1, Avenue Schweitzer
33400 Talence

ETUDE ET MISE EN ŒUVRE D UN AGENT

SNMP SOUS linux

PUIS INTEGRATION SOUS ì Clinux

Stage du 19 mars au 15 juin 2001

Table des matières

1. Introduction

C'est mon intérêt pour les réseaux informatiques qui m'a poussé vers cette étude. En effet, à l'heure de l'internet, l'information transmise via ce nouveau moyen de communication devient un enjeu aussi bien technique qu'économique. Ce médium revêt une importance toute particulière pour les entreprises, dans la mesure où il autorise à l'entreprise, non seulement de communiquer avec l'extérieur (que ce soit par mail, par l'intermédiaire de son site internet…), mais aussi d'assurer toute sa communication interne, optimisant ainsi les relations entre la direction et le personnel.

De ce dernier aspect surgit la nécessité de l'administration de réseau, puisque le responsable interne doit pouvoir faire face à toute une batterie de problèmes potentiels à résoudre. De plus, la sécurité devient aussi un enjeu économique, dans la mesure où n'importe quel utilisateur ne doit pas pouvoir avoir accès à certaines informations présentes sur le réseau interne de l'entreprise. La sécurité est aussi un aspect de l'administration des réseaux.

Mais mes motivations d'effectuer mon projet de fin d'étude au sein d'un laboratoire rattaché à l'ENSEIRB ne réside pas seulement en mon intérêt pour l'administration de réseau. Je voulais aussi pouvoir comparer mon expérience de début d'année scolaire en entreprise avec le travail de laboratoire, qui, se déroulant dans un environnement tout à fait différent, m'a permis de prendre conscience de certains des choix que j'aurai à effectuer sur le plan professionnel.

Enfin, le dernier point qui m'a amené à choisir ce projet est l'environnement de travail. J'avais en effet beaucoup entendu parlé de linux, des comparaisons avec Windows, de sa communauté très active, ainsi que de son essor actuel dans le domaine de l'embarqué. J'avais donc envie de me familiariser avec cet outil apparemment stable, modulable, et dont la gratuité entraîne tout un engouement propice à son amélioration constante.

2. Les objectifs de la mission technique

2.1. Sujet de stage

Le but de ce projet est de montrer la faisabilité de piloter du matériel par SNMP. Il est pour cela nécessaire de mettre en place un agent SNMP sous linux (UCD-SNMP) pour étudier le protocole SNMP, puis de developper un subagent pour piloter des entrées/sorties (leds) connectées sur le port parallèle du PC.

Enfin, il faudra réaliser le portage du subagent sur la carte ColdFire Motorola sous µClinux, puis adapter le code nécessaire à l'allumage des leds.

Ce projet, réalisé dans le cadre de l'ENSEIRB doit permettre de mettre en place des TP pour la filière Telecom, en vue de l'étude de l'administration de réseau par SNMP.

2.2. Cahier des charges fonctionnel

Installation et mise en œuvre des protocoles SNMP sous linux avec le produit UCD-SNMP. Cette étape doit aussi permettre de maîtriser l'environnement RED HAT fonctionnant sous linux.

Développement d'un agent SNMP spécifique (subagent) sous linux.

Intégration du subagent sur carte ColdFire MF5407 sous µClinux.

2.3. Moyens mis à disposition

Un micro-ordinateur PC sous linux en réseau.

Le package UCD-SNMP.

Le package uClinux-coldfire.

La carte d'évaluation Motorola ColdFire MF5407 en réseau.

2.4. Planification du projet

Période	19 mars – 1 avril	15 avril – 23 avril	25 avril – 6 mai
Tâche accomplie	-mise en place de linux sur le PC. -mise en place du package UCD-SNMP sur le PC. -première approche de l'administration de réseau.	-apprentissage des fonctions SNMP de base. -apprentissage de la structure du logiciel UCD-SNMP. -application de l'exemple fourni dans le package UCD-SNMP.	-écriture du fichier ENSEIRB-MIB.txt en langage ASN.1. -utilisation de l'outil mib2c pour la génération des fichiers C.
Période	9 mai – 21 mai	23 mai – 1 juin	3 juin – 15 juin
Tâche accomplie	-fabrication d'une plaquette de test destinée à être connectée sur le port parallèle du PC. -adaptation du fichier enseirb.c pour allumer les leds de la carte de test	-installation du package µClinux-ColdFire sur le PC. -portage de l'agent sous µClinux. (modification de différents fichier servant à la configuration du noyau µClinux)	-modification du code servant à l'allumage des leds sur le port parallèle du MCF5407. -écriture du rapport de projet.

3. Réalisation du projet

3.1. Généralités sur SNMP et le package UCD-SNMP

3.1.1. Nécessité d'une gestion de réseau

La première question que l'on se pose lorsque l'on travaille avec un protocole de gestion de réseau est : pourquoi a-t-on besoin de gérer le réseau ?
Le principe de gestion de réseau naît de la nécessité de connaître l'état du réseau pour son administrateur, qui doit :

-Surveiller et réparer les anomalies comme un câble coupé ou autre…
-S'occuper de toutes les configurations, que ce soit sur les postes ou sur les éléments du réseau.
-Gérer toute la sécurité du réseau (mots de passe, firewalls….)
-Mesurer et analyser les performances du réseau.

Cependant, plusieurs problèmes se posent à ce stade du raisonnement :

-Tout d'abord l'hétérogénéité des équipements, ainsi que celle des constructeurs. Ce problème interdit l'utilisation d'une technologie particulière à un constructeur, un protocole de gestion «ouvert » (non-propriétaire) doit donc être mis en œuvre, tout comme la technologie d'interconnexion « ouverte » a permis aux interconnexions hétérogènes de devenir une réalité.

-Ensuite, la multiplicité des organisations qui gèrent certaines portions de l'internet et qui n'ont pas forcément les mêmes objectifs, que ce soit en terme de coûts ou de performances. Ce problème empêche de fédérer la gestion des équipements.

Cette liste n'est bien sûr pas exhaustive, mais permet de visualiser l'essentiel des difficultés rencontrées par un administrateur réseau.

De cette nécessité d'un protocole de gestion «ouvert » est né SNMP (Simple Network Management Protocol.)

3.1.2. SNMP

3.1.2.1.Fonctionnement SNMP

Le protocole SNMP (Simple Network Management Protocol) a été développé dans les années 80 pour permettre à l'administrateur du réseau d'interroger les éléments de son réseau sans se déplacer. Le principe de SNMP est très simple : sur chacune des machines, on installe un petit programme : l'agent SNMP. Cet agent enregistre en permanence des informations relatives à la machine. Il stocke ces informations dans une MIB (Management Information Base), c'est à dire une base de données.

Ainsi, de son ordinateur, l'administrateur peut interroger chacune de ses machines et obtenir les informations qu'il souhaite, comme par exemple le nombre d'octets reçus et envoyés... Il peut aussi modifier certaines informations.

Le manager peut donc opérer de deux manières différentes sur le réseau:

-**request** (communément appelée get-request), qui permet au manager de scruter la MIB d'un agent particulier.

-**set** (ou set-request), qui autorise le manager à fixer la valeurs de certains objets gérés. C'est d'ailleurs cette propriété que nous utiliserons pour piloter du matériel à distance par SNMP.

Le protocole SNMP fonctionne au niveau 7 du modèle OSI, mais se situe directement au-dessus d'UDP. Il fonctionne sur un modèle **client-serveur**, où il n'y a qu'un seul client, la station d'administration (NMS = Network Management Station) et beaucoup de serveurs (chaque agent SNMP), le client interrogeant les serveurs pour récupérer les informations. En utilisant les protocoles de la couche UDP, SNMP acquière la transparence vis a vis du support physique, cependant, en cas de défaillance de la couche transport, le protocole ne peut informer l'administrateur.

Chaque agent est placé sur un nœud du réseau qui est dit administrable (MN : Managed Node). Ces nœuds peuvent être soit des hôtes (stations de travail ou serveurs), soit des éléments d'interconnexion (switchs, hubs, routeurs), soit des supports physiques (câbles).

Les **traps** permettent aux serveurs d'envoyer des infos non demandées par le client pour l'informer sur certains événements (erreurs, ruptures de ligne …).

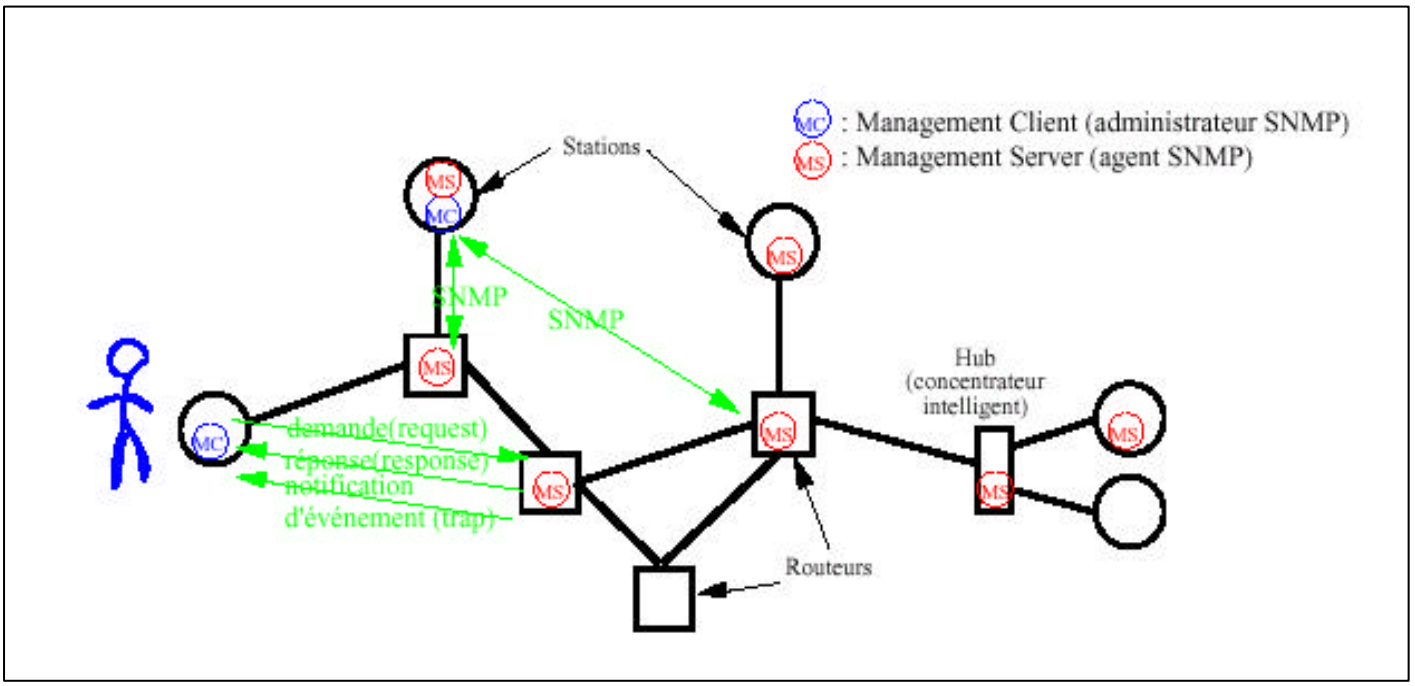

Figure 1: Architecture de la gestion de réseau SNMP

3.1.2.2. Qu'est-ce que la MIB ?

Pour se retrouver dans la foule d'informations proposées par chaque agent, on a défini une structure particulière pour les informations appelée SMI (Structure of Management Information). Chacune des informations de la MIB peut être retrouvée soit à partir de son nom de variable, soit à partir d'un arbre de classification. Cela revient à parcourir des sous-dossiers et dossiers d'un disque dur…

Supposons que vous souhaitiez consulter la variable System d'un hôte, vous pouvez soit lui demander la variable System directement, soit lui demander la variable ayant pour OID (Object IDentification) 1.3.6.1.2.1.1… correspondant à l'arborescence de la variable (ISO, Identified Organization, dod, Internet, Management, MIB2, System).

