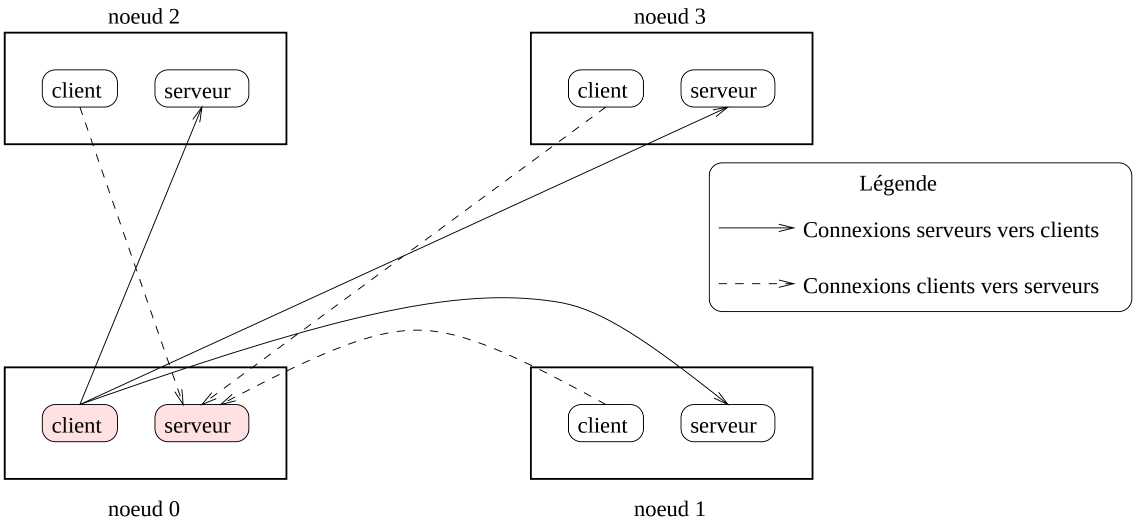

FIG. 4.1 - *Connexions RPC mettant en jeu le noeud 0*

2. Phase d'initialisation : le démon accède au noyau pour initialiser successivement les couches basses : PUT, SLR/P et SLR/V.
3. Phase de configuration : le démon interroge la couche SLR/P locale afin de connaître la plage physique contigüe allouée pour recevoir des informations des autres noeuds, et en informe les différents serveurs.
4. Phase d'émulation (en boucle) : le démon extrait de la couche PUT les différentes entrées de la LPE[1]. Chacune décrit en adressage physique une page mémoire à émettre et fournit le numéro de noeud destinataire. Le démon demande donc à la couche `hsldriver` de créer une entrée de MMU mappant dans l'espace virtuel du noyau la zone physique décrite dans l'entrée de LPE. Puis le démon rapatrie dans son espace virtuel ces données, et les envoie au serveur correspondant au noeud destinataire.

4.1.2 Fonctionnement du démon `hslserver`

Le démon `hslserver` procède en deux phases :

1. Phase de configuration : le démon HSLSERVER reçoit les connexions des clients lui indiquant la plage physique reservée par les couches SLR/P des noeuds distants. Il en informe donc la couche SLR/P locale.

1. LPE : Liste des Pages à Emettre, table gérée par le composant PCI-DDC décrivant le travail qui lui est demandé.

2. Phase d'émulation (en boucle) : le démon HSLSERVER reçoit alors les données envoyées par les clients. Pour chaque paquet de données, il demande à la couche HSLDRIVER de créer une entrée de MMU mappant dans l'espace virtuel du noyau la zone physique décrite dans les données reçues. Il les transfert alors à la couche HSLDRIVER qui les dépose dans la zone physique associée. Enfin, il signale à la couche PUT l'arrivée de données, afin qu'elle simule une interruption matérielle, s'il y a lieu.

4.2 Les fichiers de configuration

Chaque noeud du réseau se voit attribuer un numéro particulier : il constitue le numéro de noeud et est utilisé dans le champs destination des entrées de LPE. C'est un paramètre qui doit être passé aux fonctions de communication des couches PUT, SLR/P et SLR/V.

La description des noeuds constituant le réseau HSL se situe dans un fichier de configuration qui est lu par le démon `hslclient`, à son lancement. Il contient une ligne lui indiquant le numéro du noeud local, et une ligne par noeud distant informant du nom de domaine de la machine correspondante, ainsi que des numéros de programme RPC et de version de la couche de communication.

On nomme généralement ce fichier `/etc/hsl.conf`.

Prenons pour exemple une machine MPC constituée de deux noeuds : le noeud 0 s'appelle `isis.ibp.fr` et le noeud 1 s'appelle `aptinfo.ibp.fr`.

