République Algérienne Démocratique Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

ECOLE NATIONALE POLYTECHNIQUE

Département d'Electronique

Projet de fin d'études pour l'obtention du diplôme

d'Ingénieur d'Etat en Electronique

THEME

Développement d'une distribution Linux fonctionnelle

Soutenu publiquement en juin 2008

Proposé par :

Mr R.SADOUN

Réalisé par :

Melle Nacera BOUMAZA Melle Nesrine CHAOUCHE

Dédicaces

Je dédie ce modeste travail à ma mère, mon père, ma sœur et mes frères qui m'ont soutenue sans relâche, dans toutes les circonstances, tout au long de mon parcours d'études.

Je dédie, ce travail à ceux qui mon aidé, spécialement à Samir pour son soutien.

Je dédie, aussi ce travail à mon binôme Nacera.

Je dédie ce travail à tous mes ami(e)s pour leur soutien tout au long de mon parcours.

Enfin je dédie ce travail à ma chère et très spéciale promotion.

Nesrine

A ma chère mère et mon cher père, qui m'ont beaucoup soutenu tout le long de mon parcours, pour leurs sacrifices et leur amour, sans limite

Que dieu me les garde

A mes chers frères et sœurs, A toute ma famille

A mon binôme et sœur Nesrine

Je dédie aussi ce travail à ceux qui m'ont aidée surtout Samir, Hakim, Krimo, Rabah et spécialement SELMANE.

A tous ceux qui me sont chers

A HICHEM, AYMEN QUE J'ADORE

Enfin je dédie ce travail à ma chère et très spéciale promotion.

Nacera

Remerciements

Nous tenons, avant tout, à remercier notre DIEU, tout clément, tout puissant, de nous avoir donné la force de réaliser notre travail.

Nos remerciements vont exceptionnellement à Monsieur R. SADOUN, chargé de cours à l'Ecole Nationale Polytechnique d'Alger, pour son aide, son suivi, ses conseils et directives et pour son dévouement.

Nous tenons à remercier Monsieur M^{ed}.MEHENNI, Professeur à l'Ecole Nationale Polytechnique d'Alger, d'avoir accepté de présider le jury de notre mémoire.

Nos remerciements vont aussi à Monsieur ABDELLOUEL, Chargé de cours à l'Ecole Nationale Polytechnique d'Alger, d'avoir bien voulu accepter d'examiner notre travail.

Nous remercions tous les enseignants de l'Ecole Nationale Polytechnique d'Alger, spécialement ceux du département des Sciences Fondamentales et d'Electronique, pour leur apport en savoir.

Nos remerciements, vont au personnel du centre de calcul de l'Ecole et à toute personne dévouée au service de l'Ecole Nationale Polytechnique.

Enfin, nos remerciements vont à toute personne ayant contribué, de prés ou de loin, à réaliser ce travail.

TABLE DES MATIERES

ملخص

للانترنت مزايا كثيرة لكن يملك أيضا مشكلة أمنية, و من مصلحتنا حماية أنضمتنا من المخاطر الخارجية, ومن ثم فإن الهدف من مشروعنا ينطوي على إنجاز توزيعة لينوكس وظيفية شخصية, تلعب دور جدار ناري, و قد تحقق هذا الهدف اعتمادا على منهجية و نواة لينوكس, حزم و أدوات متخصصة.

الكلمات الأساسية

نظام التشغيل, لينوكس, توزيعة, جدار ناري, shorewall , iptables, برنامج , شفرة المصدر.

Abstract

Connecting to the Internet has many advantages but it also causes a security problem. Also, we have an interest in protecting our systems against these risks from outside hence The purpose of our project which involves the conduct of a Linux distribution functional, fully personalized playing the role of a firewall. This goal was achieved on the basis of a methodology and a Linux kernel, packages and special tools.

Keywords

Operating system, Linux, Distribution, Firewall, shorewall, iptables, script, source code.

Résumé

Se connecter à Internet présente beaucoup d'avantages mais elle cause aussi des failles de sécurité. Aussi, nous avons intérêt à protéger nos systèmes contre ces risques venant de l'extérieur d'où La finalité de notre projet qui consiste en la réalisation d'une distribution Linux fonctionnelle, totalement personnalisée jouant le rôle d'un pare-feu. Ce but a été atteint en partant d'une méthodologie et d'un noyau Linux, de paquets et d'outils spécifiques.

Mots clés

Système d'exploitation, Linux, Distribution, Firewall, iptables, Shorewall, script, code source.

INTRODUCTION GENERALE

Le système Linux est devenu en quelques années un phénomène de société. Il est souvent considéré comme le fer de lance d'une nouvelle tendance de l'informatique. Il reste très répandu dans les parcs des serveurs où il accapare des parts non négligeables. On constate de plus en plus son émergence dans le domaine des postes clients. Et il faut noter aussi que les systèmes embarqués lui font appel d'une manière remarquée.

Avec l'augmentation rapide de l'utilisation d'Internet, la nécessité d'implanter un environnement réseau sécurisé devient toujours plus pressant, afin de pouvoir garantir la disponibilité des données. Aujourd'hui il faut prendre certaines précautions avant de se raccorder à l'Internet. Pour cela l'utilisation d'un firewall est indispensable. Linux se positionne comme Firewall idéal. La grande disponibilité, la flexibilité et le haut degré de sécurisation de Linux en font un concurrent redoutable parmi les Firewalls à disposition dans le commerce.

Notre mémoire est organisée comme suit :

Le premier chapitre est dédié à la présentation des systèmes d'exploitation est plus précisément le système d'exploitation Linux, nous détaillerons pour ce dernier l'architecture de son noyau et la procédure de démarrage. Nous terminons ce chapitre en explicitant les outils qui nous permettent de développer une application sous Linux.

Le deuxième chapitre traite la construction d'une distribution Linux basique en suivant une démarche bien définie LFS. Nous donnons les étapes à suivre pour parvenir à la réalisation de notre distribution tout en utilisant les outils de développement exposés dans le chapitre précédant. Puis nous orientons notre distribution vers la sécurité du réseau en suivant la démarche BLFS. Pour parvenir à le faire nous installons tous les outils et protocoles nous permettant de se connecter via Internet en toute sécurité.

Dans le troisième chapitre nous faisons en sorte pour que notre distribution fasse du filtrage réseau c'est-à-dire qu'elle jouera le rôle d'un pare-feu. Pour cela nous commençons ce chapitre en expliquant le fonctionnement du firewall puis nous présentons deux manières pour

configurer le firewall soit en utilisant iptables ou shorewall. Nous optons pour shorewall puisque ce dernier n’est qu’une abstraction d’iptables.

Nous terminons notre projet de fin d’études par une conclusion, et un ensemble de perspectives.

CHAPITRE I

GENERALITES

I.1 INTRODUCTION

Pour atteindre notre but précédemment cité, on va commencer notre mémoire par présenter les systèmes d'exploitation, en les classifiant et en donnant leurs concepts de base, ainsi qu'un simple aperçu historique pour bien comprendre leurs évolutions dans le temps.

On va étudier ensuite Linux en tant qu'un système d'exploitation en détaillant l'architecture de son noyau et sa procédure de démarrage.

Après, on clôture ce premier chapitre par les outils permettant le développement d'une application sous Linux.

I.2 LES SYSTEMES D'EXPLOITATION

Le système d'exploitation a été conçu dans le but de dissimuler la complexité de l'infrastructure des ordinateurs donc l'idée était d'enrober le matériel par une couche logicielle gérant l'ensemble du système. Donc présenter à l'utilisateur une machine virtuelle plus facile à comprendre et à exploiter.

I.2.1 HISTORIQUE

I.2.1.1 PREMIERE GENERATION (1950-1960) "Exploitation porte ouverte"

Matériel à base de tubes : moins fiable, encombrant.

Mode d'exploitation : l'utilisateur écrit un programme en langage machine, il réserve la machine pour une durée déterminée, puis il rentre son programme en mémoire "à la main et il l'exécute immédiatement.

I.2.1.2 DEUXIEME GENERATION (1958 – 1968) "Traitement par lots"

Amélioration du matériel :

- transistors, circuits imprimés et circuits intégrés
- apparition et diversification des périphériques
- électromécaniques
- lecteurs de cartes perforées
- lecteurs/perforateurs de rubans
- imprimantes, télétypes
- cartes, tambours et bandes magnétiques

Mode d'exploitation : Travaux groupés en lots par un opérateur, Chargement de l'utilitaire (ou compilateur) nécessaire à un lot puis Chargement du lot sur le lecteur de cartes, Lancement de l'exécution du lot, Enchaînement des travaux.

I.2.1.3 TROISIEME GENERATION (1960-1970) - Multiprogrammation puis Temps partagé

Amélioration du matériel

• Processeurs spécialisés pour les entrées/sorties
• Accroissement des possibilités des machines et des périphériques.
• Protection mémoire par zones
• Pagination ou segmentation

Entrées/sorties en mode différé (techniques de minimisation des attentes dues aux E/S)

• Tamponnement des E/S
• Spool system

I.2.1.4 MULTIPROGRAMMATION ET TEMPS PARTAGE (1965-1980)

Multiprogrammation : Plusieurs activités progressent en parallèle

• Une activité garde le CPU jusqu'à la prochaine demande d'E/S
• Traitement des fins d'E/S du processeur grâce aux interruptions

Avantages :

- bonne gestion des ressources (CPU, mémoire, périph.)
- temps de réponse correct pour les travaux courts

Inconvénients

- complexité du matériel et du système d'exploitation

• partage et protection des ressources.

Temps partagé :

Objectif : retour à l'interactivité de l'exploitation porte ouverte

Solution

-Travail à partir d'un terminal interactif

- Refonte totale de l'interface utilisateur/système *multiprogrammation avec mécanisme garantissant un temps de réponse acceptable (quelques secondes) :*

- Allocation du processeur aux usagers à tour de rôle et pour un "quantum" de temps à chaque fois la fin du quantum est signalé par une interruption horloge

I.2.1.5 QUATRIEME GENERATION (1970- maintenant) Réseaux / Système répartis

Amélioration du matériel

• Apparition des composants à haute intégration (1969 premier microprocesseur)

• Développement des matériels et techniques de transmission de données

Intégration poussée des fonctions de communication dans les systèmes

• Réseaux d'ordinateurs hétérogènes faiblement couplés (OSI, INTERNET)

• Réseaux locaux : interconnexion de plusieurs ordinateurs au moyen d'une voie à haut débit (10/100 Mégabits/sec) ETHERNET 1975.

I.2.2 ORGANISATION DES SYSTEMES D'EXPLOITATION

Le système d'exploitation est une application composée du :

- **Noyau :** c'est l'ensemble de fonctions participant directement à l'exécution de logiciels sur le matériel et permettant éventuellement le partage et l'allocation de celui-ci. Il présente les activités telle que : chargement et lancement des programmes, gestion des processeurs, gestion des périphériques, gestion des entrées/sorties, gestion de mémoire principale et secondaire…

- **Outils :** C'est l'ensemble des logiciels de manipulation de collections de données. Ils n'interagissent pas directement avec les logiciels d'application. On trouve : éditeurs de textes**,** compilateurs, éditeurs de liens**,** recherche dans un fichier…etc.

- **Services :** ensemble de logiciels facilitant le travail du programmeur ou de l'usager mais facultatif pour cette fonction. Ils peuvent interagir avec les logiciels d'application. On cite: gestionnaires de données, gestionnaires de fenêtres, gestionnaires de communications...etc

La figure suivante illustre le modèle concentrique d'un système d'exploitation.

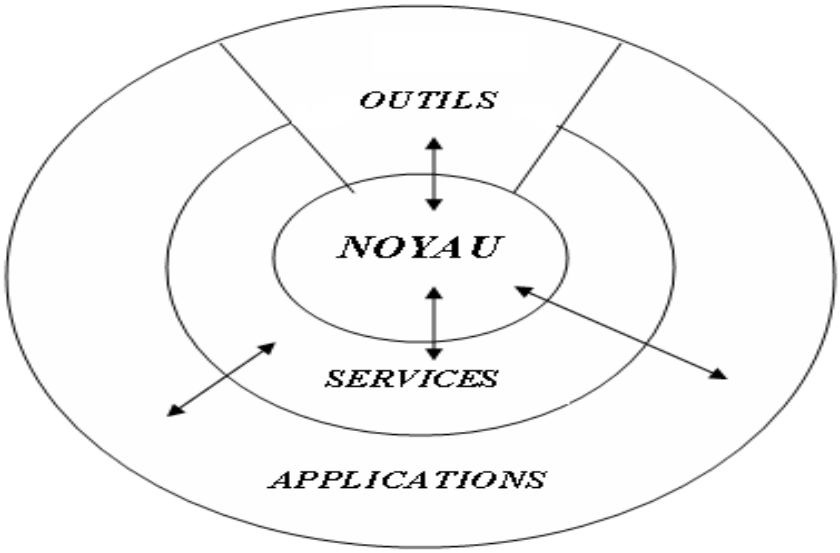

Figure I.1: *Modèle concentrique d'un système d'exploitation.*

I.2.3 CONCEPTS DE BASE DES SYSTEMES D'EXPLOITAION

I.2.3.1 COMMUNICATION INTERNE

La position du système d'exploitation entre le matériel et les applications induit un schéma de communication particulier. Les applications envoient au système des appels ***explicites*** (demande d'un service, lecture d'un fichier) ou ***implicites*** (notification de la fin d'un travail). Ces appels sont généralement *synchrones* (retour d'un appel immédiatement exécuté). Plus rarement, ils sont asynchrones (signal de suspension d'un travail, activation d'un travail précédemment bloqué).

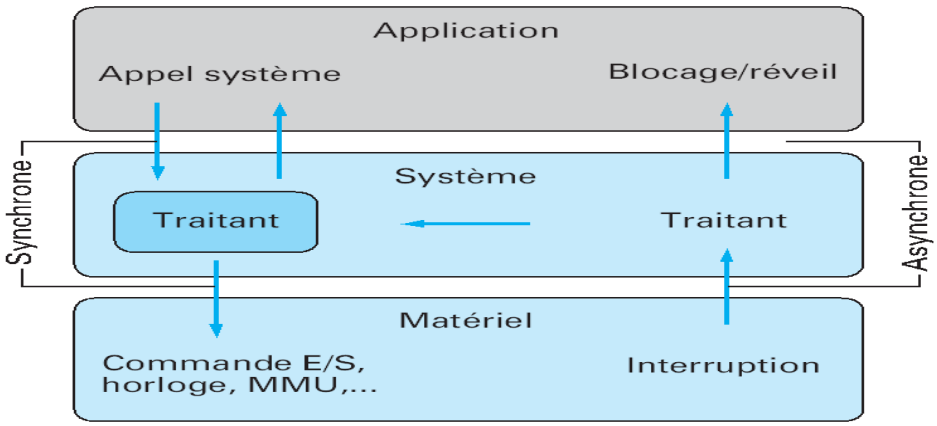

Figure I.2 *Communication interne dans un système d'exploitation.*

La nécessité d'organiser la gestion de multiples activités concurrentes a motivé l'introduction de la notion de **processus**.

I.2.3.2 GESTION DES ACTIVITES PARALLELES

La gestion d'activités parallèles coordonnées est l'une des fonctions principales d'un système d'exploitation. Les concepts et outils du parallélisme (processus, exclusion mutuelle, synchronisation et communication) ont été développés à cet effet. D'où la nécessité de définir la notion de « Synchronisation » qui est la mise en œuvre des relations entre processus en vue de respecter les contraintes de coopération. On va examiner la forme la plus simple de synchronisation qui est *: l'exclusion mutuelle.*

I.2.4 CLASSIFICATION DES SYSTEMES D'EXPLOITATION

I.2.4.1 SYSTEME MONO UTILISATEUR

Les systèmes mono utilisateur offrent à un moment donné une seule machine virtuelle à un seul ordinateur. Ceci concerne les ordinateurs dédiés à une seule fonction. Le but essentiel est de fournir un langage de commande facile à utiliser, un système de gestion de fichier simple et des facilités d'entrées/sorties adaptés au terminal et au disque.

I.2.4.2 LES SYSTEMES TRANSACTIONNELS

Ces systèmes sont caractérisés par des **data Bases** mises à jour très fréquemment. Le système d'exploitation doit prendre en charge le problème de les maintenir à jour afin que l'utilisateur ait l'impression d'être seul à disposer du système.