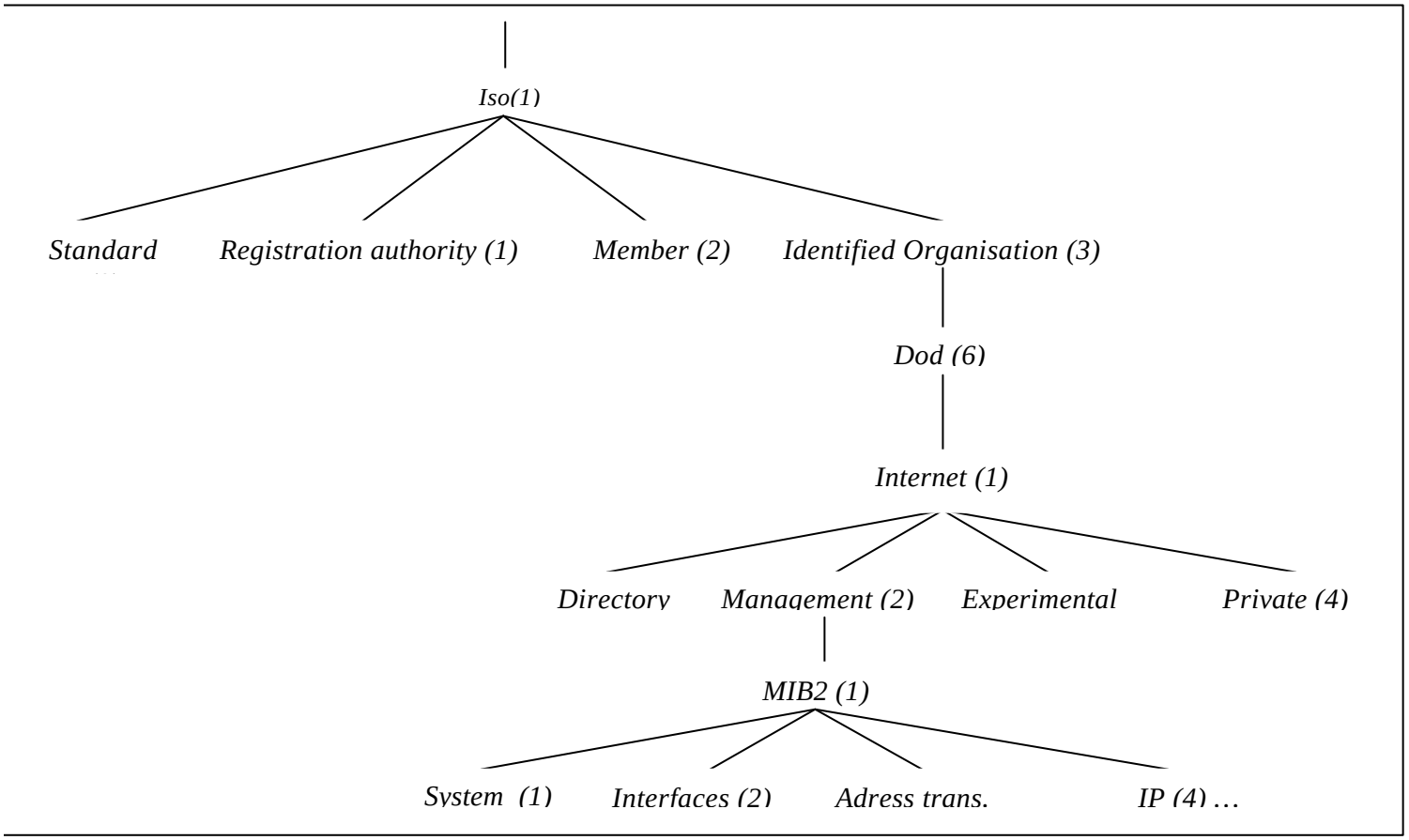

Figure 2: architecture générale de la MIB-II

Cela parait très lourd à première vue, mais le nombre de variables étant important, on ne peut se souvenir de chaque nom. Par contre, il existe de nombreux logiciels permettant d'explorer la MIB de façon conviviale (par exemple **tkmib**), en utilisant cette classification.

Chaque objet est défini par un chemin unique dans la MIB, il est donc nécessaire de souscrire un numéro identificateur à chaque fois que l'on désire créer un nouvel objet. Cette démarche se fait auprès de l'**IANA** (Internet Assigned Numbers Authority) par simple mail.

Pour information, le numéro assigné à l'ENSEIRB est 9362.

3.1.2.3.Structure informationnelle

L'information de gestion communiquée par les opérations du protocole est codée dans une syntaxe qui correspond à un sous-ensemble de la syntaxe normalisée **ASN.1**. Nous verrons les types d'objets utilisables et la structuration de ceux-ci regroupés par **groupes**. On notera que le terme **objet** correspond uniquement à des variables informatiques classiques.

Voici la macro ASN.1 utilisée pour définir des objets SNMP :

```
MODULE-IDENTITY MACRO : : =
BEGIN
        TYPE NOTATION : : =
                ″LAST-UPDATED″   Value (Update UTCTime)
                ″ORGANISATION″   Text
                ″CONTACT-INFO″   Text
                ″DESCRIPTION″    Text
                RevisionPart
        VALUE NOTATION : : =
                Value (VALUE OBJECT IDENTIFIER )
                Access (R/W, Ronly…)
                Status (optional, obsolete…)
        RevisionPart : : =
                Empty
END
```

Figure 3: déclaration d'objets SNMP

ASN.1 est un méta-langage qui permet de représenter les types et les structures de données. Les structures, décrites en utilisant ASN.1, seront ensuite codées suivant des règles BER (Basic Encoding Rules) dans le but de les préparer pour un éventuel envoi sur le réseau. Ce codage suit la séquence classique TLV (Type, Longueur, Valeur)

Les variables que gère SNMP sont regroupées en plusieurs listes :

SystemGroup :	liste des informations de configuration.
InterfacesGroup :	liste des informations génériques sur les interfaces.
AtGroup :	liste de traduction d'adresse.
IpGroup :	liste des variables liées au protocole ICMP.
TCPGroup :	liste des variables liées au protocole TCP.
UDPGroup :	liste des variables liées au protocole UDP.
EGPGroup :	liste obligatoire dans les nœuds qui implémentent EGP.
TransmissionGroup :	liste apparue dans la version MIB-II. Elle regroupe l'ensemble des variables qui correspondent aux différents types d'interfaces, token-ring, loopback…
SnmpGroup :	liste apparue dans la version MIB-II qui contient les variables se rapportant à SNMP.

3.1.2.4.Les propriétés de SNMP

- **La sécurité** : La conjugaison d'un agent SNMP avec un ensemble d'entités d'application s'appelle une communauté SNMP. Chaque communauté SNMP est identifiée par une chaîne d'octets, le nom de la communauté.

Pour assurer la sécurité des opérations de gestion, le protocole SNMP utilise la notion de message authentique qui se définit comme un message pour lequel on a contrôlé que l'entité d'application émettrice est bien membre de la communauté spécifiée dans ce message.

Il est clair qu'une administration sécurisée utilisant des entités d'applications bâties sur SNMP doit disposer de services d'authentification capables d'identifier les messages SNMP avec un fort taux de crédibilité.

Bien sûr, certaines implémentations peuvent souhaiter ne supporter qu'un service d'authentification trivial, ce qui revient à accepter tous les messages comme authentiques.

Avec les versions 2 et 3, la sécurité a été grandement améliorée, mais il ne s'agit pas là d'un enjeu capital puisque l'on se sert toujours de la version 1, bien que la version 2 soit disponible. La simplicité de SNMP est donc plus importante que son aspect sécuritaire.

-**La distribution** : En matière de distribution, la politique retenue pour SNMP est la minimisation du nombre et de la complexité des fonctions de gestion, et donc, de la charge de travail de la machine gérée (nœud).

Ceci a pour conséquences :

Une réduction des coûts de développement logiciels des agents de gestion nécessaires au support du protocole.

Une limitation des restrictions des outils de gestion développés sur les stations de gestion.

Une assimilation rapide par les développeur d'ensembles simples de fonctions de gestion.

3.1.3. Les particularités du produit UCD-SNMP

Pour se familiariser avec le protocole SNMP en utilisant le logiciel UCD-SNMP (University of California Davis), un kit « d'entraînement » est fourni. Ce kit permet d'utiliser les fonctions SNMP de base, mais donne aussi un exemple d'extension de l'agent dont on doit s'inspirer pour réaliser son propre agent.

3.1.3.1. Les fonctions SNMP de base

snmpget : application qui utilise une requête **get** pour obtenir une information d'une entité distante, c'est à dire obtenir la valeur d'un objet présent sur un nœud géré. La syntaxe est la suivante :

snmpget machine password objet

snmpset : application qui utilise une requête **set** pour forcer la valeur d'un objet dans la MIB distante, si bien sûr cet objet est accessible en écriture. La syntaxe est cette fois légèrement différente (il faut donner une valeur à l'objet distant) :

snmpset machine password objet type valeur

snmpgetnext : cette fonction est analogue à snmpget, cependant qu'elle donne la valeur de l'objet suivant dans la MIB, si toutefois il en existe un. La syntaxe est la même que pour snmpget.

snmpwalk : c'est une boucle de snmpgetnext appliquée sur un groupe de la MIB. Cette fonction donne alors l'état de tous les objet présents dans le groupe. La syntaxe est toujours la même mis à part le fait que l'argument *objet* est remplacé par un argument *groupe*.

snmptrap : cette fonction permet d'envoyer des traps depuis un agent un ou plusieurs OID peuvent être donnés en arguments, cependant le type de chaque objet doit être signifié.

snmptable : cette fonction est analogue à snmpwalk pour une table SNMP.

snmptranslate :cette fonction permet de convertir les données d'identification d'un objet. Selon les options, la conversion se fait du nom vers le chemin dans la MIB, ou inversement. On peut obtenir de nombreuses autres informations avec cette fonction.(voir : http://net-snmp.sourceforge.net)

D'autres fonctions existent, mais je ne les ai que très rarement utilisées, vous pourrez également trouver ces fonctions ainsi que leur manuel d'utilisation sur : http://net-snmp.sourceforge.net

3.1.3.2. L'exemple fourni avec le package UCD-SNMP

Dans cet exemple, on apprend comment ajouter un objet dans la MIB existante d'agent. L'objet ajouté dans ce cas est une simple chaîne de caractères, mais toute la méthodologie de synthèse est identique à celle que j'ai utilisée pour développer mon agent spécifique.