Le fichier de configuration sur `isis.ibp.fr` ressemble donc à :

```
isis% cat /etc/hsl.conf
# local node format :
# node NODE_NUMBER

node 0

# node format :
# node NODE_NUMBER hostname HOSTNAME program PROGRAM_NUMBER version VERSION

node 1 hostname aptinfo.ibp.fr. program 101000 version 1

# EOF
isis%
```

Le fichier de configuration sur `aptinfo.ibp.fr` ressemble alors à :

```
aptinfo% cat /etc/hsl.conf
# local node format :
# node NODE_NUMBER

node 1

# node format :
# node NODE_NUMBER hostname HOSTNAME program PROGRAM_NUMBER version VERSION

node 0 hostname isis.ibp.fr. program 101000 version 1

# EOF
aptinfo%
```

4.3 Activation des démons

Une fois les fichiers de configuration créés et déposés dans `/etc/`, nous pouvons activer les démons `hslclient` et `hslserver`.

Commençons par exemple par lancer les serveurs. Sur `isis.ibp.fr`, nous entrons la commande suivante :

```
isis% su
isis# /usr/local/bin/hslserver
```

Sur `aptinfo.ibp.fr`, nous lançons la même commande :

```
aptinfo% su
aptinfo# /usr/local/bin/hslserver
```

Nous lançons maintenant le client sur `isis.ibp.fr`, sur un autre terminal que celui qui a servi à lancer le serveur :

```
isis% su
isis% /usr/local/bin/hslclient -f /etc/hsl.conf -d /dev/hsl
Parsing /etc/hsl.conf
my node number is 0
foreign node 1
  hostname : aptinfo.ibp.fr.
   program : 101000
   version : 1

Opening /dev/hsl

Trying to connect to servers
connected to aptinfo.ibp.fr., program 101000, version 1
init_put: HSLPUTINIT : 0x5
Sending contig_space (0xac7000) to server node 1
Waiting for LPE entries
```

C'est au tour du client sur `aptinfo.ibp.fr` :

```
aptinfo% su
aptinfo# /usr/local/bin/hslclient -f /etc/hsl.conf -d /dev/hsl
Parsing /etc/hsl.conf
my node number is 1
foreign node 0
  hostname : isis.ibp.fr.
   program : 101000
   version : 1

Opening /dev/hsl

Trying to connect to servers
connected to isis.ibp.fr., program 101000, version 1
init_put: HSLPUTINIT : 0x10005
Sending contig_space (0xc33000) to server node 0
Waiting for LPE entries
```

Nous vérifions enfin que la phase de configuration s'est correctement passée sur les serveurs. Sur `isis.ibp.fr`, on peut lire :

```
aptinfo% su
aptinfo# /usr/local/bin/hslserver
Configuration received from 1 : 0xc33000
```

Et sur `aptinfo.ibp.fr`, on peut lire :

```
aptinfo% su
aptinfo# /usr/local/bin/hslserver
Configuration received from 0 : 0xac7000
```

Comme nous l'avons précédement indiqué, la couche SLR/V réserve de la mémoire à l'initialisation, auprès du gestionnaire de mémoire contigüe CMEMDRIVER. Nous pouvons le vérifier en interrogeant `/dev/cmem` :

```
isis% cat /dev/cmem
/dev/cmem : low level driver for contig. space

physical start :   0xac3000
virtual  start : 0xf4b2e000
length         :    1048576

+------+------------+------------+----------+----------+
| slot | phys start | virt start |     size |     name |
+------+------------+------------+----------+----------+
|    0 |   0xac3000 | 0xf4b2e000 |    16384 |      PUT |
+------+------------+------------+----------+----------+
|    1 |   0xac7000 | 0xf4b32000 |    20480 |    SLR/P |
+------+------------+------------+----------+----------+
|    2 |        0x0 |        0x0 |        0 |          |
+------+------------+------------+----------+----------+
|    3 |        0x0 |        0x0 |        0 |          |
+------+------------+------------+----------+----------+
isis%
```

Les services de communication MPC sont maintenant disponibles.

Chapitre 5

Conclusion

Mis-à-part USRPUT, les couches de communication sont accessibles uniquement depuis le noyau FreeBSD et sont disponibles au sein de modules chargeables dynamiquement. Pour utiliser les API fournies, il n'est pas possible de modifier directement le noyau car, au moment de sa compilation, les points d'entrée de PUT, SLR/P et SLR/V ne sont pas disponibles. C'est seulement lorsque les modules noyau CMEMDRIVER et HSLDRIVER sont chargés que les points d'entrée sont disponibles. Les applications utilisatrices des couches de communication noyau doivent donc elles aussi être développées au sein de modules noyau chargeables dynamiquement, seul moyen de bénéficier d'une phase d'édition de liens dynamique au moment du chargement dans l'espace mémoire du noyau.

Le choix d'orienter les développements autour de modules noyau chargeables dynamiquement permet un gain appréciable de temps lors des différentes phases de la conception d'une application, car ainsi, il n'est pas nécessaire de relinker le noyau entier ni de rebooter l'ensemble des noeuds à chaque modification logicielle.