I.2.4.3 LES SYSTEMES GENERALISTES

Ces systèmes sont conçus pour supporter un flot continu de travaux assemblés en « JOB » effectuant des traitements spécifiques. La mise à disposition et le contrôle des périphériques, conjugués à l'organisation du flot de travail sont les fonctions essentielles d'un système d'exploitation généraliste .on peut les classer en :

- **Systèmes batch :** Leur caractéristique principale réside dans l'impossibilité, pour l'utilisateur, d'agir sur le déroulement du job après qu'il ait été introduit sur l'ordinateur.
- **Systèmes multi accès :** ils partagent les ressources de traitement entre les différents jobs, pour que chaque utilisateur ait l'impression de disposer de la machine pour lui seul.

I.3 LINUX

I.3.1 SYSTEME D'EXPLOITATION LINUX

Linux a bien évolué depuis sa création, qu'il répond à bon nombre de critères industriels (32 bits, multiutilisateur, multitâche, multiplateforme, multiprocesseur, rapidité, sécurité, stabilité, convivialité) et que son installation et son utilisation sont grandement simplifiées. Cependant, il est indéniable que ce système hérite naturellement des avantages et inconvénients d'Unix et qu'il subsiste quelques phases délicates dans la mise au point (tuning) d'une machine Linux.

Linux est donc une solution de plus en plus convaincante qui nécessite non pas un esprit «bidouilleur » comme à son origine mais une réelle réflexion de stratégie informatique. Cette dernière ne peut s'appuyer que sur une démarche d'ingénieur qui consiste à cibler précisément les besoins, mettre en correspondance ces derniers avec les spécifications de Linux et valider les choix par des essais avant toute mise en production. La multiplication d'exemples d'intégration de Linux en entreprise ne fait que conforter cette idée.

I.3.2 NOYAU LINUX

Dans ce qui suit nous allons présenter l'architecture fondamentale du noyau Linux et de ses composants. On va donner un aperçu tant des principaux aspects du noyau que des formes d'activités qu'il comporte, de la gestion de mémoire, des pilotes de périphériques, de l'horloge des modules, etc.

La figure suivante montre la structure du noyau Linux. On peut la subdiviser en cinq sections distinctes dont chacune résume une fonctionnalité de façon précise et fournit celles des autres composants du noyau.

- ***Gestion des processus :*** cette section sert à établir et mettre fin à des processus et d'autres activités du noyau (interruption logicielles, tasklets, etc.) de manière responsable. C'est en outre ici qu'a lieu la communication interprocessus. L'Ordonnanceur est l'élément principal de la gestion des processus .Son rôle consiste à gérer les processus actifs, en attente ou bloqués et à veiller à ce que tous les processus d'application obtiennent une fraction de temps machine de la part des processeurs.

- ***Gestion de la mémoire :*** la mémoire d'un ordinateur est une des ressources les plus importantes. La puissance d'une machine dépend en grande partie de la mémoire centrale dont elle est dotée .De plus la gestion de la mémoire est responsable du fait que tous processus obtient sa propre région de mémoire qu'il s'agit de protéger avant l'accès à d'autre processus.

- ***Systèmes de fichiers :*** le système de fichiers joue un rôle capital sous linux. Contrairement à d'autres systèmes d'exploitation (comme Windows NT), pratiquement tout sous linux se déroule sur l'interface de fichiers.

- ***Pilote de périphériques :*** dans tous les systèmes d'exploitation, ils font abstraction du matériel sous-jacent et on autorise l'accès à l'utilisateur. Le concept de module présenté à la section donne à linux en dépit du noyau monolithique, la possibilité d'ajouter de nouveaux pilotes de périphériques au système actuel ou de retirer ceux qui ne sont plus utilisés.

- ***Réseau :*** le système d'exploitation doit gérer l'infrastructure de réseau car certaines opérations afférentes ne peuvent être assignées à aucun processus spécifiques, comme le

traitement d'un paquet entrant. Les paquets entrants sont des évènements asynchrones, ils doivent être assemblés, identifiés et retransmis avant qu'un processus ne puisse les traiter. Le noyau est donc chargé de remettre les paquets de données par l'intermédiaire d'interfaces de programmes et de réseau.

Le noyau Linux est basé sur une architecture monolithique, toutes les fonctionnalités sont concentrées dans le noyau .Il en résulte que le noyau Linux est volumineux mais le fonctionnement de ce dernier est plus efficace car il y a moins d'appels système et le système accède au ressources à partir du noyau.

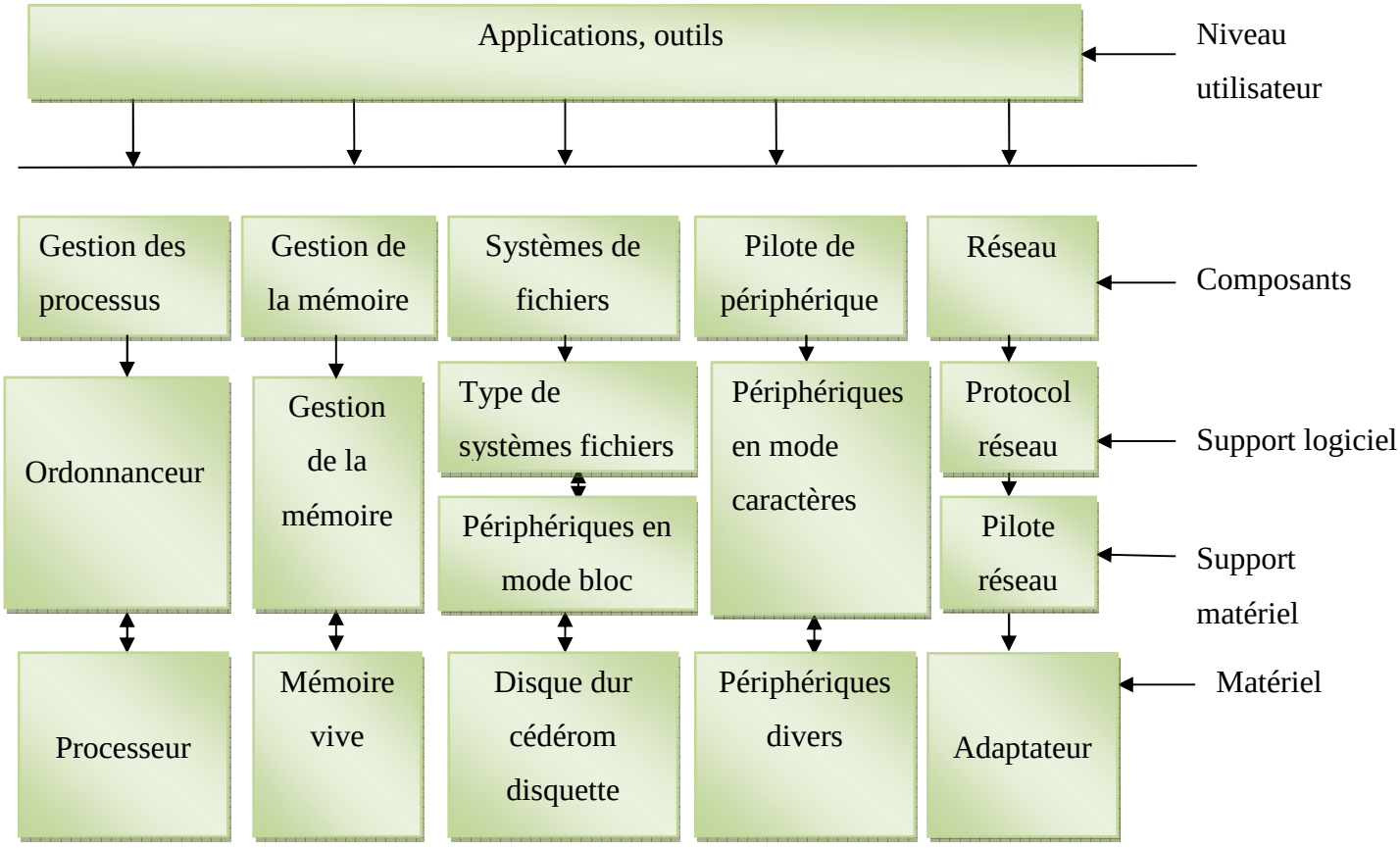

Figure II.2 *Architecture du noyau Linux.*

.

I.3.3.1 QU'EST CE QU'UNE DISTRIBUTION ?

Un système Linux est composé de nombreux éléments pour la plus part issus du monde du libre, chaque utilisateur pouvant choisir tel ou tel élément pour remplir une fonction du système.

Une distribution est donc composée de différents logiciels qui forment ensemble un système d'exploitation complet :

- Le noyau Linux.
- Les pilotes, Les bibliothèques.
- Les utilitaires d'installation et de post d'installation. Et un grand nombre de logiciels.

Chaque distribution est conçue dans un but précis, qu'il s'agisse d'optimisation pour un usage spécifique ou d'un accent porté sur l'ergonomie, la sécurité ou encore la rapidité du système.

I.3.3.2 ARCHITECTURE LOGICIELLE D'UNE DISTRIBUTION

Ce qui fait l'intérêt d'une distribution est l'exploitation du concept de couche d'abstraction. Comme nous pouvons le voir sur le schéma, les parties qui composent la distribution sont distinctes. Nous pouvons donc, par exemple, changer le noyau sans changer les logiciels et donc porter plus facilement la distribution sur une autre architecture matérielle.

Dans la figure suivante nous avons donné une illustration des composants d'une distribution Linux.

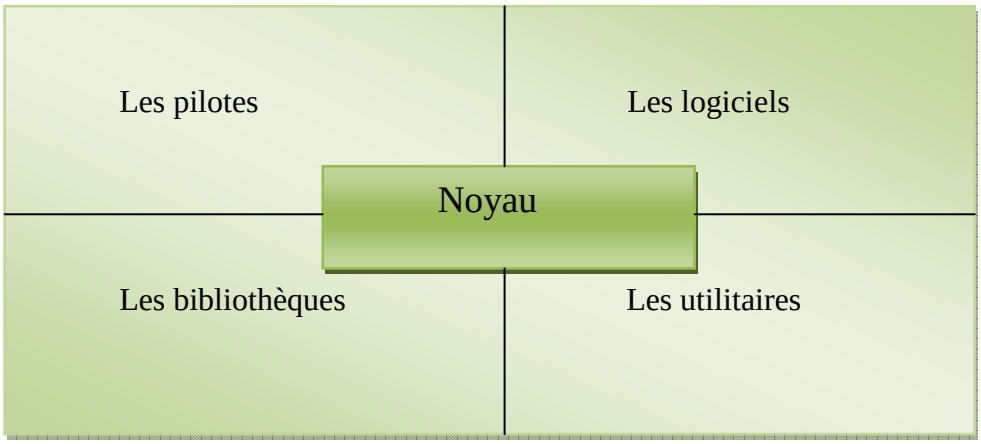

Figure II.3 *Composants d'une distribution Linux.*

I.4 LE DEMARRAGE D'UN SYSTEME UNIX

I.4.1 LE MBR « MASTER BOOT RECORD »

Le disque dur d'un PC peut être partitionné. Le premier secteur du disque contient le MBR «Master Boot Record» qui décrit le partitionnement du disque. Il est à l'origine du chargement du système d'exploitation.

La structure du MBR est la suivante:

- Les 446 premiers octets contiennent un programme de chargement, « loader» qui va démarrer l'exécution du programme de chargement propre au système d'exploitation actif.
- Les 64 octets qui suivent décrivent les partitions : taille, localisation, type.
- Les deux derniers octets du MBR constituent le magic number, une valeur numérique que certains systèmes utilisent pour vérifier la signature du secteur.

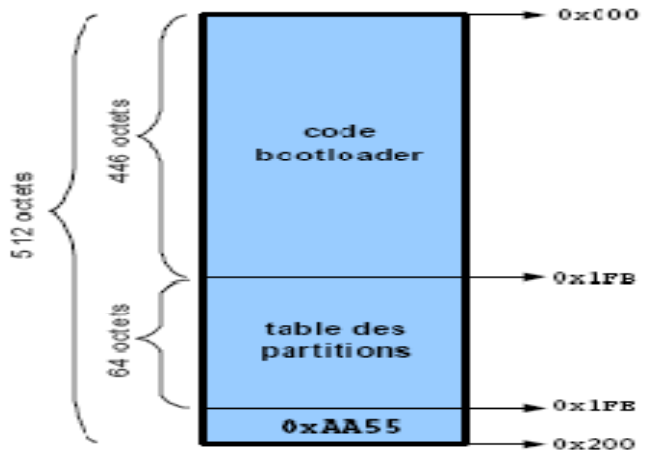

Figure I.7 *Structure du MBR.*

I.4.2 PROCEDURE DE BOOT

En général, un Pc commence par l'instruction située à l'emplacement mémoire 0x0F0000h – 0x0FFFFFh, qui contient le BIOS. Ce code effectue une recherche des différents périphériques tels que la carte graphique, le contrôleur de disque dur etc. Puis, il effectue des tests sur certains circuits vitaux comme le contrôleur DMA, la mémoire RAM et le contrôleur

d'interruption. Ensuite, le système initialise la table des vecteurs d'interruption dans le premier segment de la mémoire.

Ensuite, grâce à l'information du support de « boot » contenue dans la configuration du BIOS (disque dur, lecteur de disquettes, mémoire Flash), celui-ci sait sur quel support se trouve le système d'exploitation. Quel que soit le support de démarrage, le premier secteur est lu. Le code situé dans le MBR est chargé à l'emplacement mémoire 0x07C000h. Ce mini programme, chargé de lancer le kernel, est appelé gestionnaire de « boot ». Les plus connu du monde Pc sont Lilo et Grub.

I.4.3 L'ACTIVATION DU PROCESSUS INIT

Le système Linux offre plusieurs niveaux de fonctionnement. A chacun d'eux correspond un certain nombre de services pour les utilisateurs. Le rôle de la commande init, qui a comme argument le niveau de fonctionnement choisi, est d'activer tous les processus associé à ce niveau. La commande init trouve la définition des commandes à utiliser dans le fichier /etc/inittab.

I.4.4 LES NIVEAUX

Il existe onze niveaux, définis par les chiffres de zéro à neuf, plus les niveaux s ou S. Le tableau suivant montre leurs utilisations conventionnelles.

Niveau	Utilisation conventionnelle
0	Ce niveau est utilisé pour arrêter le système .La commande shutdown exécute, en final, le basculement au niveau 0.
1	Pour rentrer en mode mono-utilisateur, réservé à root.
2	Mode multi-utilisateurs, sans NFS.
3	Mode multi-utilisateurs avec tous les services réseaux.
4	Est laissé libre.
5	Mode multi-utilisateurs avec le démarrage du serveur graphique X11 en plus.
6	Redémarrage de la machine (la commande reboot lance le niveau 6).
s,S	S ou s sont des niveaux qui sont exécutés systématiquement par le système.

Tableau I.1 *Niveaux de démarrage.*

I.4.5 LE FICHIER /etc/inittab

Le fichier inittab détermine quelles sont les commandes que le processus init doit exécuter pour un niveau donné. Le processus init l'examine séquentiellement. Le fichier inittab est composé de lignes de commande chaque ligne est composée de quatre champs séparés par le caractère **:**

Id: niveau: action: commande: commentaire

Ces champs ont la signification suivante:

Id: pour identifier la ligne, ce champ est ignoré par init.

niveau: permet à init de décider si la commande doit être prise en compte pour le niveau demandé. On peut préciser plusieurs niveaux.

action: indique à init le contexte dont lequel la commande doit être exécutée.

Exemple

initdefault: détermine le niveau de fonctionnement par défaut.

wait: attendre que le processus qui exécute la commande soit terminé pour passer à la ligne suivante.

sysinit*:* les lignes sont exécutées une seule fois, lors du démarrage à froid du système.

commande: contient la ligne de commande à exécuter. La commande est ces arguments sont désignés par un chemin absolu.

Les scripts de démarrage:/etc/*rc*

La plupart des commandes exécutées par inittab sont des scripts. Un administrateur ne modifie pas inittab mais plutôt ces scripts de démarrage.

- **Le script rc.sysinit**

Le processus init commence par exécuter le script **/etc/rc.d/rc.sysinit** qui contient les commandes de contrôle d'initialisation. Les principales opérations sont:

- Initialisation de la variable PATH avec les répertoires /bin, /usr/bin, /sbinet /usr/sbin.
- Initialisation des disques de swap.
- Détermination du nom de la machine.

- Vérification du système de fichiers principaux root et montage du pseudo-système de fichiers /proc.
- Montage de tous les systèmes de fichiers, à l'exception des systèmes de fichiers NFS.
- Suppression des fichiers verrous des services « uucp », « X »...
- Initialisation des ports série.
- Chargement des pilotes dynamiques et les modules.
- Création du fichier /var/log/dmesg qui contient la configuration du système.