3.2. Développement de l'agent sous linux

3.2.1. Architecture du logiciel UCD-SNMP

Reconnu comme l'un des meilleur packages existants, il contient un agent extensible, une librairie SNMP, des outils de requêtes **get** ou **set** pour interroger l'agent, des outils pour générer des **traps**, un « navigateur de MIB » (tkmib). Il est de plus portable sur une grande majorité de systèmes d'exploitation.

L'avantage essentiel de ce package est qu'il permet le développement d'un nouvel objet très rapidement. En effet, une fois le code écrit en langage ASN.1, le compilateur mib2c crée la trame des fichiers .c et .h qui doivent être utilisés, un temps substantiel est ainsi gagné.

Ensuite, le package possède un programme de configuration qui lui permet d'intégrer simplement le subagent dans sa MIB.

3.2.1.1. Les fichiers MIB implémentant les objets

Vous pourrez consulter ce fichier en **annexe page 1**.

Le premier fichier nécessaire lorsque l'on implémente un nouvel objet dans la MIB est le fichier MIB. Il n'est pas absolument nécessaire au sens strict, mais il peut cependant être très utile, et cela pour trois raisons :

- Il donne une première spécification de ce qui va être implémenté
- Si ce nouveau fichier est lu en même temps que les autres fichiers MIB, l'agent intégrera le nouvel objet (si celui-ci fonctionne correctement)
- L'outil mib2c permet à partir de ce fichier MIB d'obtenir les fichiers.c et .h nécessaire au nouvel agent. (la simplicité de codage est totale)

Ces fichiers MIB sont des fichier de type .txt et sont localisés dans le répertoire :

/usr/local/share/snmp/mibs/ .

La question qu'il reste à se poser à ce stade est : quel nom donner à son nouvel objet, et où le localiser dans la MIB. Ce point est tout à fait interne mais doit être traité avec lucidité pour que les utilisateurs futurs puissent utiliser ce nouvel objet de la manière la plus naturelle possible. L'agent développé pour allumer les leds comporte donc un groupe Enseib, dans lequel se trouve le groupe Led contenant plusieurs variables correspondant à plusieurs leds.

3.2.1.2. Les fichiers C et leurs particularités

Ces fichiers sont localisés dans le répertoire */root/ucd-snmp/agent/mibgroup/* . Vous pourrez les consulter en **annexes 2 et 3**.

Je m'attarderai tout d'abord sur le fichier C header, où l'on déclare (de même s'il est crée par l'outil mib2c) trois prototypes de fonctions :

-extern void *init_nom_du_fichier(void) ;*
-extern FindVarMethod *var_nom_du_fichier ;*
-extern writeMethod *write_varName ;* (uniquement lorsque l'on veut implémenter un objet accessible en écriture)

Ce fichier est aussi utilisé par le compilateur, de sorte que l'on y définit les dépendances avec les autres fichier header. La syntaxe est : *config_require(util_funcs)*

Ce fichier permet aussi d'inclure le fichier MIB implémenté dans l'agent qui va être réalisé par la commande : *config_add_mib (HOST-RESOURCES-MIB)*

Le dernier élément classique présent dans ce fichier est le jeu de « magic numbers ». Il en existe un pour chaque objet défini. Ce n'est cependant pas le principe utilisé par mib2c qui préfère les déclarer dans le fichier de code.

Structure du fichier de code donné par mib2c :

INCLUDES
DEFINITIONS DU MODULE
INITIALISATIONS DU MODULE
VARIABLES UTILES
FONCTIONS NECESSAIRES EN LECTURE
FONCTIONS D'ECRITURE NECESSAIRES POUR UN OBJET ACCESSIBLE EN ECRITURE

Figure 4 : structure du fichier de code

Ce fichier est lui aussi obtenu à l'aide de l'outil mib2c.

Certains INCLUDES sont si fréquemment utilisés qu'ils sont insérés par défaut. Ce sont les fichiers *config.h* dont mib2c note qu'il doit toujours être inclus avant tout, ainsi que *mibincl.h* et *util_funcs.h*. il est aussi nécessaire d'inclure le fichier *enseirb.h*, spécialement édité pour cet objet.

Ensuite apparaît la ligne *oid enseirb_variables_oid[ ] = { 1,3,6,1,4,1,9362}* qui donne l'emplacement de l'objet dans la MIB. On peut remarquer que cet objet se situe dans la liste *private/enterprises* , qui est l'endroit où doivent être déclaré les nouveau objets.

Comme ce fichier est généré par mib2c, la déclaration des *magic numbers* se fait à ce stade. Cette déclaration se fait de la manière suivante :

```
Struct variable2 enseirb_variables[ ] = {
//Magic number          ,variable type  ,ro/rw  ,callbackfn     ,L      ,oidsuffix
#define LED0            1
        { LED0          ,ASN_INTEGER  ,RWRITE, var_enseirb, 2       , {1,1} },
} ;
/* L = longueur de oidsuffix */
```

Et ainsi de suite pour chaque nouvelle LED déclarée.

Est ensuite donné le corps de la fonction *init_enseirb* dont le prototype a été déclaré dans le fichier header. Une seule initialisation y est effectuée par défaut :

REGISTER_MIB("enseirb", enseirb_variables, variable2, enseirb_variables_oid) ;

Cependant, si d'autres initialisations sont nécessaires, elles doivent être effectuées dans cette fonction. Pour l'objet que nous avons crée, il nous a fallu initialiser un certain nombre de variables suivant les utilisations de ce fichier, que ce soit sous linux ou ì Clinux.

Sous linux, puisque c'est l'objet de cette partie, il a fallut initialiser les variables nécessaires à l'allumage des leds sur le port parallèle du PC. Ces initialisations sont :

```
ioperm ( lp_base, 1, 1) ;          // autorisant l'écriture sur le port parallèle
lpdata = 0 ;                       // initialisation de la variable lpdata à 0
outb ( lpdata, lp_base) ;          // écriture de la valeur lpdata = 0  sur le port parallèle
```

les variables sont déclarées juste après la déclaration des includes :

lpdata est un static char
lp_base est l'adresse du port parallèle (0x378)

Dans la suite du code, on trouve deux ensembles de fonctions qui constituent le corps du fichier :

- Tout d'abord, la fonction *var_enseirb( )* qui est appelée chaque fois que l'agent envoie un requête concernant un objet de la MIB alors constituée. La fonction permet alors de retourner la valeur courante de la variable demandée, cette valeur est *etat*, variable déclarée en *static char*.
- Ensuite, les fonctions permettant l'écriture des valeur des variables par le manager, les fonctions *write_ledi(int action,*

```
        u_char      *var_val,
        u_char      var_val_type,          //i étant le numéro de la led à allumer
        size_t  var_val_len,
        u_char      *statP,
        oid     *name,
        size_t  name_len)
```

Cette fonction laisse dans sa partie ACTION la place d'effectuer les tâche que l'on veut imposer à l'agent. dans ce cas, il s'agit d'allumer des leds. La fonction réalisant cette opération est la suivante :

```
lpdata = inb ( lp_base ) ;          //on écrit la valeur du port dans lpdata
tmp = *long_ret << i ;                  //*long_ret contient la valeur à écrire dans le di
if (tmp)                            //si on doit écrire 1
        lpdata |= tmp ;
else    {                           //si on doit écrire 0
        tmp = 1 << i ;
        lpdata &= (~tmp) ;
        }
outb(lpdata, lp_base) ;                 //on écrit lpdata sur le port
ledi = *long_ret ;                  //on donne la bonne valeur à ledi
send_easy_trap ( SNMP_TRAP_ENTREPRISESPECIFIC, 1) ;     //on envoie un trap
```

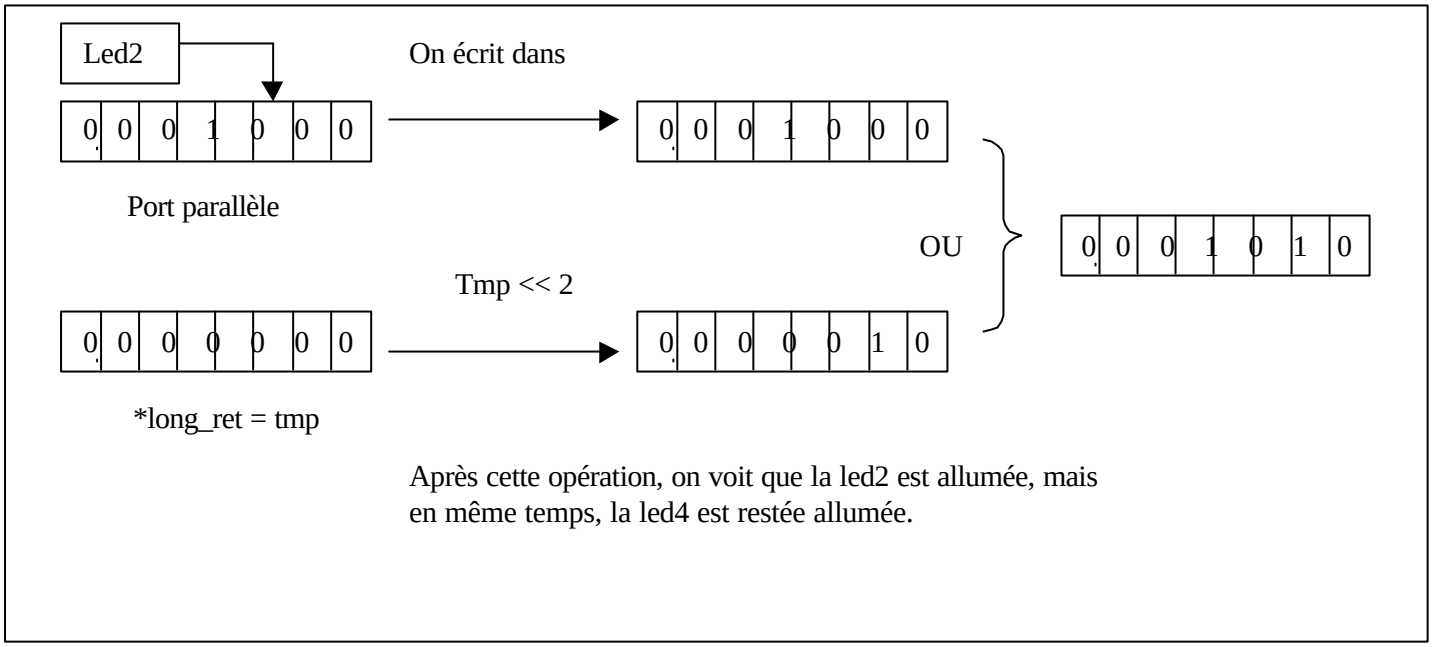

Figure 5 : algorithme d'écriture sur le port parallèle

(led2 de 0 à 1, led4 allumée)

3.2.1.3. Mise en place de l'agent sous linux

La mise en place de l'agent sous linux est on ne peut plus simple. En effet, il suffit d'exécuter les trois commandes suivantes :

- *./configure –with-mib-modules=enseirb* cette ligne permet de configurer les Makefile de manière à tenir compte du subagent.
- *make* pour construire l'exécutable
- *make instal* pour installer le démon

Il ne reste qu'à exécuter le programme en tapant **snmpd**, on peut alors utiliser les fonctions SNMP, et on peut visualiser toutes les variables de l'agent, y compris les variables ledi nouvellement implémentées.