Le processus init exécute ensuite les scripts spécifiques au niveau de démarrage demandé.

❖ **Le script rc.local**

Le dernier script exécuté par init est le script **/etc/rc.d/rc.local.** C'est ce script que l'administrateur modifie pour que soient exécutées des commandes propres au site.

I.5 LES OUTILS DE DEVELOPPEMENT

I.5.1 INTRODUCTION

Jusqu'à présent nous avons étudié le système Linux, son noyau et les logiciels complémentaires livrés avec les distributions Linux .Il s'agit maintenant de comprendre ce que sont ses programmes, comment ils fonctionnent, comment ils ont été conçus et comment nous pouvons nous-mêmes développer une application. Dans ce chapitre nous allons présenter les différents outils qui nous permettent de développer une application sous Linux.

Les outils de développement disponibles librement sous Linux sont puissants. Au cœur de la programmation en langage C ou C++ se trouve le compilateur. Celui que l'on emploie sous Linux se nomme GCC.

Le processus de développement d'une application sous Linux commence en élaborant un cahier des charges pour déterminer les spécificités du projet. Puis nous éditons le programme avec l'éditeur de texte de notre choix (vi ou emacs).Nous devons ensuite compiler le programme en utilisons le compilateur GCC et en fin nous créons l'archive du projet. Ces étapes sont illustrées dans la figure ci-dessus.

Figure I.8 *Processus de développement.*

I.5.2 COMPILER AVEC GCC

Gcc est un compilateur libre et gratuit, il s'inscrit dans toute une lignée de compilateurs du projet GNU pour différents langages : Java, C, C++, Fortran, etc. Comme on va traiter de la programmation C, nous allons donc accorder une attention plus particulière à GCC dans ce contexte.

I.5.3 PROCESSUS DE COMPILATION

Le processus de compilation sous Linux est sensiblement le même que pour tous les systèmes d'exploitation. Cependant, les formats des différents produits de la compilation et du couplage diffèrent selon le système d'exploitation. Pour avoir le fichier exécutable il faut suivre les étapes suivantes :

- **Edition des sources (programmation) :** cette étape consiste à écrire le code source du programme. Elle peut se faire à l'aide d'un éditeur texte comme vi, emacs. Le résultat de cette étape, en C, est généralement composé de fichiers sources (.c) et de fichiers d'en-têtes (.h).

- **Compilation :** c'est-à-dire traduire chacun des fichiers sources (.c) en langage de bas niveau. Le résultat sera des fichiers dont l'extension est .o.

- **Edition de lien (coupleur) :** Maintenant que tous les fichiers source ont été traduits en langage machine dans des modules objet (.o), il faut rassembler tous les morceaux du programme dans un fichier exécutable final. C'est l'étape du couplage (linking).

- **Exécution :** si aucune erreur n'a été rencontrée par le compilateur+coupleur, il est maintenant possible d'exécuter le programme en souhaitant que ce dernier soit exempt de bogues.

Dans l'Annexe ci-joint vous allez trouver un exemple pour mettre en évidence le fonctionnement de GCC.

I.5.4 L'OUTIL MAKE

Make est un utilitaire qui va permettre d'assurer la maintenance automatique d'une application composée de plusieurs fichiers liés par des relations de dépendances. Ceci permet de soulager le travail du programmeur en ne réalisant que les tâches nécessaires à l'obtention du produit final. Toutes informations et règles nécessaires au travail de make sont incluses dans un fichier Makefile.

Le principe est d'utiliser la commande make qui va appeler un fichier *Makefile* qui gère les dépendances entre fichiers .On peut utiliser la commande make -f nom du fichier Makefile.

I.5.5 SYNTAXE DU MAKEFILE

Un fichier Makefile est constitué d'une suite de commandes définissant les objets construits (les cibles) à partir d'objets dont ils dépendent (les dépendances) selon les règles de construction. La syntaxe est la suivante : **cible: liste des dépendances**

<Tab>règles (actions)

Si le fichier cible est plus ancien que les fichiers de dépendances alors il faut exécuter les actions. La syntaxe indique qu'il faut taper sur la touche de Tab avant de décrire les actions à réaliser. La cible est un nom de fichier à obtenir, elle peut être aussi une action à réaliser, on parlera alors de pseudo cible (phony target).

Le rôle de make est d'enchaîner les opérations en vérifiant d'abord l'existence des fichiers cibles, en comparant leurs dates de création avec celles des fichiers de dépendance en réalisant que les opérations nécessaires. Make traite les règles dans l'ordre du fichier pourvu qu'il y ait une dépendance par rapport à la première sinon le traitement est terminé.

Nous présentons un exemple pour illustrer l'utilisation de make que vous allez trouver dans l'Annexe.

I.5.6 LES PSEUDO-CIBLES

Une pseudo-cible demande la réalisation d'une action.

- **install** pour réaliser la compilation du programme et installer l'exécutable et les fichiers associés dans les bons répertoires.
- **clean** supprime tous les fichiers .o.
- **All** pour réaliser toutes les actions du Makefile.

I.5.7 LES VARIABLES

Les variables sont des noms utilisés pour remplacer une chaîne de caractère représentant .La définition de la variable utilise le signe =.par exemple: OBJET=multi.o fonc1.o. Ensuite la variable sera utilisée dans la définition des règles en étant précédée du signe $.Par exemple:

```
OBJET=multi.o fonc1.o
esx: $(OBJET)
gcc -o esx $(OBJET)
```

Les variables peuvent remplacer des options de compilation: CFLAGS= -g -

```
gcc $(CFLAGS) -o esx $(OBJET)
```

I.5.8 LES REGLES IMPLICITES

Pour réaliser sa tâche, l'outil make peut utiliser des règles implicites et des variables standards. Make reconnaît des suffixes de noms de fichiers (.c, .cc, .p, .tex, .sh) et applique automatiquement les règles correspondantes.

```
$(cc)  $(CFLAGS) -o esx $(OBJET)
```

Il existe aussi des variables prédéfinies qui n'ont pas une valeur figée pour tout le Makefile mais dont la valeur dépend des règles en cours par exemple:

```
esx muti.c fonc1.h
gcc -o esx $<
S< désigne ici multi.c (le premier de la liste).
```

On peut maintenant réécrire le programme précédent en utilisant les variables et les règles implicites.

```
#Exemple de fichier Makefile
OBJET=multi.o bon.o
CC=gcc
GFLAGS= -wall -g
esx : $(OBJET)
	$(CC) -o esx $(OBJET)
multi.o : multi.c multi.h
	$(CC) -c $<
bon.o : bon.c multi.h
	$(CC) -c $<
clean:
	rm *.o esx
```

I.5.9 AUTOCONF ET AUTOMAKE

Au lieu d'écrire le fichier Makefile à la main, on peut lancer le programme configure: ./configure, cela crée un fichier Makefile. Make, compile le programme en utilisant le *Makefile* généré ainsi et make install, pour l'installer. Le fichier configure, est généré grâce à autoconf et automake génère d'autres fichiers nécessaires au bon fonctionnement de configure.

Nous créons un répertoire essai et sous ce répertoire on utilise un autre qui est src ou on déplace lest rois fichiers sources créées précédemment.

Il faut un fichier de configuration qui s'appelle *configure.in*. La méthode la plus simple est d'utiliser l'outil autoscan, puis au fur et à mesure que le projet s'améliorera, on éditera le fichier *configure.in* à la main pour y ajouter des lignes supplémentaires.

Le programme autoscan génère un seul fichier *configure.scan*. On le lance depuis la racine du projet. Puis il faut le renommer en *configure.in*, et l'éditer. Voici notre *configure.in* après édition:

```
#                                -*- Autoconf -*-
# Process this file with autoconf to produce a configure script.

AC_PREREQ(2.61)
AC_INIT(/home/chaouche/essai/src/bon.c, 0.0.1)
AC_CONFIG_SRCDIR(config.h.in)
```

```
AC_CONFIG_HEADER(config.h)
AM_INIT_AUTOMAKE(bon, 0.0.1)
# Checks for programs.
AC_PROG_CC
AC_PROG_INSTALL
AC_PROG_OBJC
AM_PROG_AS
# Checks for libraries.
# Checks for header files.
# Checks for typedefs, structures, and compiler characteristics.
# Checks for library functions.
AC_CONFIG_FILES(Makefile
        src/Makefile)
```

Maintenant, il est nécessaire de préciser quels sont les fichiers à compiler, dans le fichier /src/Makefile.am.La structure de ce fichier est la suivante :

```
bin_PROGRAMS =bon
bon_SOURCES =bon.c multi.c multi.h
```

Commençons par exécuter aclocal, sans arguments. Nous générons ainsi un fichier *aclocal.m4* qui contient des macros nécessaires pour la suite. Puis autoconf, sans arguments non plus génère le tant attendu script *configure*. Ensuite, on peut utiliser autoheader pour créer le fichier de configuration *config.h.in* nécessaire pour *config.h*.

Il faut créer les fichiers *NEWS*, *README*, *AUTHORS* et *ChangeLog*. Un simple touch nom_du_fichier suffit. Il en est de même pour les trois autres fichiers. On peut mettre un nom et un e-mail dans *AUTHORS*, et de mettre une première ligne dans *ChangeLog* afin de signaler quand le projet est lancé. En ce qui concerne *NEWS*, c'est le résumé de *ChangeLog*. Ainsi, les développeurs iront voir *ChangeLog*, et les utilisateurs se contenteront de *NEWS* qui leur est plus parlant. Quant au *README*, chacun y met ce qu'il veut.

A ce stade, tous les fichiers sont créés. Lançons automake -a.

Le contenu du fichier essai est le suivant:

:~/essai$ ls

AUTHORS Makefile README config.h config.status install-sh
COPYING Makefile.am aclocal.m4 config.h.in configure missing
ChangeLog Makefile.in autom4te.cache config.h.in configure.in src
INSTALL NEWS autoscan.log config.log depcomp stamp-h1

Le contenu du répertoire src est :

:~/essai/src$ ls

Makefile Makefile.in bon.c multi.c multi.o

Makefile.am bon bon.o multi

Exécutons ./configure, puis make. En tant que root lançons make install. Le résultat serait l'installation du binaire *bon* dans */usr/local/bin*. Pour faire le tarball contenant le projet, make dist créera ce tarball *projet-version.tar.gz*, dans notre cas bon-*0.0.1.tar.gz*.

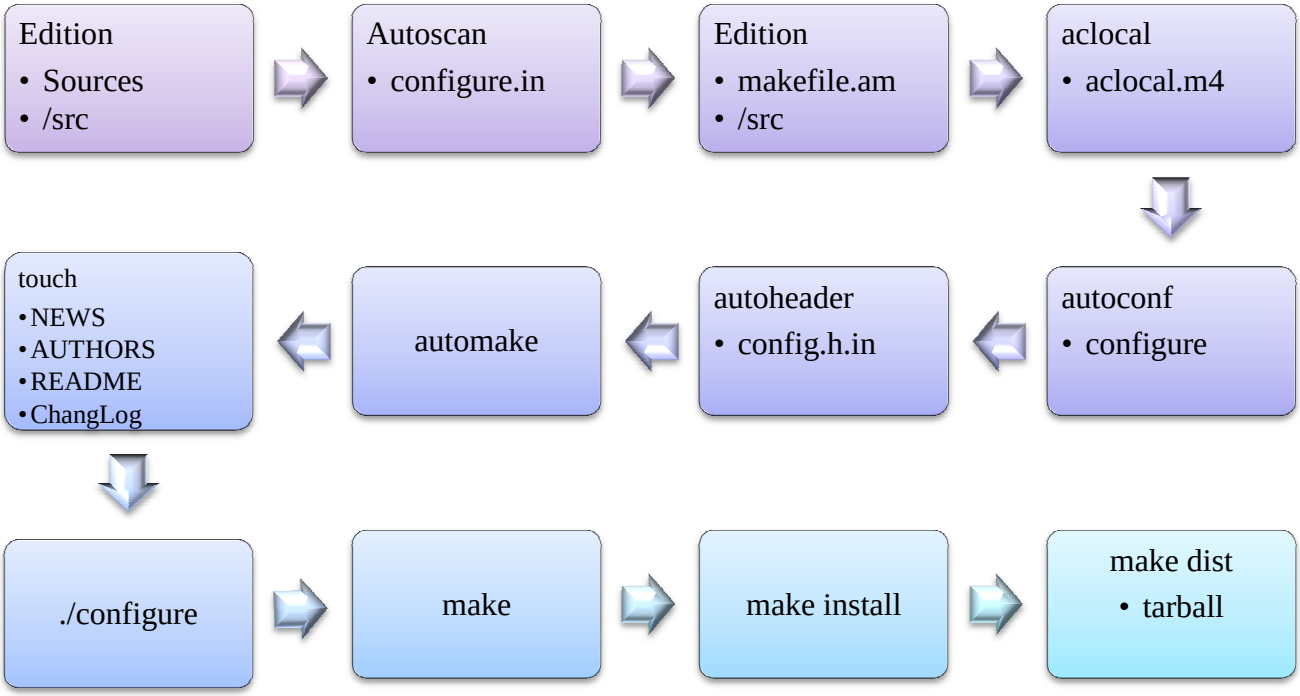

Figure I.9 *Processus de développement d'un projet sous Linux.*

CONCLUSION :

Dans ce chapitre, nous avons présenté une vue générale sur les systèmes d'exploitation, en particulier le système d'exploitation Linux. Nous avons accordé plus d'attention au noyau de ce dernier. Enfin, nous avons consacré une partie pour expliquer les outils de développement qui vont nous permettre de mieux comprendre le chapitre suivant et la manière avec laquelle nous avons procédé pour la création de notre distribution.

CHAPITRE 2

CREATION D'UNE DISTRIBUTION LINUX.

II.1 INTRODUCTION

Le but ultime de notre projet vise la création d'une distribution linux. Nous construirons, à travers une méthodologie, une distribution de base qui pourra être taillée par la suite en fonction de l'orientation que l'on souhaite lui donner. Dans noter cas, cette orientation consistera en la contribution à la réalisation d'une passerelle internet sécurisée pouvant être dotée de fonctionnalités supplémentaires telle que le filtrage multi niveaux par exemple.

Ce chapitre introduira les éléments nécessaires pour réaliser notre objectif.

II.2 NECESSITE D'UN SYSTEME HOTE

Le principe en tant que tel est très simple. En partant d'un système hôte, en tant qu'utilisateur, nous pouvons compiler tous les programmes sources essentiels en utilisant les outils de compilation de la distribution hôte, dans une partition du disque dur. Cela constitue un environnement de départ pour la construction du noyau Linux à personnaliser. Il sera placé dans la partition déjà créé. Nous allons recompiler tous les *programmes* un à un mais avec les outils de *compilation* produits à l'étape précédente.

Le résultat est un système pouvant être entièrement optimisé et personnalisé.

II.3 DIFFERENTES ETAPES POUR LA REALISATION DE NOTRE DISTRIBUTION

La réalisation de notre distribution passe par les étapes suivantes :

- Construction d'un système temporaire sur une partition du disque dur. Nous y avons intégré certains logiciels primordiaux pour tout système d'exploitation (perl, binutils, glibc, gcc…).
- Création de notre système final. Pour réaliser ceci, nous avons utilisé l'environnement « chroot ». Dans ce dernier, nous avons créé toute l'arborescence du système, quelques liens symboliques primordiaux et des fichiers tels que passwd, group et les divers journaux.
- Intégration de tous les logiciels importants nécessaires à un système final (décompression, compilation, programme pour la gestion des fichiers…etc.). A la fin de cette étape, notre système devient opérationnel. L'initialisation de certains scripts de démarrage sont nécessaires pour la configuration de la console, du clavier, des utilisateurs et celle du réseau aussi.
- Rendre la distribution **bootable** pour pouvoir l'utiliser d'une manière autonome par la suite.
- Personnalisation de notre distribution. Dans notre cas, mise en place de la première brique d'une passerelle internet qui consiste en le filtrage réseau bas niveau. Ce réseau contiendra suivant les besoins, une ou plusieurs DMZ ainsi qu'un ou plusieurs réseaux virtuels.