3.2.2. Les outils utilisés pendant la phase de développement

3.2.2.1. Le compilateur mib2c

L'outil mib2c se situe sous le répertoire */root/ucd-snmp/local/* . C'est dans ce répertoire qu'il faut copier le fichier MIB. On exécute ensuite la commande **mib2c enseirb**, ainsi les deux fichiers C qui restent à copier

sous */root/ucd-snmp/agent/mibgroup/* sont créés. Cet outil permet un gain de temps de développement substantiel, puisqu'il évite au programmeur de devir coder la partie générale qui permet à l'agent de fonctionner.

Il est à noter que cet outil donne un corps de programme qui peut bien sûr être amélioré, comme le programme contenu dans *enseirb.c* a été adapté.

3.2.2.2. Le manager tkmib

Cet outil se lance depuis la racine par la commande **tkmib** et permet de visualiser la MIB de n'importe quel agent présent sur le réseau de manière conviviale. Il évite donc l'utilisation des fonctions SNMP de base qui, bien que performante, manque de souplesse. Cet outil permet de manœuvrer au sein du réseau par l'intermédiaire de la souris, ce qui est beaucoup plus rapide que de taper des lignes de commande sous linux.

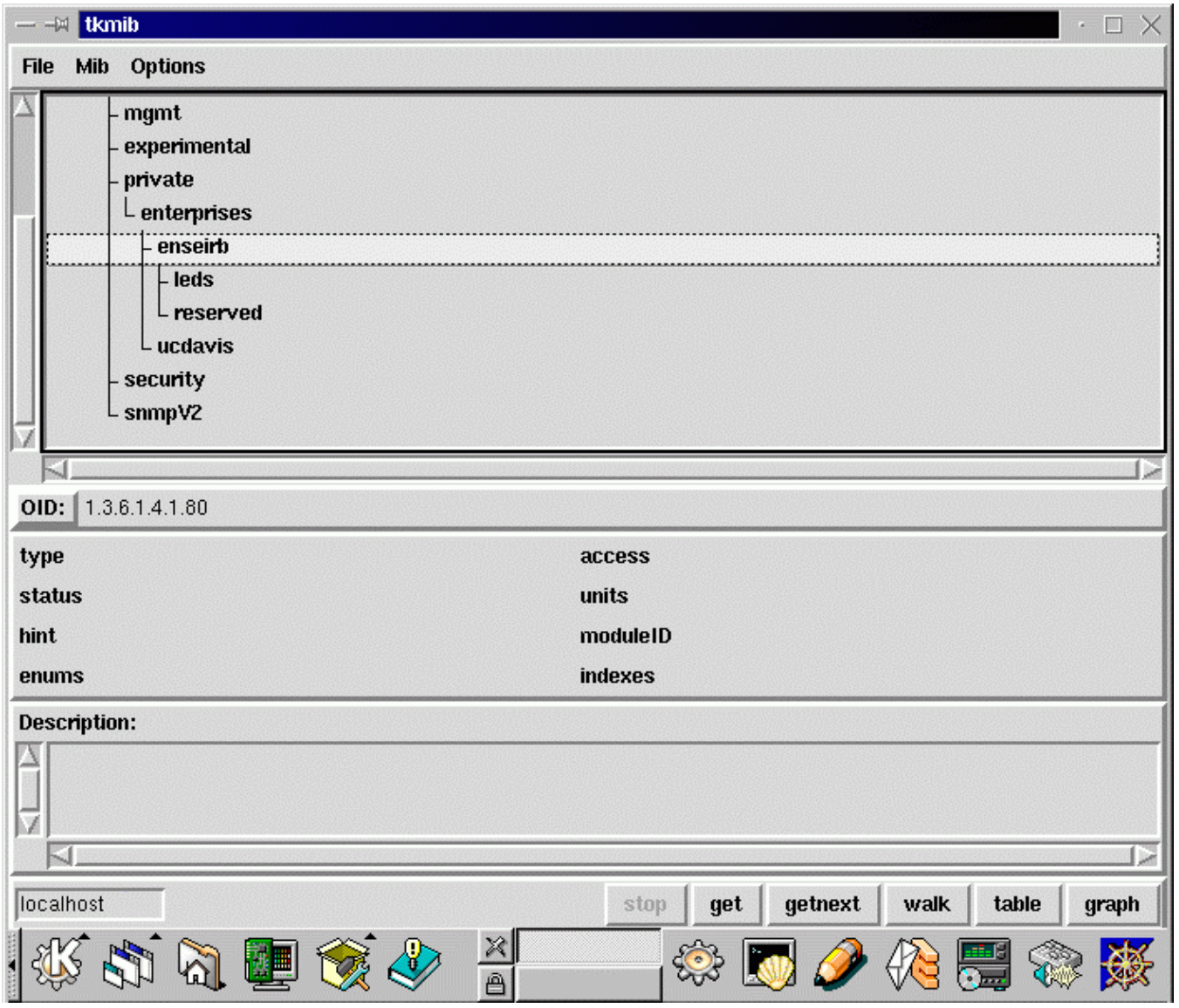

Figure 6 : le manager tkmib

3.2.3. Tests de l'agent ainsi complété

Dans un premier temps, j'ai regardé par l'intermédiaire de tkmib si l'agent que je lançais sur le PC contenait bien les nouvelles variables que j'avais codées. En effet, sous le groupe **enterprises** apparaît le sous groupe **enseirb/leds/** contenant les variables led0 à led7. Ces variables sont accessibles en écriture et leur valeur est soit 0, soit 1.

Dans un deuxième temps, j'ai réalisé sous ARES une petite carte comportant un connecteur DB25 et 8 leds (avec leur résistances de 470). Une fois connectée sur le port parallèle du PC, chaque led correspondant à une variable de la MIB, la carte donne l'état des variables ledi de la MIB.

Cette opération a bien fonctionné.

3.3. Portage de l'agent sous µClinux

3.3.1. Le package µClinux-ColdFire

Dans un premier temps, il faut installer le package sous la racine :

tar xvfz uClinux-coldfire-XXXXXXXX.tar.gz

tar xvfz uClinux-coldfire-tools-XXXXXXXX.tar.gz

Ce package contient tout le matériel nécessaire à la compilation d'un noyau µClinux destiné à être porté sur la carte d'évaluation MCF5407 de Motorola.

Il faut alors se placer sous le répertoire uClinux-coldfire qui vient d'être crée et exécuter un *make xconfig*. Se présente alors une fenêtre dans laquelle on peut sélectionner la carte d'évaluation que l'on possède pour que le logiciel configure correctement les Makefile.

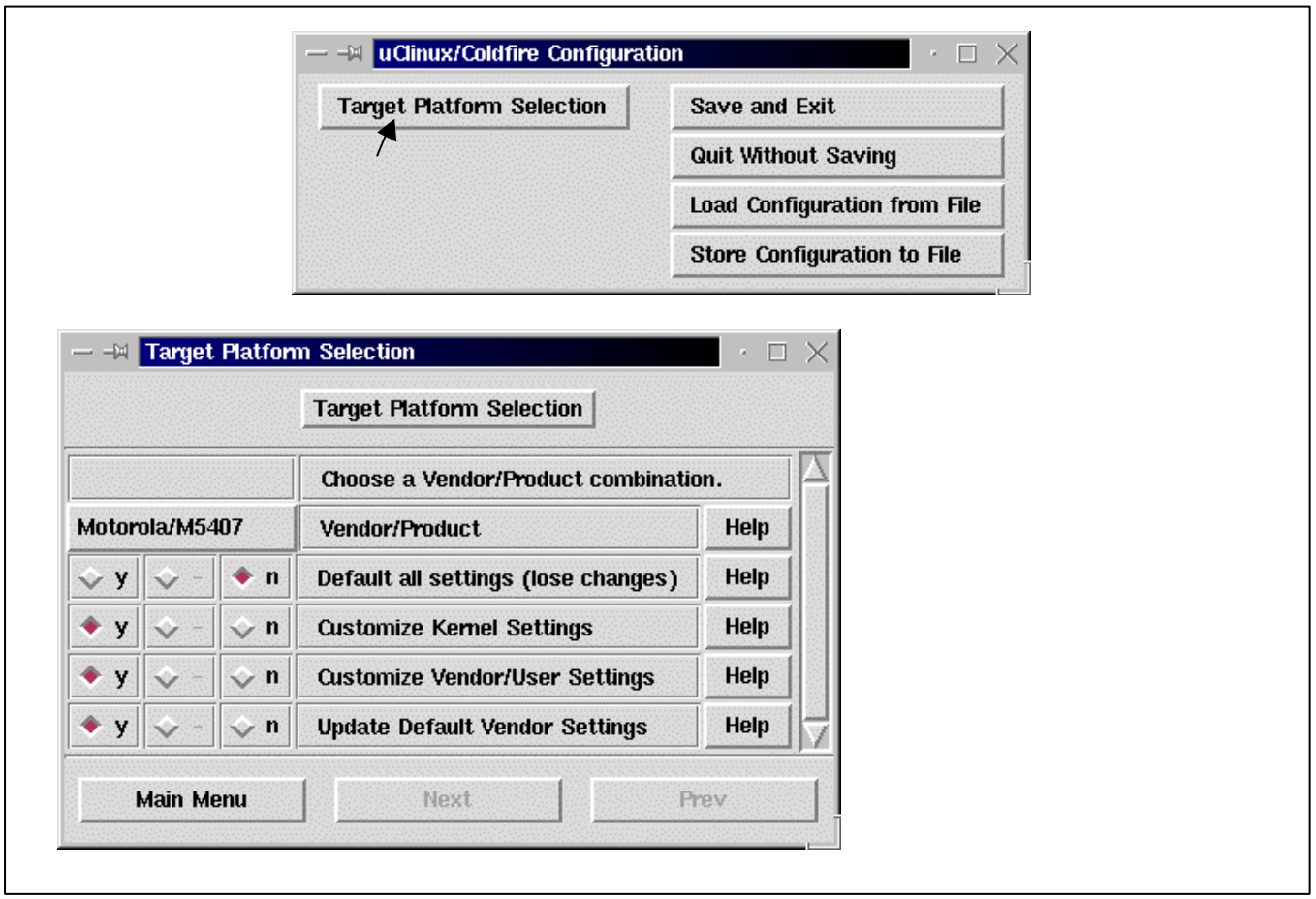

Figure 7 : fenêtres apparaissant à l'appel de xconfig

Pour utiliser le logiciel UCD-SNMP, il faut valider l'option UCD-SNMP dans la fenêtre de make xconfig.

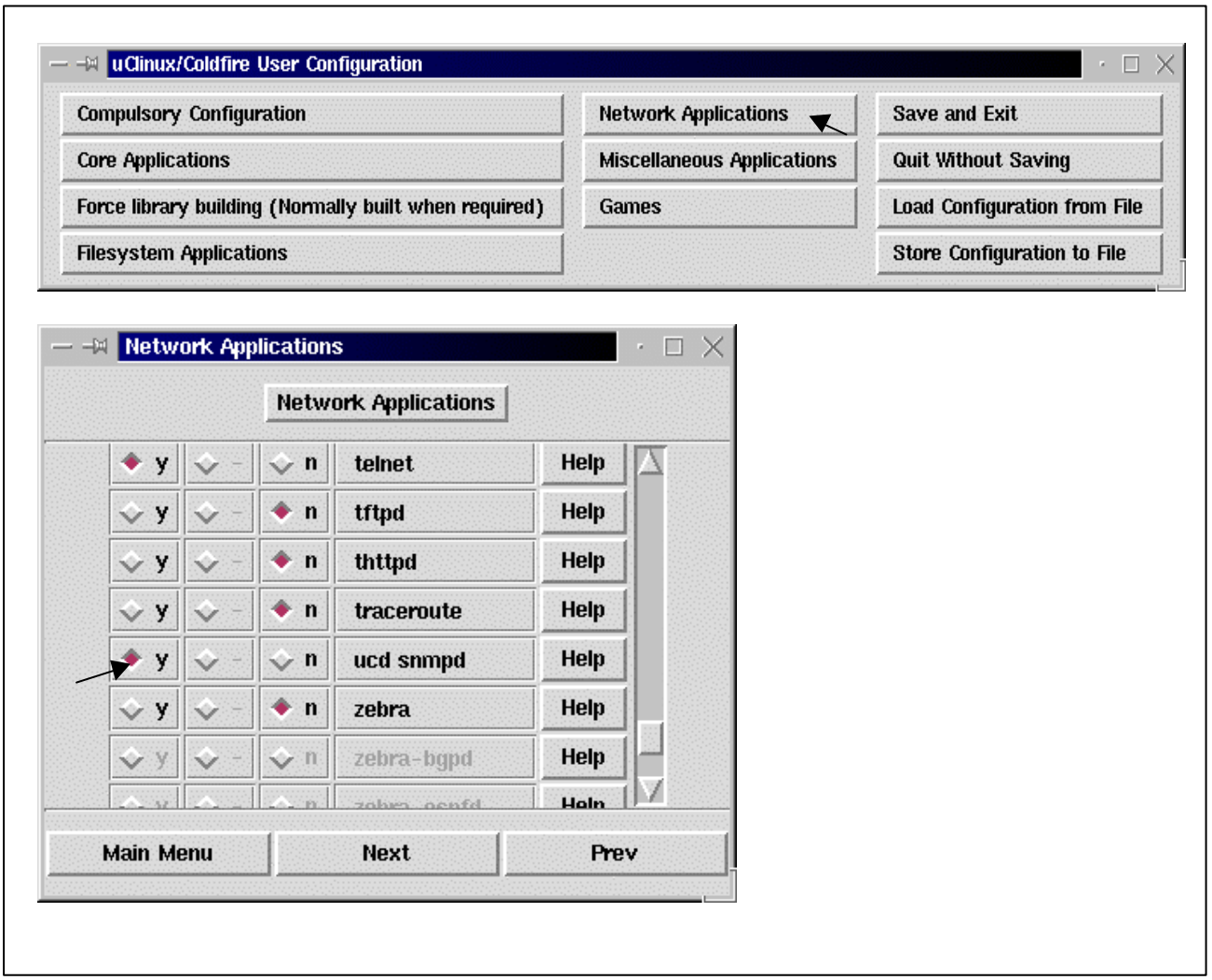

Figure 8 : fenêtres ou l'on dicte l'utilisation d'UCD-SNMP

La compilation peut alors être lancée, mais il faut d'abord pour cela construire les dépendances en exécutant un *make dep* puis construire l'image en effectuant un *make*.

Une fois l'image compilée, il ne reste qu'à la porter sur la carte d'évaluation, ce qui est facilement possible grâce à la console **kermit**.

3.3.2. L'hyperterminal Kermit

Ce terminal se lance depuis une fenêtre *xterm* par la commande **kermit**. En pressant la touche **c**, on prend la main sur le terminal avec l'invite *dBUG*. A partir de là, il faut transférer l'image sur la carte, ce qui s'effectue avec la commande **dn** (download network).

En fait, deux types de transferts sont possibles, un transfert par le réseau, ou part le port série (avec la commande **dl** :download serial). C'est le transfert par le réseau qui est choisit en raison de sa rapidité (10 Mb/s au lieu de 19200b/s).

Une fois le noyau implanté sur la carte, il faut le lancer, c'est la commande **go** qui effectue ce lancement. Il n'y a plus qu'à lui donner en argument l'addresse de début de programme qui est, pour la carte d'évaluation du MCF5407, 0x00020000.

Lorsque le système d'exploitation est lancé, le moniteur fournit à l'utilisateur une interface conviviale sous forme de répertoires et de fichier avec une invite de commande tout à fait analogue à linux.

3.3.3. Intégration de l'agent

Le problème qui se pose à ce stade de la conception, c'est l'absence de programme de configuration pour intégrer le nouvel objet développé. Cette configuration doit donc se faire manuellement, c'est le but de l'exécutable *mypatch-4.1.2*.

3.3.3.1. Le patch mypatch-4.1.2

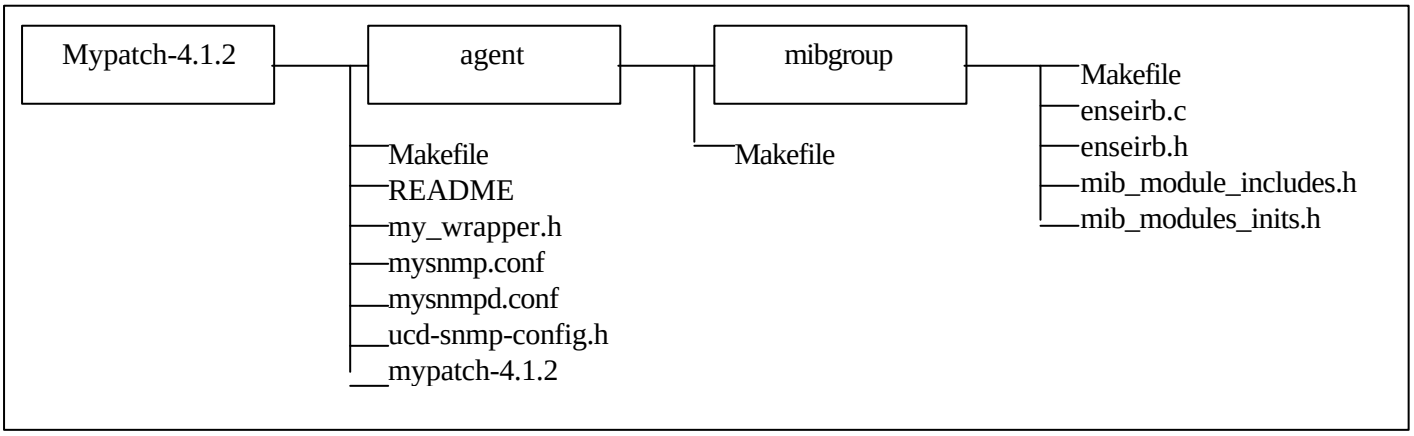

Figure 9: Le répertoire mypatch-4.1.2 (sous uClinux-coldfire/user/ucdsnmp)

Ce répertoire est construit selon la même architecture que le répertoire *ucdsnmp*, c'est à dire avec un sous-répertoire *agent* et un sous-répertoire *mibgroup*. La plupart des fichiers présents dans ce répertoire sont présentés en annexe.

Tout d'abord, le fichier *Makefile* du répertoire *mypatch-4.1.2*. Il y a une erreur à la fin de ce fichier puisque son auteur déclare copier les fichiers *snmp.conf* et *snmpd.conf* dans un répertoire *default*. Or ce répertoire n'existe pas, il a donc fallut placer ces fichiers au bon endroit, c'est à dire dans le répertoire */etc/config*.

Le fichier *my_wrapper.h* contient les includes nécessaires à la compilation du fichier *enseirb.c*. En effet, certain headers ne sont pas déclarés et le compilateur ne peut fonctionner correctement.

Les fichiers *mysnmp.conf, mysnmpd.conf* et *ucd-snmp-config.h* découlent simplement de l'exécution du programme de configuration sous linux, ce sont donc des copies. Ils sont nécessaires pour la configuration de l'agent sous ì Clinux.

Le fichier *snmpd.conf* est particulièrement interressant, puisqu'il contient les informations de réseau de l'agent ainsi que les droits de lecture et d'écriture. Ces droits sont accordés grâce à la fonction *com2sec* dont l'utilisation est la suivante :

#	*Communauté*	*manager*	*password*
com2sec	*local*	*127.0.0.1*	*tst*
com2sec	*mynetwork*	*192.9.200.10*	*tst*

Grâce à ces quelques lignes, on autorise l'appareil 192.9.200.10 à devenir le manager de l'agent qui va être porté. En l'occurrence, cette adresse IP est l'adresse de la station qui me permet de manager cet agent.

Nouss nous trouvons sous le répertoire *uClinux-coldfire/user/ucdsnmp*, le fichier *mypatch-4.1.2* va permettre de copier les bons fichiers aux bons endroits pour que le noyau puisse être généré correctement.

Dans les répertoires *agent* et *mibgroup* se trouvent les fichiers issus de la configuration sous linux qui sont immédiatement exploitables.

3.3.3.2.Le problème de l'affichage

L'affichage s'effectue, comme sous linux, sur des leds connectées au port parallèle du microprocesseur. Cependant, la démarche est assez différente. En effet, on dispose de tous les accès sur les ports, les fonctions d'initialisation des droit d'écriture sont donc dans ce cas inutiles. Il suffit de déclarer un pointeur à la bonne adresse puis de lui affecter la valeur que l'on désire.

Cela est fait de la manière suivante :

```
Short *PADAT = 0x10000244 ;                    //addresse des données du port parallèle

If (*long_ret == 1)
        *PADAT &= 0x007F ;                     //(valeur courante) & (valeur à affecter)
else
        *PADAT  |= 0x0080 ;
```

On peut remarquer que la led s'allume lorsque la valeur lui correspondant dans *PADAT est à 0.

On peut alors piloter les leds du port parallèle par la fonction **snmpset** de la manière suivante :

```
Snmpset 192.9.200.11 tst ledi.0 i 1            //pour allumer la led i
```

Cette opération peut-être effectuée depuis le manager **tkmib**.