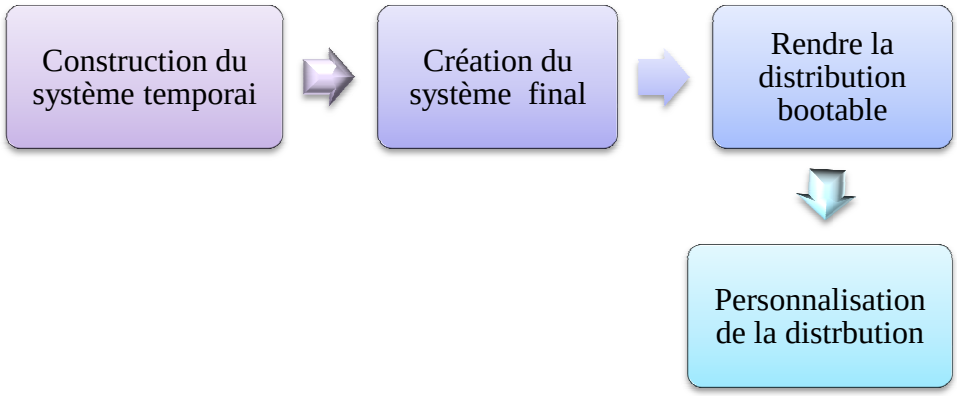

Figure II.1 *Démarche pour la création d'une distribution Linux.*

II.4 ILLUSTRATION DE QUELQUES PAQUETS A INSTALLER

II.4.1 LES PAQUETS TOOLCHAIN

Toolchain est la composition de trois éléments binutils, GCC et Glibc.

Le paquet **Binutils** doit être le premier à être compilé. Car **Glibc** et **GCC** réalisent différents tests sur l'éditeur de liens et l'assembleur disponibles pour déterminer leurs propres fonctionnalités à activer.

Le paquet **GCC** (GNU *Compiler Collection*) a été conçu pour remplacer le compilateur C (CC) fourni sur Unix qui est devenu très extensible. Aussi le support de nombreux autres langages a été ajouté.

Le paquet **Glibc** contient la bibliothèque C principale fournissant toutes les routines basiques pour allocation mémoire, la recherche des répertoires…etc.

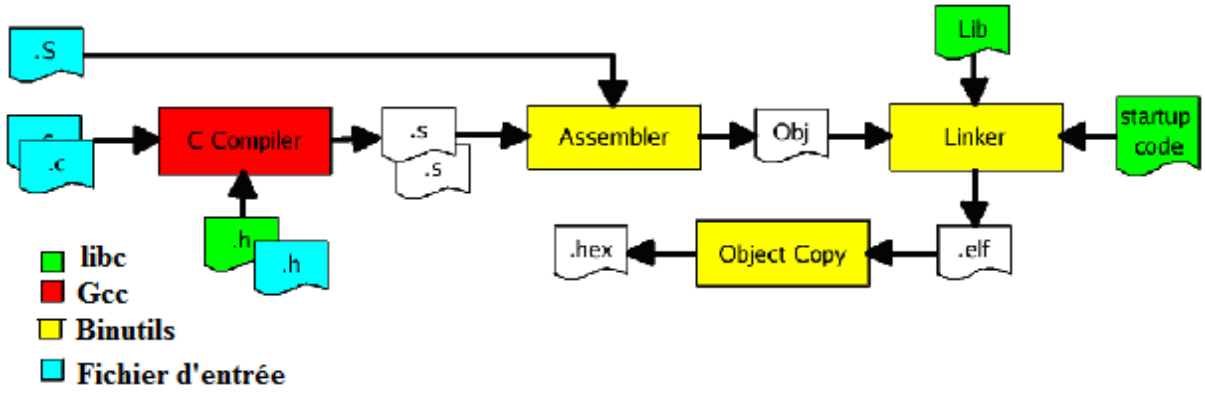

Figure II.2 *Relation entre les paquets de Toolchain lors du traitement d'un fichier.*

II.4.2 LE PAQUET NCURSES

L'étape suivante consiste à mettre Bash en place, mais bash a besoin de ncurses, donc nous devons installer celui-ci en premier.

Ncurses remplace termcap dans la manière de gérer les écrans textes, mais apporte également une compatibilité descendante en prenant en charge les appels termcap. Car il peut faire la gestion des écrans type caractère indépendamment des terminaux et cela en utilisant

les bibliothèques qui venant avec. Dans l'objectif d'avoir un système simple et propre, il est préférable de désactiver l'ancienne méthode termcap. Si le besoin d'utilisation de termcap se fait sentir, on pourra recompiler ncurses avec prise en charge de termcap.

Architecture de Ncurses :

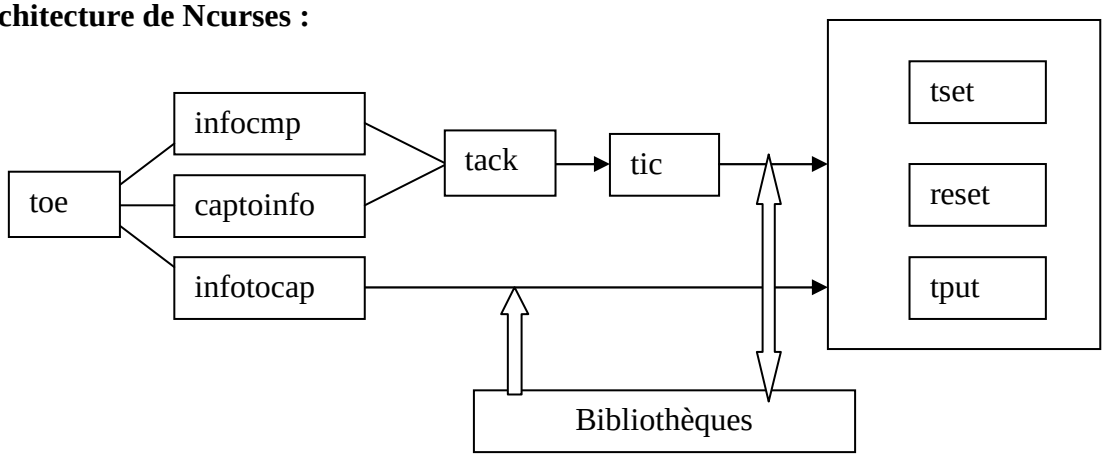

Figure II.3 *Architecture de ncurses.*

Explication

- **toe :** Liste le nom et la description de tous les types de terminaux disponibles.
- **infocmp :** Compare ou affiche les descriptions terminfo.
- **captoinfo :** Convertit une description termcap en description terminfo.
- **infotocap :** Convertit une description terminfo en description termcap.
- **tack :** Vérifie les actions terminfo, tester la correction d'une entrée dans la base de données terminfo.
- **tic :** traduit un fichier terminfo au format source dans un format binaire nécessaire pour les routines des bibliothèques ncurses.
- **tset :** Peut être utilisé pour initialiser des terminaux.
- **reset :** Réinitialise un terminal avec ses valeurs par défaut.
- **tput :** Rend les valeurs de capacités dépendant du terminal disponibles au shell ; il peut aussi être utilisé pour réinitialiser un terminal ou pour afficher son nom long.

Bibliothèques :

- **libcurses** : Un lien vers libncurses.

- **libncurses** : Contient des fonctions pour afficher du texte de plusieurs façons compliquées sur un écran de terminal ; un bon exemple d'utilisation de ces fonctions est le menu affiché par le **make menuconfig** du noyau
- **libform** : Contient des fonctions pour implémenter des formes

II.4.3 LE PAQUET BASH (BOURNE AGAIN SHELL)

Ce paquet contient les fonctionnalités suivantes :

- **bash :** Un interpréteur de commandes largement utilisé ; il réalise un grand nombre d'expansions et de substitutions sur une ligne de commande donnée avant de l'exécuter, rendant cet interpréteur très puissant
- **bashbug** Un script shell pour aider l'utilisateur à composer et à envoyer des courriers électroniques contenant des rapports de bogues.
- **sh** Un lien symbolique vers le programme **bash.**

II.4.4 GRUB

Le *GNU GRand Unified Boot loader* (ou GRUB) est un utilitaire permettant à l'utilisateur de sélectionner le système d'exploitation ou noyau qui doit être chargé au démarrage du système. Il permet également à l'utilisateur de transmettre des arguments au noyau.

Son installation permet aussi de pouvoir aisément démarrer au choix Windows ou GNU/linux sur machine embarquant ces deux systèmes.

Grub se décompose en deux parties logicielles :

- Un premier petit ensemble **(stage 1)** est écrit sur un support d'amorçage tel qu'une disquette ou le MBR de notre machine. Le stage 1 ne fait qu'appeler le stage 2 dans */boot/grub* qui, prenant le relais, assure l'exécution de Grub.
- Un deuxième (**stage 2**), plus volumineux car c'est en réalité le "vrai" corps actif de Grub, est installé dans le répertoire /*boot/grub* approprié, qu'il soit situé sur la partition racine, dans une partition */boot*, ou sur un support quelconque de type floppy ou cdrom.
- Un élément optionnel (**stage 1.5**) est occasionnellement utilisé afin de fournir au Stage 1 la méthode appropriée, liée au système de fichiers utilisé, pour accéder au stage 2.

Installation du Grub :

Après avoir téléchargé et désarchiver le grub (dans notre cas grub-0.97.tar.bz2). Il est temps d'appliquer le correctif, ensuite, nous le préparerons pour la compilation et on recommandera au script de configuration d'installer Grub dans */usr* .Et enfin, nous l'installerons mais il faut remplacer le i-386 par le répertoire adéquat pour le matériel utilisé. Et tout cela est fait en lançant le script :

```
tar –xvf ../grub-0.97.tar.bz2
patch -Np1 -i /lfs-sources/grub-0.97-disk_geometry-1.patch
./configure --prefix=/usr
make install
mkdir -v /boot/grub
cp -v /usr/lib/grub/i386-pc/stage{1,2} /boot/grub
```

II.4.5 MODULE INIT TOOLS

Ce paquet contient des programmes de gestion des modules des noyaux Linux pour les versions 2.5.47 et ultérieures.

Les modules sont une façon qui a été trouvée pour rendre Linux (le noyau) plus léger. En effet, ce sont le plus souvent des drivers qui ont été éjectés du noyau et qui sont chargés en mémoire, soit automatiquement lorsqu'on se sert du périphérique concerné, soit manuellement. Ils permettent de réduire la taille mémoire utilisée par le noyau lui-même. Le concept de modules permet aussi d'ajouter des drivers de périphériques au système sans devoir recompiler le noyau.

Ils résident dans /lib/modules/version_du_noyau, ce répertoire contient les dossiers suivants qui permettent de séparer les modules selon leur type :

`block/`	périphériques de type bloc
`fs/`	systèmes de fichier (FAT, ext2, etc.)
`misc/`	périphériques divers (carte tuner TV, port parallèle, carte son, joystick, etc.)
`net/`	drivers des cartes réseaux et protocoles
`scsi/`	périphériques SCSI
`cdrom/`	drivers CDROM non IDE (anciens CD-ROMs)
`ipv4/`	protocole IP v4 et v6 (protocole réseau)
`ipv6/`	
`pcmcia/`	modules pour gérer les cartes PCMCIA
`video/`	gestion des cartes vidéo en console

Tableau II.1 *Les périphériques.*

Ce répertoire peut contenir d'autres dossiers, plus un fichier nommé **modules.dep**, contenant les dépendances entre modules. Ce fichier est généralement regénéré à chaque boot par la commande « **depmod –a »** placée dans les scripts de démarrage.

Le paquet module init tools contient des programmes permettant la manipulation des différents modules du noyau. Donc :

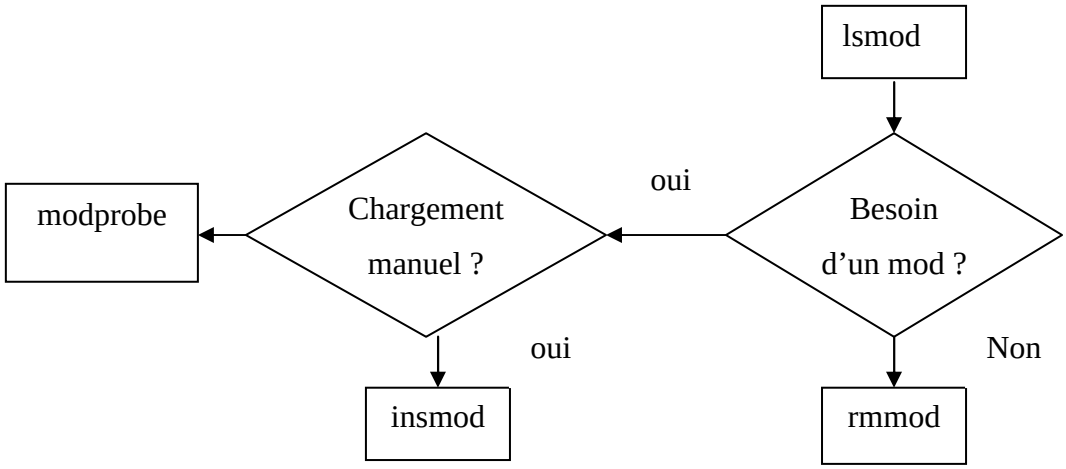

Figure II.4 *Fonctionnement de Module init tools.*

- **lsmod :** affiche la liste des modules chargés, les dépendances entre eux, et nous informe si les modules sont utilisés ou non.
- **insmod module :** charge un module en mémoire. Si ce module est spécifié sans extension, cette commande cherchera le module dans des répertoires par défaut, en général */lib/module/version_du_noyau.* Sinon, il faut donner le chemin où trouver le module.
- **rmmod module :** décharge un module.

- **modprobe :** Utilise un fichier de dépendances, créé par **depmod**, pour charger automatiquement les modules adéquats.

II.4.6 SHADOW

Ce paquet contient des programmes de gestion de mots de passe d'une façon sécurisée c'est à dire des outils pour : ajouter, modifier, supprimer des utilisateurs et des groupes, pour : initialiser et changer leur mots de passe, et bien d'autres tâches administratives.

Important :

Si on décide d'utiliser le support de Shadow : les programmes qui ont besoin de vérifier les mots de passe (gestionnaires d'affichage, programmes FTP…etc.) ont besoin d'être *compatible avec shadow*, c'est-à-dire qu'ils ont besoin d'être capables de fonctionner avec des mots de passe shadow

Configuration de Shadow :

- Pour activer les mots de passe shadow, on lance la commande suivante : **pwconv**
- Pour activer les mots de passe shadow pour les groupes, on lance : **grpconv**
- Pour la réinitialisation tous les mots de passe des utilisateurs actuels avec on utilise la commande **passwd** et les mots de passe des groupes avec la commande **gpasswd**.

II.4.7 SYSVINIT

SysVinit (qui sera nommé par la suite « init ») fonctionne en utilisant un schéma de niveaux d'exécution.

Ils sont au nombre de sept (numérotés de 0 à 6). En fait, il en existe plus mais ils sont pour des cas spéciaux et ne sont généralement pas utilisés. Chacun d'entre eux correspond à des actions que le système est supposé effectuer lorsqu'il démarre. Le niveau d'exécution par défaut est 3.

Descriptions de ce que font les arguments des scripts :

- **Start :** Le service est lancé.
- **Stop :** Le service est stoppé.
- **Restart :** Le service est stoppé puis de nouveau lancé.

- **Reload :** La configuration du service est mise à jour. Ceci est utilisé après que le fichier de configuration d'un service a été modifié, quand le service n'a pas besoin d'être redémarré.
- **Status :** Indique si le service est en cours d'exécution ainsi que les PID associés.

IMPORTANT

On peut faire l’installation de tous ces paquets d’une façon automatisée en utilisant les **scripts** que nous avons construits et contenus dans le système de validation dans le CD ci-joint.

II .5 INITIALISATION DES SCRIPTS DE DEMARRAGE DU SYSTEME

Dans cette partie, on va montrer comment installer et configurer les scripts de démarrage de notre système.

On commence d’abord par l’installation de lfs-bootscript (dans notre cas lfs-bootscript-6.3.tar.bz2) qui sert à démarrer ou arrêter notre système en utilisant un ensemble de scripts de démarrage.