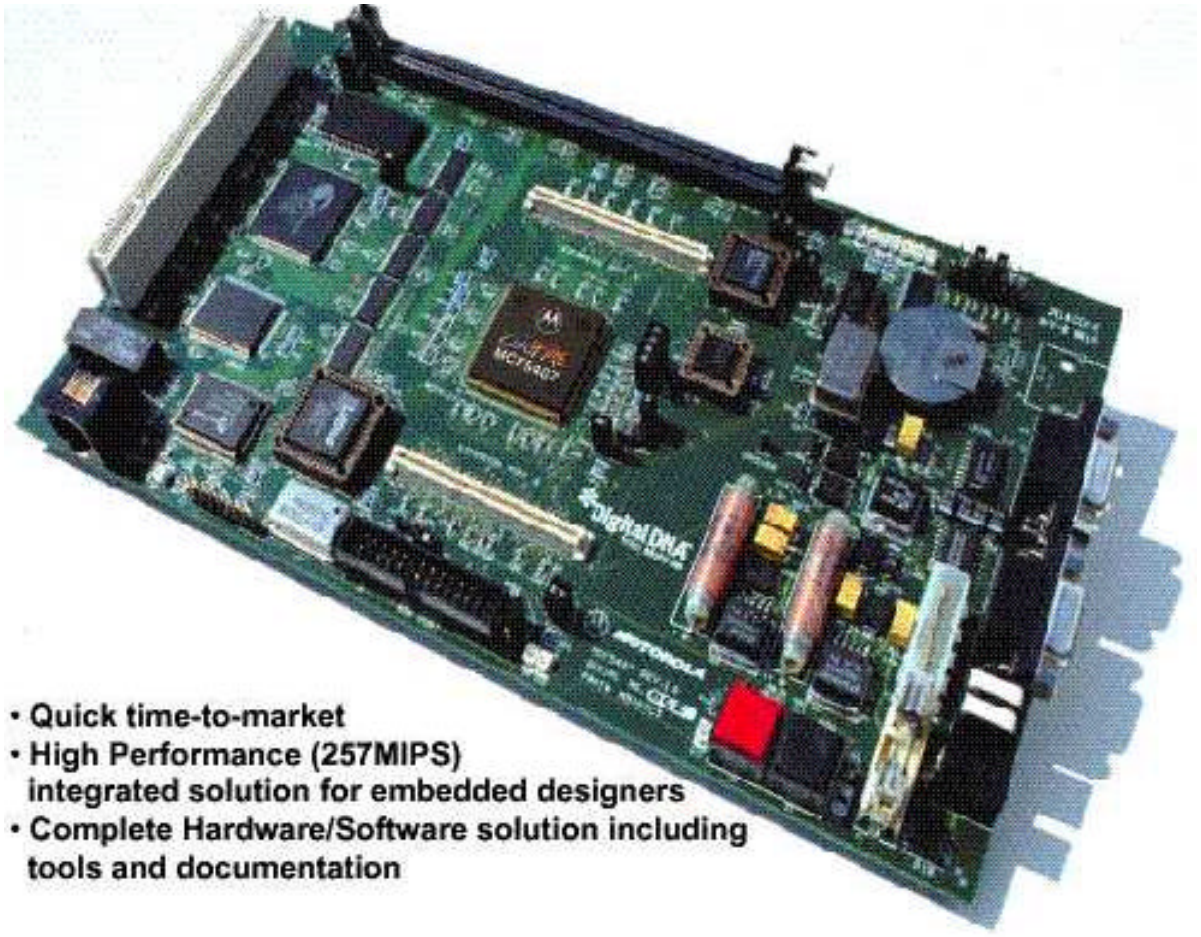

Figure 10 : carte d'évaluation motorola MCF5407

4. Conclusion

Ce projet de fin d'étude a été pour moi l'occasion de tester ce qu'était le travail en laboratoire et m'a permis de confronter deux expériences totalement différentes dans la même année, puisque j'ai effectué mon stage de troisième année en entreprise.

Le premier point de comparaison qui m'a frappé est le large panel de moyens mis à disposition des chercheurs en laboratoire, ce qui n'est malheureusement pas toujours le cas en entreprise. Le travail demandé est aussi plus précis et le cahier des charges n'évolue pas en permanence comme en entreprise où le client trouve toujours une fonctionnalité à ajouter au produit développé.

L'étendue des connaissances est aussi plus importante en laboratoire, étant donnée la diversité des missions qui doivent y être accomplies.

Plus spécifiquement, je suis heureux d'avoir pu travailler sur ce sujet. En effet, ce projet, à haute teneur en informatique, m'a permis de me familiariser avec linux, un système d'exlpoitation en pleine expension, ainsi qu'avec un outil de gestion de réseau, SNMP. Cette nouvelle expérience des réseau sera pour moi un atout certain dans ma future carrière, et, sur un plan plus personnel m'a donné une autre idée (moins scolaire) des applications de l'internet.

Ce projet, s'il n'était pas forcément d'une grande complexité, m'a toutefois souvent posé des problèmes dans la mesure où je ne suis pas un informaticien de formation. En effet, certaines solutions peuvent paraître évidentes à des spécialistes, alors que pour un néophite comme moi, cela demande une plus longue recherche.

Je voudrais profiter de cette conclusion pour remercier tout particulièrement Patrice KADIONIK, mon tuteur, qui m'a toujours épaulé et conseillé durant ce stage.

5. Glossaire

IANA :	Internet Assigned Numbers Authority
MIB :	Management Information Base
IP :	Internet Protocol
TCP :	Transmission Control Protocol
EGP :	Exterior Gateway Protocol
UDP :	User Datagram Protocol
API :	Application Programmer’s Interface
ASN.1 :	Abstract Syntax Notation 1
NMS :	Network Management Station
SNMP :	Simple Network Management Protocol

6. Bibliographie

Gestion des réseaux ouverts — Marshall Rose, InterEdition

Understanding SNMP MIBS — Perkins & McGinnis, Prentice Hall

http://linas.org/linux/NMS.html

http://www.moretonbay.com/coldfire/

http://www.guill.net/reseaux/Snmp.html

http://net-snmp.sourceforge.net

http://www.ensimag.imag.fr

http://www.enseirb.fr/~kadionik

http://www.csc.liv.ac.uk/~daves

http://motorola.com

7. Annexes

Fichier MIB ENSEIRB-MIB.txt

```
ENSEIRB-MIB DEFINITIONS ::= BEGIN

-- MIB of the ENSEIRB School of electrical Engineering

-- IMPORTS: Include definitions from other mibs here, which is always
-- the first item in a MIB file.
IMPORTS
      enterprises                           FROM SNMPv2-SMI,
      MODULE-IDENTITY                        FROM SNMPv2-SMI
      MODULE-COMPLIANCE, OBJECT-GROUP       FROM SNMPv2-CONF;

--
-- A brief description and update information about this mib.
--
enseirb MODULE-IDENTITY
    LAST-UPDATED "0104010000Z"            -- 01 Apr 2001, midnight
    ORGANIZATION "ENSEIRB"
    CONTACT-INFO "
                  Author:     Patrice Kadionik
                              ENSEIRB, School of Electrical Engineering
                  postal:     PO Box 99
                        33402 TALENCE CEDEX
                              FRANCE
                  email:      kadionik@enseirb.fr
                  phone:      +33-5-56-84-65-00
                 "
    DESCRIPTION  "MIB for uClinux remote control by SNMP
                 "
    ::= { enterprises 80 }

-- Define typical mib nodes, like where the objects are going to lie.
-- we'll prefix everything in this mib with ust (ucd snmp tutorial)
leds               OBJECT IDENTIFIER ::= {  enseirb 1 }
reserved           OBJECT IDENTIFIER ::= { enseirb 2 }

-- Define the sections of the mib them selves:

--
-- 4 LEDs connected to the PC parallel port
--
-- leds            OBJECT IDENTIFIER ::= { enseirbObjects 1 }

--
-- LED 0 connected to bit 0 of the PC parallel port
--
led0            OBJECT-TYPE
    SYNTAX      Integer32 (0..1)
```

```
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
      "Led 0 connected to bit 0 of the PC parallel port."
    DEFVAL { 0 }
    ::= { leds 1 }

led1            OBJECT-TYPE
    SYNTAX      Integer32 (0..1)
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
      "Led 1 connected to bit 1 of the PC parallel port."
    DEFVAL { 0 }
    ::= { leds 2 }

led2            OBJECT-TYPE
    SYNTAX      Integer32 (0..1)
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
      "Led 2 connected to bit 2 of the PC parallel port."
    DEFVAL { 0 }
    ::= { leds 3 }

led3            OBJECT-TYPE
    SYNTAX      Integer32 (0..1)
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
      "Led 3 connected to bit 3 of the PC parallel port."
    DEFVAL { 0 }
    ::= { leds 4 }

led4            OBJECT-TYPE
    SYNTAX      Integer32 (0..1)
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
      "Led 4 connected to bit 4 of the PC parallel port."
    DEFVAL { 0 }
    ::= { leds 5 }

led5            OBJECT-TYPE
    SYNTAX      Integer32 (0..1)
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
      "Led 5 connected to bit 5 of the PC parallel port."
    DEFVAL { 0 }
    ::= { leds 6 }

led6            OBJECT-TYPE
```

```
    SYNTAX      Integer32 (0..1)
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
      "Led 6 connected to bit 6 of the PC parallel port."
    DEFVAL { 0 }
    ::= { leds 7 }

led7            OBJECT-TYPE
    SYNTAX      Integer32 (0..1)
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
      "Led 7 connected to bit 7 of the PC parallel port."
    DEFVAL { 0 }
    ::= { leds 8 }
END
```

Fichier enseirb.h

```
/* This file was generated by mib2c and is intended for use as a mib module
  for the ucd-snmp snmpd agent. */


#ifndef _MIBGROUP_ENSEIRB_H
#define _MIBGROUP_ENSEIRB_H


/* we may use header_generic and header_simple_table from the util_funcs module */


config_require(util_funcs)


/* function prototypes */


void   init_enseirb(void);
FindVarMethod var_enseirb;

WriteMethod write_led0;
WriteMethod write_led1;
WriteMethod write_led2;
WriteMethod write_led3;
WriteMethod write_led4;
WriteMethod write_led5;
WriteMethod write_led6;
WriteMethod write_led7;



#endif /* _MIBGROUP_ENSEIRB_H */
```