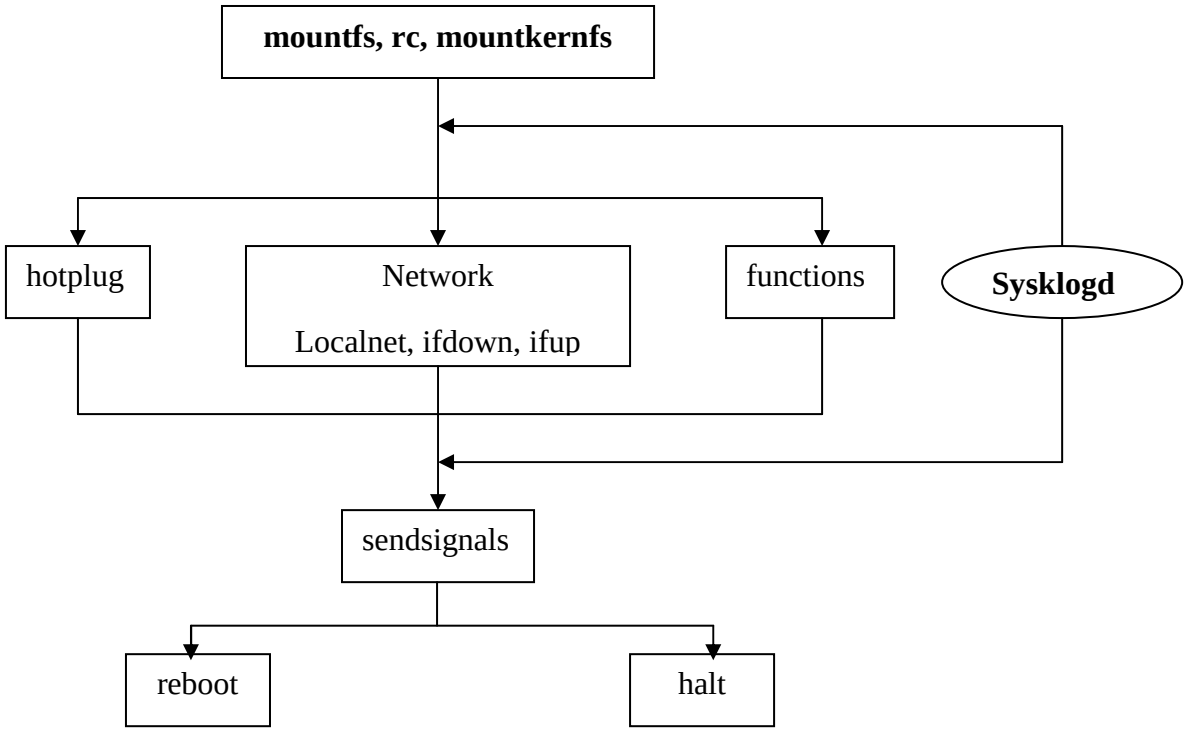

Figure II.5 *Fonctionnement de lfs-bootscript.*

Description : ce paquet contient plusieurs programmes tel que :

- **mountfs :** Monte tous les systèmes de fichiers, sauf ceux marqués *noauto* et les systèmes réseaux.
- **mountkernfs :** Monte les systèmes de fichiers virtuels fournis par le noyau, tels que *proc*.
- **rc :** Le script de contrôle du niveau d'exécution maître ; il est responsable du lancement des autres scripts un par un dans une séquence déterminée par le nom des liens symboliques en cours de traitement.
- **ifdown :** Assiste le script network pour l'arrêt des périphériques réseaux.
- **ifup :** Assiste le script network pour le démarrage des périphériques réseaux.
- **localnet :** Configure le nom d'hôte du système et le périphérique de boucle locale.
- **hotplug :** Charge des modules pour les périphériques système.
- **sysklogd :** Lance et arrête les démons des journaux système et noyau.
- **Sendsignals :** S'assure que chaque processus est terminé avant que le système redémarre ou ne s'arrête.
- **halt, reboot :** Arrêt/Redémarre le système.

II.6 FONCTIONNEMENT DES SCRIPTS DE DEMARRAGE

Linux utilise un service de démarrage spécial nommé **SysVinit** qui est basé sur un concept de *niveaux d'exécution*. Cela peut être bien différent d'un système à un autre, du coup, LFS a sa propre façon de le faire mais il respecte généralement les standards établis.

Pour avoir plus d'informations sur le démarrage, veuillez voir le fonctionnement de **SysVinit** dans la partie concernant le démarrage d'un système Linux (Chapitre 1).

Après avoir lancé les scripts de démarrage, il est temps de faire les configurations nécessaires pour rendre notre système plus portable et plus léger pour l'utilisation, donc on a intérêt à configurer tout ce qui est console, clavier …etc. ce qui est disponible dans l'Annexe.

II.7 RENDRE LE SYSTEME BOOTABLE

Il est temps de rendre amorçable notre système Linux minimal. Nous allons donc traiter la création d'un fichier **fstab**, de la construction d'un noyau pour le nouveau système et de l'installation du chargeur de démarrage Grub.

- **Création du fichier /etc/fstab**

Cette table est utilisée par quelques programmes pour déterminer les partitions à monter par défaut, leurs ordres, et quels systèmes de fichiers sont à vérifier.

On donnera en annexe ci –joint notre exemple de /etc/fstab.

II.7.1 INSTALLATION DE LINUX

Au démarrage de l'ordinateur, le noyau est la première partie du système d'exploitation à être chargée. Il détecte et initialise tous les composants matériels de l'ordinateur, puis rend disponibles ces composants par un ensemble de fichiers pour les logiciels qui en ont besoin, et transforme un CPU unique en une machine multitâches capable d'exécuter des bouts de programmes quasiment au même moment.

Pour installer le noyau, on doit confirmer qu'il est complètement nettoyé, et pour faciliter la configuration on la fait via l'interface par menu, on compile l'image du noyau et on installe les modules en utilisant les commandes suivantes :

```
make mrproper
make menuconfig
make
make modules_install
```

Rendre le nouveau système bootable

Notre nouveau de système est pratiquement fini. Il faut s'assurer que le système peut démarrer proprement.

Le chargement au démarrage est complexe. Tout d'abord, on doit vraiment connaître le chargeur actuel et tout autre système d'exploitation présent sur le disque dur amorçable. Nous avons vu précédemment comment compiler et installer le chargeur de démarrage Grub. On configure grub comme c'est déjà mentionné et on crée le fichier /boot/grub/menu.lst en ajoutant les entrées souhaitées.

II.8 SECURITE

II .8.1 INTRODUCTION

La finalité de notre PFE est de contribuer à la réalisation d'une distribution fonctionnelle jouant le rôle d'une passerelle multi réseau. C'est une distribution qui va sécuriser les communications sortantes et entrantes en filtrant les trames. Nous accordons une attention particulière à la sécurité.

La sécurité prend différentes formes en informatique. Cette partie donne des exemples provenant de trois types différents de sécurité : accès, prévention et détection.

L'accès

L'accès pour les utilisateurs est généralement géré par **login** ou par une application conçue pour gérer la fonction de connexion. Nous montrons comment améliorer **login** en mettant en place des politiques avec les modules PAM. L'accès via le réseau peut aussi être sécurisé par des politiques initialisées avec iptables, ce qui est généralement appelé un pare-feu.

Prévention

En prévention des brèches, comme un cheval de Troie, des applications comme GnuPG aident, par exemple à donner la possibilité de confirmer des paquetages signés en reconnaissant des modifications d'archives TAR après que le mainteneur l'ait créé.

Détection

Enfin, nous arrivons à la détection avec un paquetage qui stocke les "signatures" de fichiers critiques (définis par l'administrateur), régénère les "signatures" et les compare aux fichiers qui ont été modifiés.

Dans ce qui suit, nous allons présenter la procédure d'installation des paquets indispensables pour la sécurité informatique.

III.8.2 ILLUSTRATION DE QUELQUES PAQUETS A INSTALLER

III.8.2.1 LE PAQUET OPENSSL

OpenSSL est un package logiciel incluant une librairie assez complète de cryptographie. L'IGC un organisme qui va donner à ses utilisateurs des certificats qui permettent de les identifier. Les paramètres de l'IGC sont configurés via un fichier nommé openssl.cnf.

On a besoin de ce paquet pour effectuer certaines tâches de cryptographie telles que:

- créer des paramètres de clés RSA, DH et DSA;
- créer des certificats X.509, CSR et CRL;
- calculer des empreintes numériques;
- crypter et décrypter par des algorithmes de chiffrement;

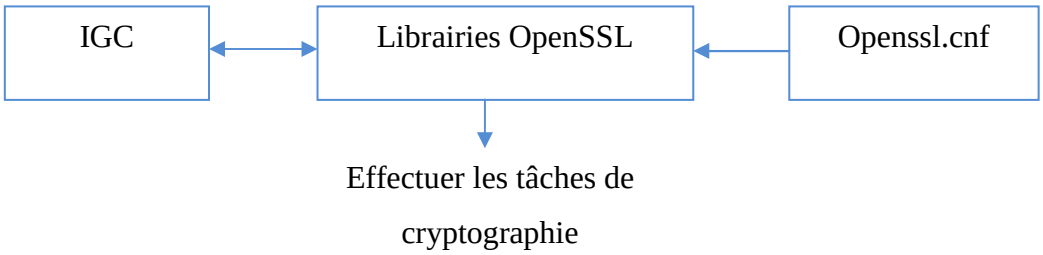

Figure II.6 *Architecture d'OpenSSL.*

Dans notre cas nous allons utiliser le paquet OpenSSL version 0.9.8 et le patch openssl-0.9.8d-fix_manpages-1.patch.

Avant installation du paquet, nous allons procéder de la manière suivante :

Utilisons la commande suivante: sed –i -e 's/mcpu/march/' config pour éviter les avertissements affichés lors de l'installation du paquet. La fonction de substitution sed permet de changer la première chaine mcpu dans le fichier config par march.

Utilisons un patch pour fixer les conflits entre les manuelles installés par openSSL et ceux trouvés dans d'autres paquets (particulièrement, shadow).

Effectuons la configuration en utilisant les options suivantes :

--openssldir=/etc/ssl : pour installer openSSL dans le répertoire /etc/ssl.

--prefix=/usr : pour installer les programmes d'openSSL dans le répertoire /usr.

L'installation du paquet s'effectue en copiant :

- le manuel d'OpenSSL dans /usr/share/man.
- les certificats en utilisant la commande cp –v –r certs/etc/ssl.
- la documentation (HOWTO et README) dans le répertoire /usr/share/doc/openssl-0.9.8d.

Les commandes utilisées sont présentées ci-dessous.

```
sed -i -e 's/mcpu/march/' config
patch -Np1 -i ../openssl-0.9.8d-fix_manpages-1.patch &&
./config --openssldir=/etc/ssl --prefix=/usr shared &&
make MANDIR=/usr/share/man
make MANDIR=/usr/share/man install &&
cp -v -r certs /etc/ssl &&
install -v -d -m755 /usr/share/doc/openssl-0.9.8d &&
cp -v -r doc/{HOWTO,README,*.{txt,html,gif}} \
/usr/share/doc/openssl-0.9.8d
```

III.8.2.2 LINUX PAM (PLUGGABLE AUTHENTICATION MODULES OU MODULES D'AUTHENTIFICATION ENFICHABLES)

Linux-PAM est un système de bibliothèques qui gèrent les tâches d'authentification des applications (services) sur le système. Les bibliothèques fournissent une interface d'abstraction stable (Application Programming Interface - API) qui accorde les privilèges aux programmes (comme login et su) en ajournant la réalisation des tâches standard d'authentification.

La figure suivante illustre la relation entre les applications, la bibliothèque PAM et les modules d'authentification .Les applications (login, telnet and ftp) utilise une API (PAM authentication interfaces) qui charge le module d'authentification déterminé dans Pam.conf (qu'on va détailler ultérieurement).

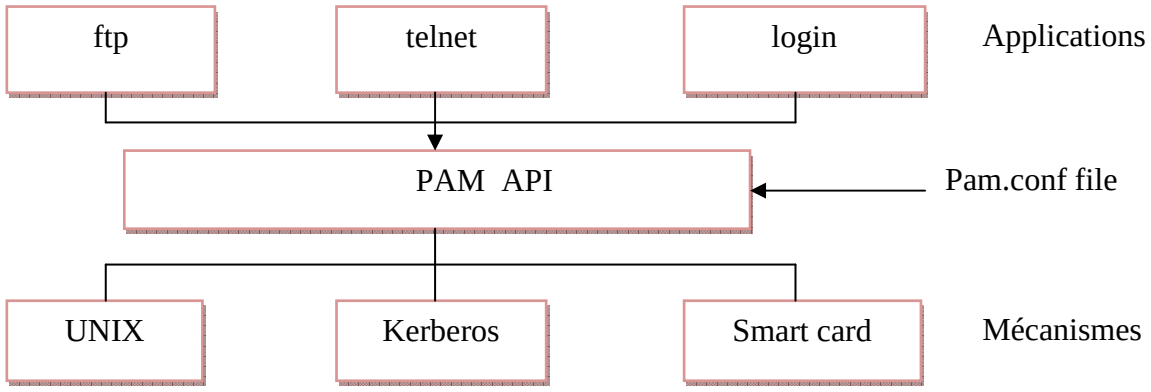

Figure II.7 *Architecture PAM.*

Dans notre cas nous allons utiliser le paquet Linux-PAM-0.99.4.0.tar.bz2.
Nous lançons la configuration avec des options qui nous permettent de :

- Installer les bibliothèques dans le répertoire /usr/lib.
- Installer la documentation dans le répertoire /usr/share/doc/Linux-PAM-0.99.4.0.
- Permettre à l'administrateur de choisir le fichier de configuration utilisé (`etc/pam.d` ou `/etc/pam.conf`).

Par la suite nous compilant ce paquet. Pour cela nous utilisons les commandes ci-dessus.

```
./configure --libdir=/usr/lib \
--sbindir=/lib/security \
--enable-securedir=/lib/security \
--enable-docdir=/usr/share/doc/Linux-PAM-0.99.4.0 \
--enable-read-both-confs &&
 make
```

Après l'installation du paquet, nous avons besoin de changer les droits d'accès à unix_chkpwd .Ce fichier contient le password-helper. Pour plus de sécurité, nous donnons les droits seulement à l'utilisateur root.

```
make install &&
chmod -v 4755 /lib/security/unix_chkpwd &&
```

Configuration de Linux PAM

Les informations de configuration sont disponibles dans `/etc/pam.d` ou `/etc/pam.conf` suivant la préférence de l'utilisateur. Ci-dessous, se trouvent des fichiers d'exemples pour chaque type :

Le fichier /**etc/pam.conf** est construit une liste de règles. La syntaxe du fichier de configuration se présente comme suit :

service-name module-type control-flag module-path args

- **service-name** est habituellement le nom familier de l'application correspondante : **login** et **su** par exemples. Le nom de **service**, **other,** est réservé pour donner des règles par default. Seules les lignes qui mentionnent le service courant (ou en cas d'absence de l'entrée **other**) seront associées avec l'application ou le service associé.

- **module-type :**
  - **auth :** authentification de l'utilisateur (établit la correspondance entre l'utilisateur et celui pour lequel il prétend être).
  - **account :** gestion des utilisateurs **(**Est-ce que le mot de passe de l'utilisateur a expiré ? Cet utilisateur a-t-il le droit d'accéder au service demandé ?).
  - **session :** tâches à effectuer en début et fin de chaque session ex : montage de répertoires, contrôle des ressources.
  - **password :** mettre à jour les mécanismes d'authentification. La méthode standard sous UNIX basée sur les mots de passe représente l'exemple évident : veuillez entrer un mot de passe de remplacement.
- **control-flag :** contrôle de réussite après la tâche d'authentification.
  - required : la réussite d'au moins un des modules required est nécessaire.
  - requisite : la réussite de tous les modules requisite est nécessaire.
  - sufficient : la réussite d'un seul module sufficient est suffisante.
  - optional : la réussite d'au moins un des modules required est nécessaire si aucun autre n'a réussi.
- **module-path :** chemin du module : en général dans /usr/lib/security.
- **args :** (arguments) optionnels.

Un exemple du fichier /etc/pam.conf est présenté ci-dessus :

```
        # Begin /etc/pam.conf

        other           auth            required        pam_unix.so
nullok
        other       account     required      pam_unix.so
        other       session     required      pam_unix.so
        other           password        required        pam_unix.so
nullok

        # End /etc/pam.conf
```

La syntaxe des fichiers contenus dans le répertoire **/etc/pam.d/** est identique excepté que le champ *service* est absent. Dans ce cas, le *service* est le nom du fichier dans le répertoire **/etc/pam.d/ .** Le nom de fichier doit être en minuscules.

```
# Begin /etc/pam.d/other
auth            required        pam_unix.so     nullok
account         required        pam_unix.so
session         required        pam_unix.so
password        required        pam_unix.so     nullok
# End /etc/pam.d/other
```

III.8.2.3 TRIPWIRE

Tripwire aide à assurer l'intégrité de répertoires et de systèmes de fichiers importants en identifiant tout changement apporté à ceux-ci. L'utilisation de Tripwire pour détecter des intrusions dans le système et analyser les dommages causés, aide à contrôler les changements apportés au système et accélère la vitesse de sa remise en état lorsqu'il est victime d'une violation, en réduisant le nombre de fichiers devant être restaurés pour le réparer**.**

La figure suivante illustre les étapes de fonctionnement de Tripwire en commençant par l'installation, la configuration et en fin la procédure de vérification de l'intégrité de des répertoires.