Fichier enseirb.c

```
/* This file was generated by mib2c and is intended for use as a mib module
   for the ucd-snmp snmpd agent. */


#ifdef IN_UCD_SNMP_SOURCE
/* If we're compiling this file inside the ucd-snmp source tree */


/* This should always be included first before anything else */
#include <config.h>


/* minimal include directives */
#include "mibincl.h"
#include "util_funcs.h"


#else /* !IN_UCD_SNMP_SOURCE */


//#include <ucd-snmp/ucd-snmp-config.h>
//#include <ucd-snmp/ucd-snmp-includes.h>
//#include <ucd-snmp/ucd-snmp-agent-includes.h>


#endif /* !IN_UCD_SNMP_SOURCE */

#include <stdlib.h>
#include <unistd.h>
#include <asm/io.h>

#include "my_wrapper.h"                 //specifique au portage sous uClinux
#include "enseirb.h"


#define         OFF 0
#define         ON  1

#define         lp_base 0x378           //specifique a l'agent sous linux

static char lp_data ;                   //specifique a l'agent sous linux

static char led0 = OFF;
static char led1 = OFF;
static char led2 = OFF;
static char led3 = OFF;
static char led4 = OFF;
static char led5 = OFF;
```

```
static char led6 = OFF;
static char led7 = OFF;

short *PADAT = 0x10000244;              //specifique au portage sous uClinux




/*
 * enseirb_variables_oid:
 *   this is the top level oid that we want to register under.  This
 *   is essentially a prefix, with the suffix appearing in the
 *   variable below.
 */


oid enseirb_variables_oid[] = { 1,3,6,1,4,1,80 };


/*
 * variable2 enseirb_variables:
 *   this variable defines function callbacks and type return information
 *   for the enseirb mib section
 */


struct variable2 enseirb_variables[] = {
/*  magic number        , variable type , ro/rw , callback fn  , L, oidsuffix */
#define   LED0                1
  { LED0                , ASN_INTEGER   , RWRITE, var_enseirb, 2, { 1,1 } },
#define   LED1                2
  { LED1                , ASN_INTEGER   , RWRITE, var_enseirb, 2, { 1,2 } },
#define   LED2                3
  { LED2                , ASN_INTEGER   , RWRITE, var_enseirb, 2, { 1,3 } },
#define   LED3                4
  { LED3                , ASN_INTEGER   , RWRITE, var_enseirb, 2, { 1,4 } },
#define   LED4                5
  { LED4                , ASN_INTEGER   , RWRITE, var_enseirb, 2, { 1,5 } },
#define   LED5                6
  { LED5                , ASN_INTEGER   , RWRITE, var_enseirb, 2, { 1,6 } },
#define   LED6                7
  { LED6                , ASN_INTEGER   , RWRITE, var_enseirb, 2, { 1,7 } },
#define   LED7                8
  { LED7                , ASN_INTEGER   , RWRITE, var_enseirb, 2, { 1,8 } },

};
/*    (L = length of the oidsuffix) */


/*
 * init_enseirb():
 *   Initialization routine.  This is called when the agent starts up.
 *   At a minimum, registration of your variables should take place here.
```

```
 */
void init_enseirb(void) {


  /* register ourselves with the agent to handle our mib tree */
  REGISTER_MIB("enseirb", enseirb_variables, variable2,
               enseirb_variables_oid);


  /* place any other initialization junk you need here */
      ioperm(lp_base, 1, 1) ;                  //specifique a l'agent sous linux
      lpdata = 0 ;                             //specifique a l'agent sous linux
      outb (lpdata, lp_base) ;                 //specifique a l'agent sous linux
}


/*
 * var_enseirb():
 *   This function is called every time the agent gets a request for
 *   a scalar variable that might be found within your mib section
 *   registered above.  It is up to you to do the right thing and
 *   return the correct value.
 *     You should also correct the value of "var_len" if necessary.
 *
 *   Please see the documentation for more information about writing
 *   module extensions, and check out the examples in the examples
 *   and mibII directories.
 */
unsigned char *
var_enseirb(struct variable *vp,
                oid     *name,
                size_t  *length,
                int     exact,
                size_t  *var_len,
                WriteMethod **write_method)
{


  /* variables we may use later */
  static long long_ret;
  static unsigned char string[SPRINT_MAX_LEN];
  static oid objid[MAX_OID_LEN];
  static struct counter64 c64;


  if (header_generic(vp,name,length,exact,var_len,write_method)
                                  == MATCH_FAILED )
    return NULL;


  /*
   * this is where we do the value assignments for the mib results.
   */
```

```
  switch(vp->magic) {


    case LED0:
        *write_method = write_led0;
        long_ret = led0;
        return (unsigned char *) &long_ret;

    case LED1:
        *write_method = write_led1;
        long_ret = led1;
        return (unsigned char *) &long_ret;

    case LED2:
        *write_method = write_led2;
        long_ret = led2;
        return (unsigned char *) &long_ret;

    case LED3:
        *write_method = write_led3;
        long_ret = led3;
        return (unsigned char *) &long_ret;

    case LED4:
        *write_method = write_led4;
        long_ret = led4;
        return (unsigned char *) &long_ret;

    case LED5:
        *write_method = write_led5;
        long_ret = led5;
        return (unsigned char *) &long_ret;

    case LED6:
        *write_method = write_led6;
        long_ret = led6;
        return (unsigned char *) &long_ret;

    case LED7:
        *write_method = write_led7;
        long_ret = led7;
        return (unsigned char *) &long_ret;


    default:
      ERROR_MSG("");
  }
  return NULL;
}
```

```
int
write_led0(int      action,
            u_char   *var_val,
            u_char   var_val_type,
            size_t   var_val_len,
            u_char   *statP,
            oid      *name,
            size_t   name_len)
{
  static long *long_ret;
  int size;
  char tmp;


  switch ( action ) {
        case RESERVE1:
          if (var_val_type != ASN_INTEGER){
              fprintf(stderr, "write to led0 not ASN_INTEGER\n");
              return SNMP_ERR_WRONGTYPE;
          }
          if (var_val_len > sizeof(long_ret)){
              fprintf(stderr,"write to led0: bad length\n");
              return SNMP_ERR_WRONGLENGTH;
          }
          break;


        case RESERVE2:
          size = var_val_len;
          long_ret = (long *) var_val;


          break;


        case FREE:
             /* Release any resources that have been allocated */
          break;


        case ACTION:
             /* The variable has been stored in long_ret for
             you to use, and you have just been asked to do something with
             it.  Note that anything done here must be reversable in the UNDO case
*/


            lpdata = inb ( lp_base ) ;    //specifique a l'agent sous sous linux
            tmp = *long_ret << i ;
            if (tmp)
                  lpdata |= tmp ;
            else  {
```

```
                tmp = 1 << i ;
                lpdata &= (~tmp) ;
                }
            outb(lpdata, lp_base) ;


     if(*long_ret == 1)                 //specifique au portage sous uClinux
              *PADAT &= 0x007f;
       else
              *PADAT |= 0x0080;
       led0 = *long_ret;
         break;


       case UNDO:
            /* Back out any changes made in the ACTION case */
         break;


       case COMMIT:
            /* Things are working well, so it's now safe to make the change
            permanently.  Make sure that anything done here can't fail! */
         break;
  }
  return SNMP_ERR_NOERROR;
}




int
write_led1(int      action,
           u_char   *var_val,
           u_char   var_val_type,
           size_t   var_val_len,
           u_char   *statP,
           oid      *name,
           size_t   name_len)
{
  static long *long_ret;
  int size;
  char tmp;


  switch ( action ) {
       case RESERVE1:
         if (var_val_type != ASN_INTEGER){
             fprintf(stderr, "write to led1 not ASN_INTEGER\n");
             return SNMP_ERR_WRONGTYPE;
         }
         if (var_val_len > sizeof(long_ret)){
             fprintf(stderr,"write to led1: bad length\n");
             return SNMP_ERR_WRONGLENGTH;
```

```
          }
          break;


        case RESERVE2:
          size = var_val_len;
          long_ret = (long *) var_val;


          break;


        case FREE:
             /* Release any resources that have been allocated */
          break;


        case ACTION:
             /* The variable has been stored in long_ret for
             you to use, and you have just been asked to do something with
             it.  Note that anything done here must be reversable in the UNDO case
*/
        if(*long_ret == 1)
                *PADAT &= 0x00bf;
        else
                *PADAT |= 0x0040;
        led1 = *long_ret;
          break;


        case UNDO:
             /* Back out any changes made in the ACTION case */
          break;


        case COMMIT:
             /* Things are working well, so it's now safe to make the change
             permanently.  Make sure that anything done here can't fail! */
          break;
  }
  return SNMP_ERR_NOERROR;
}




int
write_led2(int      action,
            u_char   *var_val,
            u_char   var_val_type,
            size_t   var_val_len,
            u_char   *statP,
            oid      *name,
```

```
            size_t   name_len)
{
  static long *long_ret;
  int size;
  char tmp;


  switch ( action ) {
        case RESERVE1:
          if (var_val_type != ASN_INTEGER){
              fprintf(stderr, "write to led2 not ASN_INTEGER\n");
              return SNMP_ERR_WRONGTYPE;
          }
          if (var_val_len > sizeof(long_ret)){
              fprintf(stderr,"write to led2: bad length\n");
              return SNMP_ERR_WRONGLENGTH;
          }
          break;


        case RESERVE2:
          size = var_val_len;
          long_ret = (long *) var_val;


          break;


        case FREE:
             /* Release any resources that have been allocated */
          break;


        case ACTION:
             /* The variable has been stored in long_ret for
             you to use, and you have just been asked to do something with
             it.  Note that anything done here must be reversable in the UNDO case
*/
        if(*long_ret == 1)
                *PADAT &= 0x00df;
        else
                *PADAT |= 0x0020;
        led2 = *long_ret;
          break;


        case UNDO:
             /* Back out any changes made in the ACTION case */
          break;


        case COMMIT:
             /* Things are working well, so it's now safe to make the change
```

```
            permanently.  Make sure that anything done here can't fail! */
          break;
  }
  return SNMP_ERR_NOERROR;
}



int
write_led3(int      action,
           u_char   *var_val,
           u_char   var_val_type,
           size_t   var_val_len,
           u_char   *statP,
           oid      *name,
           size_t   name_len)
{
  static long *long_ret;
  int size;
  char tmp;


  switch ( action ) {
        case RESERVE1:
          if (var_val_type != ASN_INTEGER){
              fprintf(stderr, "write to led3 not ASN_INTEGER\n");
              return SNMP_ERR_WRONGTYPE;
          }
          if (var_val_len > sizeof(long_ret)){
              fprintf(stderr,"write to led3: bad length\n");
              return SNMP_ERR_WRONGLENGTH;
          }
          break;


        case RESERVE2:
          size = var_val_len;
          long_ret = (long *) var_val;


          break;


        case FREE:
             /* Release any resources that have been allocated */
          break;


        case ACTION:
             /* The variable has been stored in long_ret for
             you to use, and you have just been asked to do something with
```

```
            it.  Note that anything done here must be reversable in the UNDO case
*/
        if(*long_ret == 1)
                *PADAT &= 0x00ef;
        else
                *PADAT |= 0x0010;
        led3 = *long_ret;
          break;


        case UNDO:
             /* Back out any changes made in the ACTION case */
          break;


        case COMMIT:
             /* Things are working well, so it's now safe to make the change
             permanently.  Make sure that anything done here can't fail! */
          break;
  }
  return SNMP_ERR_NOERROR;
}




int
write_led4(int      action,
            u_char   *var_val,
            u_char   var_val_type,
            size_t   var_val_len,
            u_char   *statP,
            oid      *name,
            size_t   name_len)
{
  static long *long_ret;
  int size;
  char tmp;


  switch ( action ) {
        case RESERVE1:
          if (var_val_type != ASN_INTEGER){
              fprintf(stderr, "write to led4 not ASN_INTEGER\n");
              return SNMP_ERR_WRONGTYPE;
          }
          if (var_val_len > sizeof(long_ret)){
              fprintf(stderr,"write to led4: bad length\n");
              return SNMP_ERR_WRONGLENGTH;
          }
          break;
```

```
        case RESERVE2:
          size = var_val_len;
          long_ret = (long *) var_val;


          break;


        case FREE:
             /* Release any resources that have been allocated */
          break;


        case ACTION:
             /* The variable has been stored in long_ret for
             you to use, and you have just been asked to do something with
             it.  Note that anything done here must be reversable in the UNDO case
*/
        led4 = *long_ret;
          break;


        case UNDO:
             /* Back out any changes made in the ACTION case */
          break;


        case COMMIT:
             /* Things are working well, so it's now safe to make the change
             permanently.  Make sure that anything done here can't fail! */
          break;
  }
  return SNMP_ERR_NOERROR;
}




int
write_led5(int      action,
            u_char   *var_val,
            u_char   var_val_type,
            size_t   var_val_len,
            u_char   *statP,
            oid      *name,
            size_t   name_len)
{
  static long *long_ret;
  int size;
  char tmp;


  switch ( action ) {
```

```
        case RESERVE1:
          if (var_val_type != ASN_INTEGER){
              fprintf(stderr, "write to led5 not ASN_INTEGER\n");
              return SNMP_ERR_WRONGTYPE;
          }
          if (var_val_len > sizeof(long_ret)){
              fprintf(stderr,"write to led5: bad length\n");
              return SNMP_ERR_WRONGLENGTH;
          }
          break;


        case RESERVE2:
          size = var_val_len;
          long_ret = (long *) var_val;


          break;


        case FREE:
             /* Release any resources that have been allocated */
          break;


        case ACTION:
             /* The variable has been stored in long_ret for
             you to use, and you have just been asked to do something with
             it.  Note that anything done here must be reversable in the UNDO case
*/
        led5 = *long_ret;
          break;


        case UNDO:
             /* Back out any changes made in the ACTION case */
          break;


        case COMMIT:
             /* Things are working well, so it's now safe to make the change
             permanently.  Make sure that anything done here can't fail! */
          break;
  }
  return SNMP_ERR_NOERROR;
}




int
write_led6(int      action,
            u_char   *var_val,
```

```
            u_char   var_val_type,
            size_t   var_val_len,
            u_char   *statP,
            oid      *name,
            size_t   name_len)
{
  static long *long_ret;
  int size;
  char tmp;


  switch ( action ) {
        case RESERVE1:
          if (var_val_type != ASN_INTEGER){
              fprintf(stderr, "write to led6 not ASN_INTEGER\n");
              return SNMP_ERR_WRONGTYPE;
          }
          if (var_val_len > sizeof(long_ret)){
              fprintf(stderr,"write to led6: bad length\n");
              return SNMP_ERR_WRONGLENGTH;
          }
          break;


        case RESERVE2:
          size = var_val_len;
          long_ret = (long *) var_val;


          break;


        case FREE:
             /* Release any resources that have been allocated */
          break;


        case ACTION:
             /* The variable has been stored in long_ret for
             you to use, and you have just been asked to do something with
             it.  Note that anything done here must be reversable in the UNDO case
*/
        led6 = *long_ret;
          break;


        case UNDO:
             /* Back out any changes made in the ACTION case */
          break;


        case COMMIT:
             /* Things are working well, so it's now safe to make the change
```

```
            permanently.  Make sure that anything done here can't fail! */
          break;
  }
  return SNMP_ERR_NOERROR;
}



int
write_led7(int      action,
           u_char   *var_val,
           u_char   var_val_type,
           size_t   var_val_len,
           u_char   *statP,
           oid      *name,
           size_t   name_len)
{
  static long *long_ret;
  int size;
  char tmp;


  switch ( action ) {
        case RESERVE1:
          if (var_val_type != ASN_INTEGER){
              fprintf(stderr, "write to led7 not ASN_INTEGER\n");
              return SNMP_ERR_WRONGTYPE;
          }
          if (var_val_len > sizeof(long_ret)){
              fprintf(stderr,"write to led7: bad length\n");
              return SNMP_ERR_WRONGLENGTH;
          }
          break;


        case RESERVE2:
          size = var_val_len;
          long_ret = (long *) var_val;


          break;


        case FREE:
             /* Release any resources that have been allocated */
          break;


        case ACTION:
             /* The variable has been stored in long_ret for
             you to use, and you have just been asked to do something with
```

```
             it.  Note that anything done here must be reversable in the UNDO case
*/
        led7 = *long_ret;
          break;


        case UNDO:
             /* Back out any changes made in the ACTION case */
          break;


        case COMMIT:
             /* Things are working well, so it's now safe to make the change
             permanently.  Make sure that anything done here can't fail! */
          break;
  }
  return SNMP_ERR_NOERROR;
}
```

Makefile du répertoire mypatch-4.1.2

```
# Generated automatically from Makefile.top:Makefile.in by configure.
#
# Minimum environment and virtual path setup
#
SHELL       = /bin/sh
srcdir            = .
top_srcdir  = .


#
# Paths
#
prefix            = /usr/local
exec_prefix = /usr/local
bindir            = ${exec_prefix}/bin
sbindir           = ${exec_prefix}/sbin
libdir            = ${exec_prefix}/lib
datadir           = ${prefix}/share
includedir  = ${prefix}/include/ucd-snmp
mandir            = ${prefix}/man
man1dir           = $(mandir)/man1
man3dir           = $(mandir)/man3
man5dir           = $(mandir)/man5
man8dir           = $(mandir)/man8
snmplibdir  = $(datadir)/snmp
mibdir            = $(snmplibdir)/mibs
persistentdir     = /tmp


#
# Programs
#
INSTALL           = /usr/bin/install -c
SED         = sed
RANLIB            = ranlib
LN_S        = ln -s

#
# Compiler arguments
#
CFLAGS            += -g
EXTRACPPFLAGS     = -x c
# LDFLAGS         =

#
# Shared library commands (or not)
#
SHLIB_CFLAGS            =
SHLIB_EXTENSION         = a
```

```
SHLIB_VERSION           =
SHLIB_LDCONFIG_CMD      = :
SHLIB_LD_CMD            = ar cr
SHLIB_LD_LIBS           =
CC_RUNTIME_ARG          =

#
# Makefile.in (at the root of ucd-snmp)
#

TARG        =     bin/snmpget bin/snmpgetnext bin/snmpset  \
                  bin/snmptranslate bin/snmpwalk bin/snmpbulkwalk \
                  bin/snmptest bin/snmptrapd bin/snmpnetstat bin/snmpd

SUBDIRS           = snmplib agent
# SUBDIRS         = snmplib agent apps local ov man
INSTALLDIRS = snmplib agent
TESTDIRS    = testing

CPP         = $(CC) -E                                  \
                  -I$(srcdir)/agent/mibgroup -I. -I$(srcdir)     \
                  -DDONT_INC_STRUCTS -DBINDIR=$(bindir)          \
                  $(EXTRACPPFLAGS) $(INCS)


CPP = $(CC) -E -I$(srcdir)/agent/mibgroup -I. -I$(srcdir) -DDONT_INC_STRUCTS -
DBINDIR=$(bindir) $(EXTRACPPFLAGS) $(INCS)

INSTALLHEADERS=ucd-snmp-config.h $(srcdir)/version.h

all:  sedscript EXAMPLE.conf ucd-snmp-config.h subdirs

test:   all
      ( cd testing; $(MAKE) test )

sedscript: sedscript.in config.h $(srcdir)/agent/mibgroup/mibdefs.h
      $(CPP) -DPREFIX=$(prefix) -DLIBDIR=$(libdir) -DDATADIR=$(datadir)
$(srcdir)/sedscript.in | egrep '^s[/#]' | sed 's/REMOVEME//g;s# */#/#g;s/
*#/#/g;s#/ *#/#g;s/# g/#g/;' > sedscript


ucd-snmp-config.h: config.h
      @egrep -v
"IN_UCD_SNMP_SOURCE|SYSTEM_INCLUDE|MACHINE_INCLUDE|mib_module_config" config.h >
ucd-snmp-config.h

subdirs:
      for i in $(SUBDIRS) ; do      \
        ( cd $$i ; $(MAKE) ) ;           \
      done


test: all testdirs
```

```
testdirs:
      for i in $(TESTDIRS) ; do     \
        ( cd $$i ; $(MAKE) ) ;           \
      done


distall: ${srcdir}/configure ${srcdir}/config.h


install:    all installdirs
      for i in $(INSTALLDIRS) ; do  \
        ( cd $$i ; $(MAKE) install ) ;   \
      done
      @for i in $(INSTALLHEADERS) ; do \
            $(INSTALL) $$i $(includedir) ; \
            echo "install:  installed $$i in $(includedir)";\
      done

installdirs:
      @$(SHELL) $(srcdir)/mkinstalldirs $(snmplibdir) $(mibdir) $(includedir)
      @-$(SHELL) $(srcdir)/mkinstalldirs $(persistentdir)

apps/snmpget apps/snmpgetnext apps/snmpset apps/snmptranslate apps/snmpwalk
apps/snmpbulkwalk apps/snmptest apps/snmptrapd apps/snmpnetstat/snmpnetstat
agent/snmpd: makeall


depend:
      cd snmplib; $(MAKE) depend
      cd apps; $(MAKE) depend
      cd agent; $(MAKE) depend
      cd testing; $(MAKE) depend

nosysdepend:
      cd snmplib; $(MAKE) nosysdepend
      cd apps; $(MAKE) nosysdepend
      cd agent; $(MAKE) nosysdepend
      cd testing; $(MAKE) nosysdepend

makefileindepend:
      cd snmplib; $(MAKE) makefileindepend
      cd apps; $(MAKE) makefileindepend
      cd agent; $(MAKE) makefileindepend
      cd testing; $(MAKE) makefileindepend


clean:
      for i in $(SUBDIRS) $(TESTDIRS); do \
        ( cd $$i ; $(MAKE) clean ) ; \
      done
      rm -f EXAMPLE.conf sedscript ucd-snmp.txt
```

```
distclean: clean configclean

configclean:
	rm -f config.cache config.log config.h
	rm -f Makefile snmplib/Makefile                    \
	      agent/Makefile agent/mibgroup/Makefile       \
	      apps/Makefile  apps/snmpnetstat/Makefile \
	      man/Makefile mibs/Makefile ov/Makefile       \
	      local/Makefile testing/Makefile
	rm -f agent/dlmods/Makefile
	rm -f mibs/.index
	rm -f mib_module_config.h ucd-snmp-config.h    \
	      agent/mibgroup/mib_module_includes.h     \
	      agent/mibgroup/mib_module_inits.h  \
	      agent/mibgroup/mib_module_shutdown.h     \
	      agent/mibgroup/mib_module_dot_conf.h
	rm -f *.core


configure: configure.in aclocal.m4
	cd ${srcdir} && autoconf
	echo "Please run configure now."
	sh -c exit 2


# autoheader might not change config.h.in, so touch a stamp file.
#
config.h.in: stamp-h.in
stamp-h.in: configure.in acconfig.h
	cd ${srcdir} && autoheader
	echo timestamp > ${srcdir}/stamp-h.in

config.h: stamp-h
stamp-h: config.h.in
	CONFIG_FILES=
	echo timestamp > ${srcdir}/stamp-h

touchit:
	touch configure config.h.in
	touch config.h
	touch stamp-h stamp-h.in
	touch Makefile

Makefile: Makefile.in
	CONFIG_HEADERS=

#config.status: configure
#	./config.status --recheck

EXAMPLE.conf: sedscript EXAMPLE.conf.def
	$(SED) -f sedscript $(srcdir)/EXAMPLE.conf.def > EXAMPLE.conf

TAGS:
```

```
	find $(srcdir) -name '*.[ch]' -print | etags -

version:
	@if test "x$(VERSION)" = "x"; then \
	  echo "you need to supply a VERSION string."; \
	  exit 2; \
	fi
	agent/mibgroup/versiontag $(VERSION) reverse

tag:
	agent/mibgroup/versiontag $(VERSION) tag

tar:
	agent/mibgroup/versiontag $(VERSION) tar

dist: version tag tar

FAQ.html:
	local/FAQ2HTML FAQ

# If we're installing the UCD SMNP daemon, add it to /etc/inittab.  We also
# must provide a default configuration file.
build-romfs:
#	echo "snmp:unknown:/bin/snmpd -s -P /var/log/snmpd.pid" >>$(ROMFS)/etc/inittab
#	echo "snmp:unknown:/bin/snmpd -s -P /var/log/snmpd.pid -c /etc/snmpd.conf"
>>$(ROMFS)/etc/inittab
#	cp ucdsnmpd-default $(ROMFS)/etc/snmpd.conf
#	cp ucdsnmp-default $(ROMFS)/etc/snmp.conf
	cp mysnmpd.conf $(ROMFS)/etc/config/snmpd.conf
	cp mysnmp.conf $(ROMFS)/etc/config/snmp.conf
```