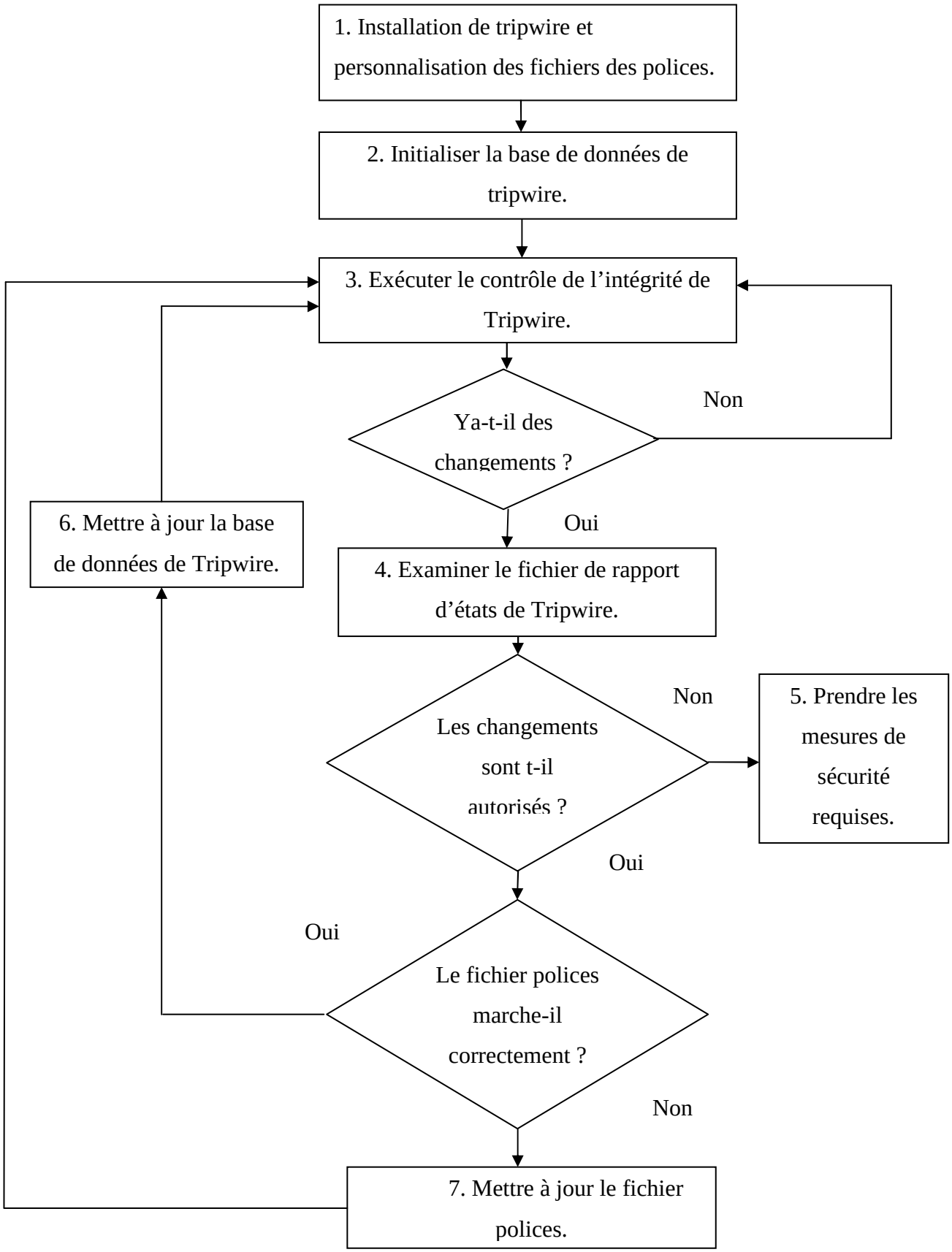

Figure II.8 *Etapes de fonctionnement de tripwire.*

Tout d'abord nous créons un lien symbolique vers le répertoire de construction, Puis nous appliquons le patch ce dernier contient gcc-4 fixes. Nous utilisons la commande sed pour installer les programmes dans /etc/lib/tripwire.la configuration s'effectue avec paramètre suivants :

- --prefix=/usr : indiquer aux scripts de configuration de préparer l'installation dans /usr.
- --sysconfdir=/etc/tripwire : pour indiquer aux programmes créer par le paquet de chercher les fichiers de configuration dans /etc/tripwire.

Par la suite nous lançons la compilation.

```
ln -s contrib install &&
patch -Np1 -i ../tripwire-2.4.0.1-gcc4_build_fixes-1.patch &&
sed -i -e 's@TWDB="${prefix}@TWDB="/var@' install/install.cfg &&
./configure --prefix=/usr --sysconfdir=/etc/tripwire &&
make
```

L'installation s'effectue en copiant la documentation dans /usr/share/doc/tripwire.

```
make install &&
cp -v policy/*.txt /usr/share/doc/tripwire
```

Configuration de tripwire

Tripwire utilise un fichier de règles pour déterminer quels sont les fichiers dont l'intégrité est vérifiée. Le fichier de règles par défaut est twpol.txt trouvé dans /etc/tripwire/. Ce fichier à besoin d'être configuré en suivant les étapes suivantes :

Nous allons demander la génération d'un fichier *hashé* et chiffré, en précisant l'utilisation de notre clé du site préalablement générée lors de l'installation de l'outil. Le fichier ici créé est un fichier de configuration global. Puis nous initialisons la base de données.

```
twadmin --create-polfile --site-keyfile /etc/tripwire/site.key \
/etc/tripwire/twpol.txt &&
tripwire --init
```

On peut vérifier la base de données par la commande ci dessous .Un rapport a été généré, dans le répertoire /var/lib/tripwire/report/.

```
tripwire --check > /etc/tripwire/report.txt
```

III.8.2.4 HEIMDAL

Heimdal est une implémentation libre de Kerberos5. C'est un protocole d'authentification réseau, il préserve l'intégrité des mots de passe dans un réseau sans confiance.

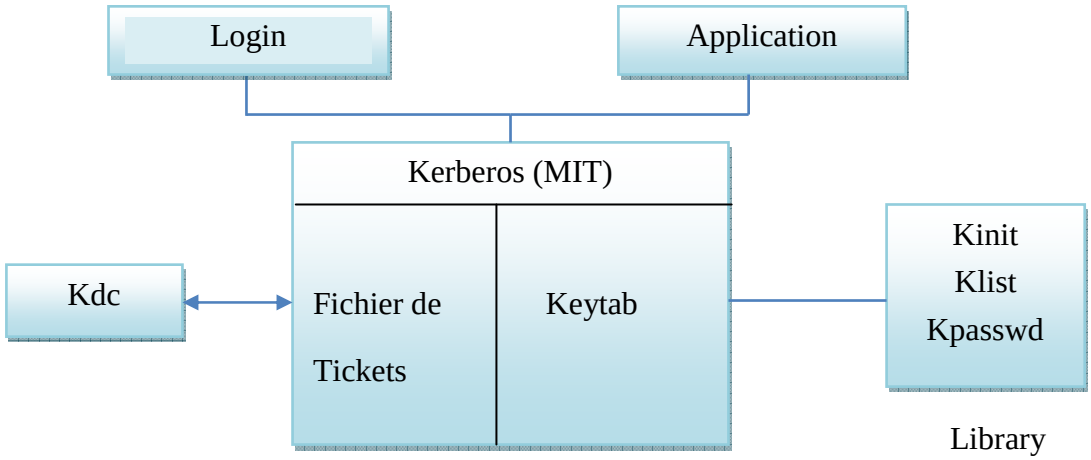

Figure II.9 *Architecture de heimdal.*

Lorsqu'un utilisateur souhaite s'authentifier il envoie son mot de passe chiffré au KDC (key Distribution Center : service émetteur de tickets), après authentification l'utilisateur demande au KDC des tickets (avec le programme kinit) qui serviront comme une preuve pour s'authentifier au prés des services.

Nous avons choisi d'utiliser : heimdal-0.7.2.tar.gz,
heimdal-0.7.2-fhs_compliance-1.patch,
heimdal-0.7.2-cracklib-1.patch.

Commençons l'installation par l'application des correctifs. Le premier set à activer kpasswd et kadmin pour utiliser la bibliothèque cracklib**.**

```
patch -Np1 -i ../heimdal-0.7.2-cracklib-1.patch
```

Le deuxième patch change toutes les références de /var/heimdal à /var/lib/heimdal dans le code source et la documentation pour qu'elles soient conforme avec FHS.

Nous lançons la configuration en utilisons des options qui nous permettent de :

- Indiquer aux scripts de configuration de préparer l'installation dans le répertoire /usr.
- Modifier la valeur par défaut /usr /libexec par /usr/sbin.
- Indiquer que l'emplacement des donnés est /var/lib/heimdal.
- Modifier l'emplacement de la base de donnés locale avec /var/lib/local pour être compatible avec FHS.
- Enab
- Utiliser la bibliothèque readline.

Nous compilons en suite le paquet.

```
patch -Np1 -i ../heimdal-0.7.2-fhs_compliance-1.patch &&
./configure --prefix=/usr \
--sysconfdir=/etc/heimdal \
--libexecdir=/usr/sbin \
--datadir=/var/lib/heimdal \
--localstatedir=/var/lib/heimdal \
--enable-shared \
--with-readline=/usr &&
make
make install
```

Configuration de heimdal

Nous créons le fichier de configuration Kerberos avec la commande suivante :

```
install -v -m755 -d /etc/heimdal &&
cat > /etc/heimdal/krb5.conf << "EOF"
# Begin /etc/heimdal/krb5.conf
[libdefaults]
default_realm = <EXAMPLE.COM>
encrypt = true
[realms]
<EXAMPLE.COM> = {
kdc = <hostname.example.com>
admin_server = <hostname.example.com>
kpasswd_server = <hostname.example.com>
}
[domain_realm]
.<example.com> = <EXAMPLE.COM>
[logging]
kdc = FILE:/var/log/kdc.log
admin_server = FILE:/var/log/kadmin.log
default = FILE:/var/log/krb.log
# End /etc/heimdal/krb5.conf
EOF
chmod -v 644 /etc/heimdal/krb5.conf
```

Dans ce script nous avons besoin de substituer :

<EXAMPLE.COM> avec le domaine en majuscule, nous le remplaçons par lfs.

<hostname.example.com> avec le nom du hôte.nom du domaine c'est-à-dire lfs.enp.edu.dz.

`encrypt = true` fournit un cryptage de tout le trafic entre clients et serveurs Kerberos.

Le paramètre `[realms]` indique aux programmes client où chercher les services d'authentification KDC.

La section `[domain_realm]` fait correspondre le domaine au realm.

Stockez le mot de passe dans un fichier clé avec la commande ci-dessous.

```
install -v -m755 -d /var/lib/heimdal &&
kstash
```

Création de la base de données KDC :

```
kadmin -l
```

Finalement, nous installons le script de démarrage `/etc/rc.d/init.d/heimdal` inclus dans le paquetage blfs-bootscripts .

```
make install-heimdal
```

III.8.2.5 STUNNEL

Stunnel est un outil libre permettant de créer un tunnel sécurisé entre deux machines. En choisissant le protocole et le port d'écoute sur lequel on veut que les données soient cryptées. Stunnel nécessite la création de certificats SSL (X509) pour fonctionner, ce qui permet au client et au serveur de s'identifier mutuellement de manière sûre.

La figure suivante schématise le fonctionnement de stunnel.

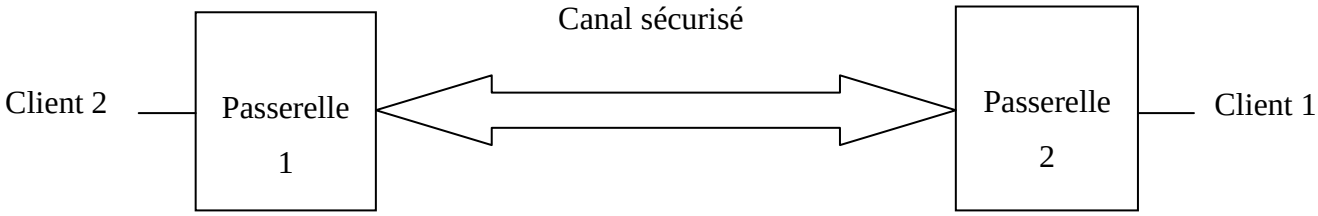

Figure II.10 *Fonctionnement de stunnel.*

Nous avons choisi d'installer le paquet stunnel-4.15.tar.bz2.L'installation de stunnel doit être faite par un utilisateur non privilégié. Pour cela, nous créons le groupe stunnel et l'utilisateur stunnel.

```
groupadd -g 51 stunnel &&
useradd -c "Stunnel Daemon" -d /var/lib/stunnel \
-g stunnel -s /bin/false -u 51 stunnel &&
install -v -m 1770 -o stunnel -g stunnel -d /var/lib/stunnel/run
```

Nous configurons stunnel en :

- Forçant la configuration dans /etc (--sysconfdir=/etc).
- Indiquant l'emplacement des donnée dans /var/lib
- Désactivant libwrap si tcpwrappers n'est pas encours installé.

Puis nous lançons la compilation.

```
./configure --prefix=/usr --sysconfdir=/etc \
--localstatedir=/var/lib --disable-libwrap &&
make
make install
```

Configuration de stunnel

Nous créons un fichier de configuration basique de stunnel en utilisant les commandes suivantes :

```
cat >/etc/stunnel/stunnel.conf << "EOF" &&
# File: /etc/stunnel/stunnel.conf
pid = /run/stunnel.pid
chroot = /var/lib/stunnel
client = no
setuid = stunnel
setgid = stunnel
EOF
chmod -v 644 /etc/stunnel/stunnel.conf
```

Dans ce qui suit, nous allons expliquer les lignes de ce fichier de configuration :

- **pid :** défini le fichier contenant le numéro de processus de stunnel.
- **chroot =** répertoire de chroot du processus stunnel.
- **client =** yes | no

  yes : activer le mode client (Le service distant utilise SSL).

  no : par défaut (mode server).
- **setgid =** nom (Unix seulement)

  Nom de groupe utilisé en mode daemon (les éventuels autres noms de groupe attribués sont supprimés).
- **setuid =** nom (Unix seulement).
- Nom d'utilisateur utilisé en mode daemon.

```
[<service>]
accept = <hostname:portnumber>
connect = <hostname:portnumber>
```

Les commandes ci-dessus permettent d'accepter des connexions sur le port spécifié et se connecter au port spécifié.

Nous terminons par l'installation du script de démarrage /etc/rc.d/init.d/stunnel inclus dans le paquetage blfs-bootscripts .

```
make install-stunnel
```

CONCLUSION

Dans ce chapitre, nous avons mis en évidence les éléments nécessaires pour la construction de notre distribution .Nous avons intégré des utilitaires nous permettant d'utiliser notre distribution en toute sécurité. Dans le chapitre suivant, nous allons orienter cette distribution pour en faire une passerelle.

CHAPITRE 3

PARE-FEU

III.1 INTRODUCTION

Chaque ordinateur connecté à Internet (et d'une manière plus générale à n'importe quel réseau) est susceptible d'être victime d'une intrusion pouvant compromettre l'intégrité du système ou bien altérer les données.

Les pirates informatiques, ayant l'intention de s'introduire dans les systèmes informatiques, recherchent dans un premier temps des failles, c'est-à-dire une vulnérabilité nuisible à la sécurité du système, dans les protocoles, les systèmes d'exploitations, les applications, voire même le personnel d'une organisation. Ils scrutent donc le réseau (en envoyant des paquets de données de manière aléatoire) à la recherche d'une machine connectée (machine cible), puis cherchent une faille de sécurité afin de l'exploiter et d'accéder aux données s'y trouvant.

Dans ce chapitre nous allons mettre en place un système qui permet à la passerelle de contrer ces intrusions.

III.2 LE PARE-FEU

Le pare-feu (firewall) est un système qui protège le réseau local du monde extérieur. Il fait la surveillance de tout trafic entrant/sortant du réseau local, afin de bloquer la circulation des paquets de données, en analysant les informations contenues dans les couches 3,4 et 7 du modèle OSI .Le pare-feu est une machine comprenant au minimum deus interfaces :une pour le réseau externe et une autre pour le réseau à protéger.

La figure suivante présente un pare-feu avec trois interfaces réseau. Une interface avec le réseau local une autre avec le serveur (DMZ) et la dernière avec l'internet.

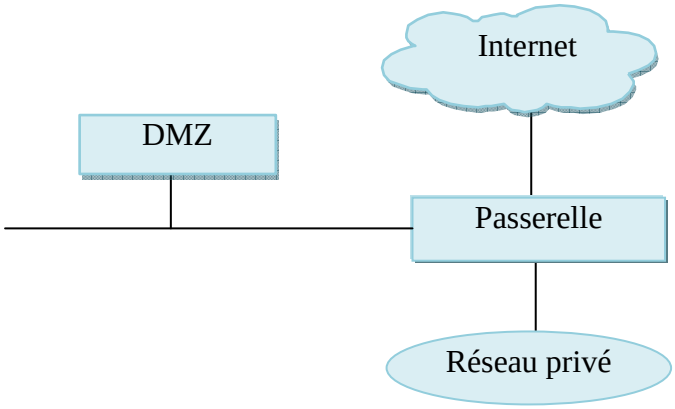

Figure III.1 *schéma d'un pare-feu avec trois interfaces.*

III.3 NECESSITE D'UN PARE-FEU

Le pare-feu est utilisé pour les raisons suivantes:

- **Se protéger de l'extérieur :** le pare-feu permet de se protéger contre les malveillances externes, les pare-feu permettent d'écarter divers intrus comme:
  - les pirates qui cherchent à saboter (saturation des liaisons, corruption de données);
  - les espions (problème de confidentialité de l'information).
- Se protéger de l'intérieur : pour éviter la fuite de l'information non contrôlée vers l'extérieur.
- Contrôler les flux : permet de surveiller tous les flux de trafic entre le réseau internet et le réseau externe. Afin d'interdire l'accès à certains sites et interdire les spams.
- Faciliter l'administration du réseau : sans pare-feu, chaque machine du réseau est potentiellement exposée aux attaques d'autres machines d'Internet. Le pare-feu simplifie la gestion de la sécurité, donc l'administration du réseau, car il centralise les attaques au niveau du pare-feu plutôt que sur le réseau tout entier.

Il existe deux types de filtrage pour le un pare-feu : le filtrage de paquets et le filtrage applicatif.

- ❖ Le filtrage de paquets : les paquets de données passant par un pare-feu contiennent les entêtes suivants:

- l'adresse IP de la machine émettrice du paquet;
- l'adresse IP de la machine réceptrice du paquet;
- le numéro du port.

Le filtrage de paquets peut être basé soit sur le filtrage par adresse lorsque l'adresse IP est analysée, soit sur le filtrage par protocole lorsque le type de paquets et le port sont analysés.

- ❖ Le filtrage applicatif : il permet de filtrer les communications application par application c'est à dire que ce type de filtre travail au niveau de la couche sept du modèle OSI. Le filtrage applicatif suppose une connaissance de l'application.

III.4 NETFILTER

Netfilter est le nom d'une partie du kernel Linux qui est destinée à assurer la surveillance de tous les transferts de données réseaux. Sa tâche est de faire du "Network Packet Filtering", c'est à dire du "Filtrage de Paquets Réseaux". Netfilter supporte actuellement les protocoles IPv4, IPv6, DECnet, et ARP, et en partie IPX via des patchs expérimentaux. Nous ne nous intéresserons ici qu'à IPv4.

Comme Netfilter est un élément implanté profondément dans le noyau Linux, on ne peut le configurer aussi simplement qu'avec une interface graphique. On ne peut pas non plus le paramétrer directement via un fichier de configuration du "/etc/". L'unique moyen que nous avons de dialoguer avec lui est un programme appelé "iptables".

En fait, netfilter se présente comme une série de cinq hooks, sur lesquels des modules de traitement de paquets vont se greffer. Ces points sont:

NF_IP_ROUTING;

NF_IP_LOCAL_IN;

NF_IP_LOCAL_OUT;

NF_IP_FORWARD;

NF_IP_POSTROUTING.

Ces cinq hooks sont associé à cinq chaines qui seront présentées ultérieurement.

III.4.1 LES CHAINES

Netfilter utilise cinq types de chaines pour distinguer tous les flux qui ont une relation avec le firewall :

INPUT: tout flux destiné au firewall.

OUTPUT: tout flux sortant du firewall.

FORWARD: tout flux traversant le firewall et n'est pas destiner au firewall.

PREROUTING: tout flux qui vient d'enter au firewall.

POSTROUTING: tout flux qui vient de sortir du firewall.

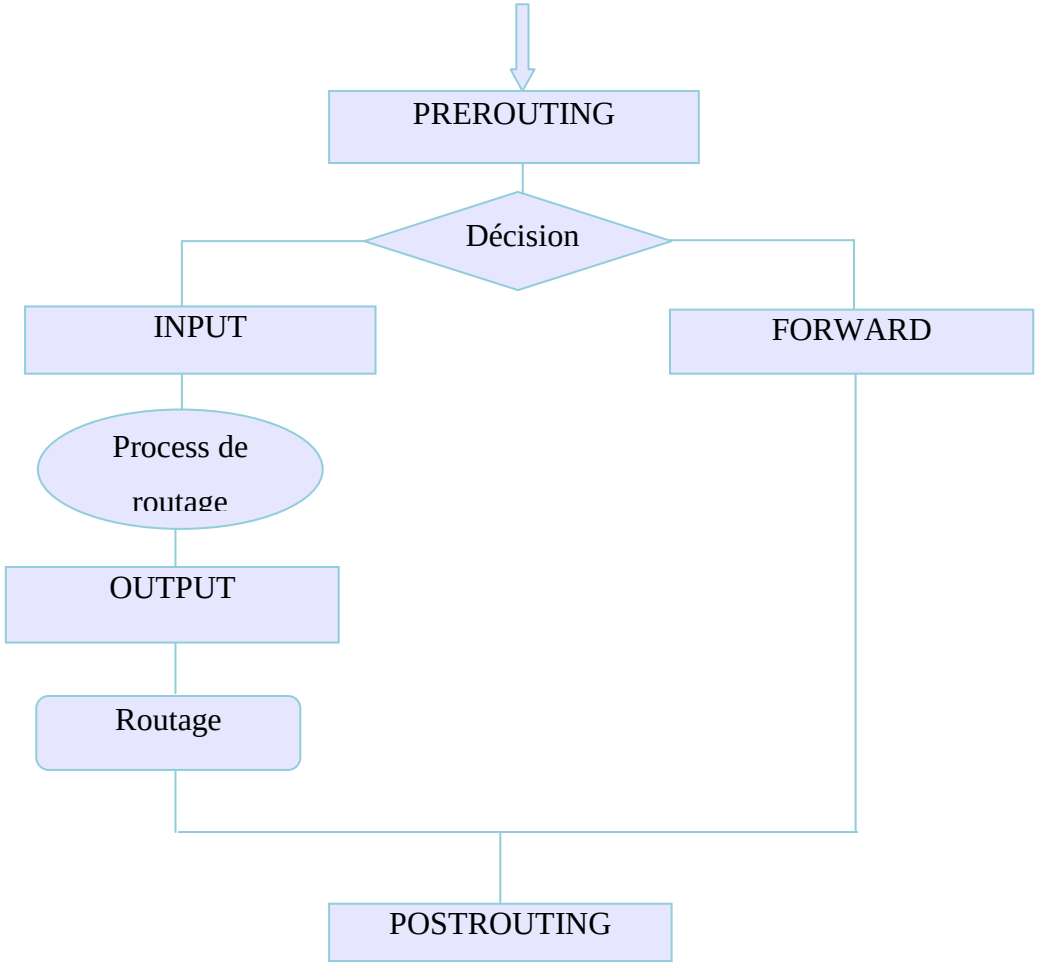

Figure III.2 *Fonctionnement de Netfilter.*

Linux utilise les "chaînes" pour stocker les règles de filtrage. Un paquet est envoyé dans une chaîne où il est filtré en fonction des règles qui y ont été insérées.

Un paquet arrive sur notre pare-feu. Il arrive tout d'abord dans le nœud PREROUTING où l'on peut exécuter certaines tâches (comme la translation d'adresses ou nattage).

Ensuite, si le paquet est à destination d'une adresse de notre pare-feu, alors le paquet est envoyé dans la chaîne "INPUT". Sinon le paquet est envoyé dans la chaîne "FORWARD".

Si aucune règle n'a rejeté le paquet dans la chaîne INPUT, alors le paquet est traité par les processus locaux (par exemple un serveur WEB) et envoyé dans la chaîne "OUTPUT" où il est filtré de nouveau.

Enfin les paquets des chaînes FORWARD et OUTPUT sont envoyés dans le nœud "POSTROUTING" où l'on peut exécuter certaines tâches (comme la translation d'adresses).

Un paquet est envoyé sur la chaîne FORWARD quand les adresses IP ne correspondent pas à l'une des adresses de notre serveur.

III.4.2 PRINCIPES DE NETFILTER

Selon les traitements à appliquer sur le flux, Netfilter utilise trois types de tables dans chacune à un traitement particulier sera appliquée si la condition est vérifiée :

- Filter : comme son nom l’indique, cette table sert à filtrer les paquets réseaux. C’est à dire que nous allons pouvoir trier les paquets qui passent à travers le réseau, et supprimer ceux qui ne nous intéressent pas, ou que nous trouvons dangereux.
- Nat : Pour la translation d’adresses (redirection de port, d'adresse,...).Grace à cette table nous pouvons transformer notre machine Linux en une passerelle.
- Mangle : Pour la modification des paquets. L’idée de cette technique est par exemple de fournir à Linux la possibilité d’avoir un contrôle sur les débits des flux de données entrants et sortants de la machine, afin de rendre certains flux plus prioritaires que d’autres. Certains flux vont être volontairement privilégiés, afin de garantir un meilleur confort d’utilisation à l’utilisateur. Cette technique s’appelle le QoS.

III.4.3 REGLES DU FIREWALL

Les règles, comme leur nom l'indique, sont une série de critères auquel doivent ou non répondre les paquets. Si le paquet réseau ressemble à l'un ou l'autre des critères, alors la règle est appliquée. Les différentes règles d'une chaînes sont appliquées les unes à la suite des autres.

Les critères peuvent être multiples :

- Interface source ou destination.
- Adresse IP source ou de destination.
- Port source ou de destination.
- Type de trame.
- Nombre de paquets.
- Paquet marqué par la table Mangle.

  Etc.

Il peut y avoir autant de règles que l'on veut dans une chaîne.

Enfin, à chaque règle est associée une action (ou "CIBLE" dans la nomenclature de Netfilter) à effectuer si la règle doit s'appliquer. C'est là que Netfilter agit, qu'il fait (enfin) quelque chose avec le paquet réseau. Les principales actions sont :

ACCEPT: le paquet est accepté.

DROP: le paquet est écrasé sans avertir la source.

QUEUE: le paquet est mis dans une file d'attente.

RETURN: retour à la cible appelante.

LOG / ULOG : Le paquet est autorisé à continuer de passer, mais ses caractéristiques sont notées au passage. En général, c'est qu'on estime que le paquet est "louche", et que l'on veut en prendre note. Mais plus souvent encore, on décidera de le supprimer. Car, en informatique, tout ce qui est louche est anormal, et tout ce qui est anormal doit être supprimé.

MARK: le paquet est marqué en y attachant une information.

REJECT: le paquet est écrasé avec avertissement.

SNAT: l'adresse IP source est substituée par une autre.

DNAT: l'adresse IP destination est substituée par une autre.

MASQUERADE: Le paquet va être modifié, afin de dissimuler (de masquer en fait) certaines informations concernant son origine.

III.4.4 EXEMPLES D'UTILISATION DE LA COMMANDE IPTABLES

Lister les règles.

#iptables -L

Supprimer les tables prédéfinies (option " -F") et toutes les tables utilisateurs (option "-X").

iptables -t filter -F

iptables -t filter -X

La table filter possède 3 chaînes, donc elles sont toutes les trois à initialiser. Par défaut, on décide donc de tout détruire (DROP) :

iptables -t filter -P INPUT DROP

iptables -t filter -P OUTPUT DROP

iptables -t filter -P FORWARD DROP

Pour ajouter une règle;

#iptables -A INPUT -p tcp --drop www -j ACCEPT

Supprimer une chaine en spécifiant son numéro.

iptables -D INPUT --drop 80 -j DROP iptables -D INPUT 1

III.5 SHOREWALL

III.5.1 PRINCIPE DU SHOREWALL

"**Shoreline Firewall**", plus communément appelé "**Shorewall**", est un outil pour configurer plus facilement *Netfilter*. Shorewall peut être utilisé sur un serveur Linux en routeur/firewall, sur une gateway/routeur/serveur multifonction ou sur un système GNU/Linux autonome.

III.5.2 METHODE D'INSTALLATION DE SHOREWALL

Les paquets utilisés sont :

- shorewall.common-4.0.0.tar.bz2 (common files).
- shorewall.shel-4.0.0.tar.bz2 (shel compiler)
- shorewall. perl-4.0.0.tar.bz2 (perl compiler).

Nous avons le choix d'utiliser l'un des compilateurs, comme on peut utiliser les deux à la fois.

La méthode d'installation de ces paquets diffère de celle utilisée précédemment. Pour les trois paquets nous allons exécuter le script *install.sh.*Nous allons illustrer la méthode d'installation pour le premier paquet.

```
cd   shorewall.common-4.0.0
. /install.sh
```

Pour vérifier si la configuration (que nous aborderons ultérieurement) du firewall est compatible avec la version de shorewall nous utilisons `make check`

Si il n'ya aucune erreur, nous lançons le shorewall : `shorewall start`

Pour l'arrêter il suffit de taper `shorewall stop`

III.6 APPLICATION CONFIGURATION DU SHOREWALL

Configurer le shorewall revient à modifier ces fichiers de configuration ces derniers se trouvent dans le répertoire */etc/shorewall* pour la plus part des paramétrages, on a juste besoin de quelques-uns d'entre eux. Des squelettes de fichiers sont créés durant La procédure d'installation de Shorewall.

Shorewall voit le réseau ou il opère comme composé d'un ensemble de zones qui sont définies dans */etc/shorewall/zones.*

Le Shorewall peut posséder plusieurs interfaces réseaux selon son utilisation, pour bien expliquer nous allons illustrer par le schéma suivant qui contient :

- La DMZ est utilisée pour isoler nos serveurs accessibles depuis Internet des systèmes locaux.
- La zone Local est composée des systèmes Local 1, Local 2 et Local 3…Local 13 (dans notre cas).
- Tous les systèmes du FAI vers l'extérieur et qui englobe la Zone Internet

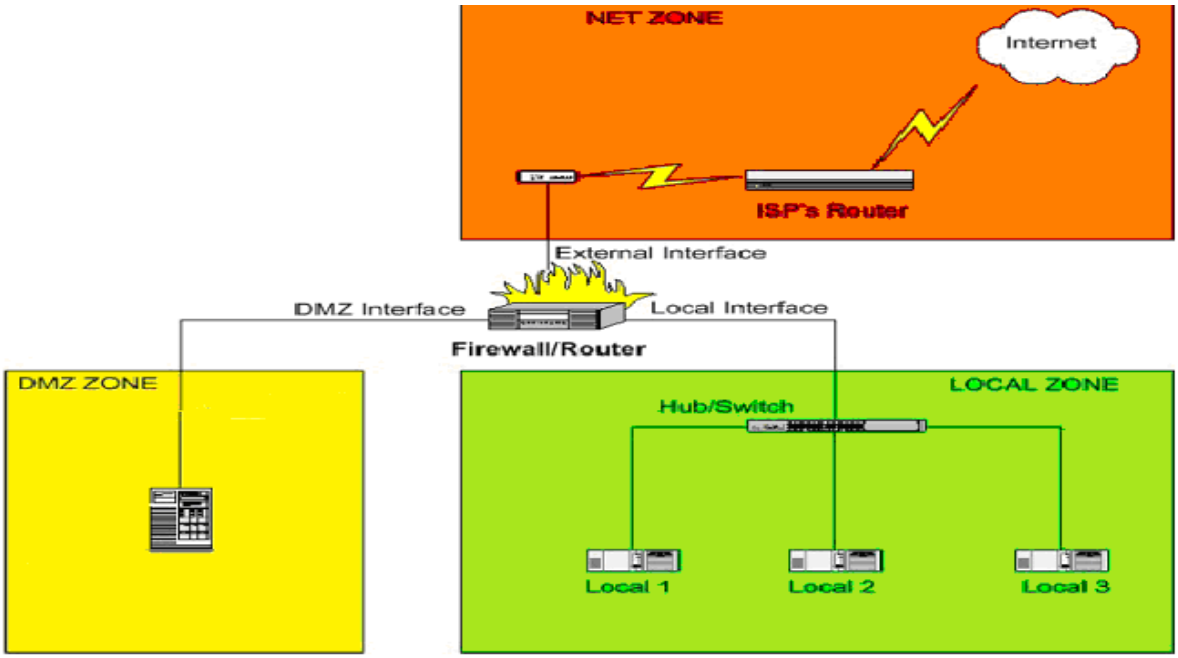

Figure III.3 *Configuration d'un Shorewall à plusieurs interfaces.*

Remarques

- Si notre Interface Externe est ppp0 ou ippp0 alors on peut fixer CLAMPMSS=yes dans */etc/shorewall/shorewall.conf*
- Notre *Interface Locale* sera un adaptateur Ethernet (eth0, eth1 or eth2) et doit être connecté à un hub ou un switch. Les ordinateurs locaux doivent être connectés au même switch (note: Si on a une machine unique, on peut connecter le firewall directement à l'ordinateur en utilisant un **câble croisé**).
- Normalement, la FAI nous assigne des adresses Publiques. On peut configurer l'interface externe du Shorrewall en utilisant l'une de ces adresses permanentes et on peut décider comment utiliser le reste de nos adresses (Adresse IP et routage).

III.6.1 LE FICHIER /ETC/SHOREWALL/ZONES

Notre exemple de configuration de zones est donné dans le tableau suivant :

Nom	Description
Net	Internet
Loc	Local Network
Dmz	Zone Démilitarisée

Tableau III.1 *configuration des zones.*

/etc/shorewall/zones : Permet de définir les zones que nous allons utiliser dans la configuration de shorewall.

#ZONE	TYPE	OPTIONS	IN OPTIONS	OUT OPTIONS
$Fw	Firewall			
Net	ipv4			
Loc	ipv4			
Dmz	ipv4			

Remarques

- Shorewall reconnaît le firewall comme sa propre zone par défaut, le firewall est connu comme **fw** cela peut être modifié dans le fichier */etc/shorewall/shorewall.conf* .
- Mise à part **fw**, Shorewall n'attache aucune importance au nom des zones. Les Zones sont définies par l'utilisateur.

Les règles concernant le trafic à autoriser ou à interdire sont exprimées en utilisant les termes de zones :

- On peut exprimer notre politique par défaut pour les connexions d'une zone vers une autre zone dans le fichier */etc/shorewall/policy*.
- On peut définir les exceptions à ces politiques par défaut dans le fichier */etc/shorewall/rules.*

III.6.2 LE FICHIER /ETC/SHOREWALL/INTERFACES

Permet de définir à quoi correspondent les interfaces réseaux. Dans notre exemple on va utiliser : eth0, eth1, eth2.

ZONE	INTERFACE	BROADCAST	OPTIONS
Net	Eth0	Detect	tcpflags,dhcp,routefilter,nosmurfs,logmaartians
Loc	Eth1	Detect	tcpflags,nosmurfs
Dmz	Eth2	Detect	

Si une zone communique avec plus d'une interface, on inclue simplement une entrée pour chaque interface et on répète le nom de zone autant de fois que nécessaire.

Dans la réalité, généralement on ne peut se permettre autant d'adresses IP Publiques que de périphériques à assigner si bien que nous utiliseront des adresses IP Privées. RFC 1918 réserve plusieurs plages d'adresse IP à cet usage:

10.0.0.0	-	10.255.255.255
172.16.0.0	-	172.31.255.255
192.168.0.0	-	192.168.255.255

Remarques

- On peut définir des zones plus compliquées en utilisant le fichier */etc/shorewall/hosts* mais dans la plus part des cas, ce n'est pas nécessaire.
- Si les connexions d'un certain type sont autorisés de la zone A au firewall et sont aussi autorisés du firewall à la zone B cela ne veut pas dire que ces connections sont autorisés de la zone A vers la zone B. Cela veut plutôt dire qu'on a un Proxy qui tourne sur le firewall qui accepte les connections de la zone A et qui ensuite établit ces propres connections du firewall à la zone B.
- Pour chaque requête de connexion sur le firewall, la requête est d'abord évalué à travers le fichier */etc/shorewall/rules* file. Si aucune règle dans ce fichier ne correspond, la connexion interroge ensuite la première police dans */etc/shorewall/policy* qui correspond à la requête et l'applique. Si cette police est REJECT ou DROP, la requête est a nouveau évaluée à travers les règles du fichier */etc/shorewall/common.de*

III.6.3 LE FICHIER /ETC/SHOREWALL/POLICY

Permet d'accepter ou de rejeter le trafic entre les différentes zones. On peut illustrer par :

#Source	Dest	Policy	LOG LEVEL	LINMIT:BURST
Loc	net	ACCEPT		
Loc	dmz	REJECT	Info	
Loc	$FW	REJECT	Info	
Loc	All	REJECT	Info	
$FW	net	REJECT	Info	
$FW	dmz	REJECT	Info	
$FW	loc	REJECT	Info	
$FW	all	REJECT	Info	
dmz	net	REJECT	Info	
dmz	$FW	REJECT	Info	
dmz	loc	REJECT	Info	

III.6.4 LE FICHIER ETC/SHOREWALL/RULES

Ceci est le fichier de configuration de la partie firewall. C'est à nous de le configurer selon nos besoins. Nous pouvons prendre comme exemple :

#DEST	LIMIT	GROUP
DNS/ACCEPT	$FW	Net
SSH/ACCEPT	Loc	$FW
SSH/ACCEPT	Loc	Dmz
DNS/ACCEPT	Dmz	Net
Ping/REJECT	Net	$FW
Ping/ACCEPT	Net	$FW
Ping/ACCEPT	Loc	$FW

III.6.5 LE FICHIER ETC/SHOREWALL/ROUTESTOPPED

Permet le routage sur les interfaces définis dans ce fichier lorsque shorewall n'est pas démarré.

Par défaut shorewall ne peut pas démarrer tant que nous n'avons pas modifié le fichier /etc/default/shorewall et changer la ligne :

Startup=0 Par **Startup=1**

Configuration du réseau

Le choix de configuration de notre réseau dépend d'abord du nombre d'adresses Public IP dont on a besoin, c'est à dire du nombre d'entités adressables qu'on a sur notre réseau. En fonction du nombre d'adresses qu'on a déjà, notre FAI peut servir ce jeu d'adresses de deux manières:

- **Routed** : Le trafic vers chacune de nos adresses sera routé à travers une unique adresse passerelle. Cela sera généralement fait si notre FAI nous assigne un sous réseau complet. Dans ce cas, nous assignerons l'adresse passerelle comme adresse IP de l'interface externe de notre firewall/router.
- **Non-Routed** : Notre FAI nous donnera directement le trafic de chaque adresse.

III.6.6 IP MASQUERADING (SNAT)

Les adresses réservées par la RFC 1918 sont parfois désignées comme **non-routables** car les routeurs Internet (*backbone*) ne font pas circuler les paquets qui ont une adresse de destination appartenant à la RFC-1918. Lorsqu'un de nos systèmes en local demande une connexion à un serveur par Internet, le firewall doit appliquer un ***Network Address Translation*** **(**NAT**)**. Le firewall réécrit l'adresse source dans le paquet, et la remplace par l'adresse de l'interface externe du firewall; en d'autres mots, le firewall fait croire que c'est lui même qui initie la connexion. Ceci est nécessaire afin que l'hôte de destination soit capable de renvoyer les paquets au firewall. Lorsque le firewall reçoit le paquet de réponse, il remet l'adresse de destination et fait passer le paquet vers l'ordinateur qui a demandé la connexion.

Il faut noter aussi que Shorewall suit la convention utilisée avec Netfilter :

- ***MASQUERADE* :** désigne le cas ou nous laissons notre firewall détecter automatiquement l'adresse de l'interface externe.
- ***SNAT* :** désigne le cas où on spécifie explicitement l'adresse source des paquets sortant de notre réseau local.

Sous Shorewall, autant le *Masquerading* et le *SNAT* sont configurés avec des entrées dans le fichier */etc/shorewall/masq*. Nous utiliserons normalement le Masquerading si notre adresse IP externe est dynamique, et SNAT si l'adresse IP est statique. On illustre par l'exemple :

#Interface	Subnet	Adress
Eth0	192.168.201.0	192.0.2.176

III.6.7 PORT FORWARDING (DNAT)

Quand SNAT est utilisé, il est impossible pour les hôtes sur Internet d'initialiser une connexion avec un des systèmes puisque ces systèmes n'ont pas d'adresses publiques IP, DNAT fournit une méthode pour autoriser des connexions sélectionnées depuis Internet.

Ce procédé est appelé *Port Forwarding* or *Destination Network Address Translation* (DNAT). On configure le port forwarding en utilisant les règles DNAT dans le fichier */etc/shorewall/rules.*

III.6.8 PROXY ARP

Sa configuration est faite dans le fichier */etc/shorewall/proxyarp* et son principe est le suivant :

- Un hôte **H** derrière le firewall est assigné à une de nos adresses publiques (**A**), a le même masque de réseau (**M**) que l'interface externe du firewall.
- Le firewall répond à ARP "qui a" demandé **A**.
- Quant **H** délivre une requête ARP "qui a" pour une adresse du sous-réseau définit par **A** et **M**, le firewall répondra (avec l'adresse MAC si le firewall s'interface à **H**).

CONCLUSION

Dans ce dernier chapitre, nous avons réussi à orienter notre distribution Linux en une passerelle .Cet objectif a été atteint en installant et configurant shorewall selon nos besoin. Dans notre cas, nous avons illustré cette étape, en donnant un exemple de configuration du shorewall adapté pour trois interfaces réseau.

CONCLUSION ET PERSPECTIVES

Le travail qui a nous a été assigné s'inscrit dans la conception et la réalisation d'une passerelle Linux pouvant répondre à diverses fonctionnalités inhérentes à l'accès au réseau internet à partir d'un campus universitaire. L'idée directrice a été une réalisation à partir d'éléments constitutifs dont le noyau linux. Ce qui définit parfaitement la notion de distribution.

Notre contribution à ce projet peut être classée en deux parties :

- la réalisation d'une distribution de base
- mise en place des premières briques pour sa personnalisation

La distribution obtenue peut être qualifiée de base. En plus de servir au projet sus cité, elle pourra être un prélude à diverses orientations possibles : poste client, serveur et autres. On soulignera pour les électroniciens, l'intérêt particulier et grandissant de l'intégration du système linux dans la conception des systèmes embarqués. Notre réalisation pourra être l'élément de base d'une telle construction.

Le haut degré de sécurisation de Linux nous a permis d'orienter notre distribution vers la sécurité réseau. Nous avons fait de notre distribution un firewall permettant de se protéger des malveillances externes et d'autoriser que les communications ayant été explicitement autorisées ou d'interdire les échanges qui ont été explicitement interdits.

Pour la mise en œuvre d'une politique de sécurité locale à un serveur, deux approches sont envisageables :

Iptables : nous avons vu que la définition des règles de filtrage est une tâche plus difficile.

Shorewall : avec ce dernier la configuration du pare-feu a été plus facile vu le niveau d'abstraction qu'il présente.

Nous avons privilégié la deuxième approche vue le niveau d'abstraction qu'il présente.

Diverses extensions restent à faire. Nous citons les divers proxy, l'intégration de divers caches (dns, arp, html, ..), le filtrage applicatif, la détection d'intrusion, la protection

antivirale, …. Une interface web couplée à son serveur pourra servir pour une configuration aisée de la passerelle ainsi obtenue.

ANNEXE

A.1 Système d'exploitation Linux

A.1.1 Caractéristiques générales de Linux

- **Linux est gratuit**

  On peut installer un système Unix complet sans autre coût que celui du matériel.

- **Linux est entièrement paramétrable dans tous les composants**

  Grâce à la licence publique GNU (GPL), on peut librement lire et modifier le code source du noyau et de tous les programmes du système.

- **Linux fonctionne sur des machines bas de gamme bon marché**

  On peut même mettre en place un serveur réseau sur un vieux système à base d'Intel

  80386 avec 4Mo de RAM.

- **Linux est puissant**

  Les systèmes Linux sont très rapides car ils exploitent pleinement les caractéristiques de composants matériels. L'objectif principal de Linux est l'efficacité et en fait de nombreux choix de conception faits par des Unix commerciaux, ont été refusés par Linus du fait des pertes de performances qu'ils impliquaient.

- **Linux possède un code source d'une excellente qualité**

  Les systèmes Linux sont généralement très stables ; ils ont un taux d'erreur et un temps de maintenance très faibles.

- **Le noyau Linux peut être très petit et compact**

  En fait il est possible de faire tenir une image du noyau et un système de fichiers complet, comprenant tous les programmes fondamentaux du système sur un espace mémoire très réduit.

- **Linux est compatible avec beaucoup de systèmes d'exploitation classiques**

  Cela permet par exemple de monter directement des systèmes de fichiers pour toutes les versions de MS-DOS et MS Windows, SVR4, Solaris, SunOS, etc. linux peut également utiliser de nombreux matériels réseau comme Ethernet, FDDI *(Fiber Distributed Data Interface)*, HIPPI *(High Performance Parallel Interface),* etc. Grâce à l'utilisation de librairies, les systèmes Linux sont même capables d'exécuter

directement des applications compilées pour d'autres systèmes d'exploitation. Linux peut ainsi exécuter, par exemple, des applications écrites pour MS-DOS , MS Windows,SVR3 et R4, et d'autres systèmes fonctionnant sur plateforme Intel 80x86.

- **Linux bénéficie d'un bon support**

  Il est parfois plus simple d'obtenir des modifications (patches) et des mises à jour pour Linux que pour un système propriétaire.

A.1.2 Linux face aux autres noyaux Unix

- **Le noyau Linux est monolithique.** C'est un gros programme complexe, constitué de différentes composantes dotées de leurs logiques propres. Il est en cela assez conventionnel, la plupart des Unix commerciaux étant monolithiques. Une exception notable est le système Mach 3.0 de Carnegie-Mellon, qui suit une approche à base micronoyau.

- Les noyaux Unix traditionnels sont compilés et liés statiquement .la majorité des noyaux modernes peuvent dynamiquement charger et supprimer des parties du code du noyau (typiquement des pilotes de périphériques) que l'on appelle habituellement ***modules***. la gestion des modules est excellente dans Linux puisqu'il est capable de charger et supprimer les modules automatiquement à la demande. parmi les variantes commerciales d'Unix, seul le noyau SVR4.2 offre une telle possibilité.

- **Support multiprocesseur**. Certaines versions du noyau Unix tirent partie de systèmes multiprocesseurs. Linux 2.2 offre un support pour les systèmes multiprocesseurs symétriques (SMP), ce qui signifie non seulement que le système peut utiliser plusieurs processeurs, mais également que chaque processeur peut traiter n'importe quelle tâche, sans discrimination. cependant, Linux 2.2 n'utilise pas le SMP de façon optimale. différentes activités du noyau qui pourraient être exécutées de façon concurrente (comme la gestion de fichiers ou du réseau) doivent actuellement être exécutées de façon séquentielle.

- **Les systèmes de fichiers.** Certaines fonctionnalités avancées font défaut au système de fichiers standard de Linux, comme la journalisation. Cependant, des systèmes de fichiers plus avancés sont disponibles sous Linux, même s'ils ne sont pas inclus dans son code source. Grâce à la puissance d'une technologie de système de fichiers virtuel orienté objet (inspiré par Solaris et SVR4), porter sous Linux un système de fichiers étranger est une tâche relativement aisée.

A.1.3 Structure d'un système de fichiers sur Linux

Le tableau ci-dessus présente la structure d'un système de fichier sous Linux.

Répertoire	Signification
/	Répertoire racine. Point de départ de toute la hiérarchie du système de fichiers. Le système de fichiers contenant ce répertoire est monté automatiquement par le noyau pendant l'amorçage du système. Ce système de fichiers est appelé système de fichiers racine.
/proc/	Répertoire contenant le pseudo système de fichiers du noyau. Ce pseudo système de fichiers contient des fichiers permettant d'accéder aux informations sur le matériel, la configuration du noyau et sur les processus en cours d'exécution.
/dev/	Répertoire contenant tous les fichiers spéciaux permettant d'accéder aux périphériques. Sous Linux la plupart des périphériques sont accessibles à travers des fichiers spéciaux.
/bin/	Répertoire contenant les commandes systèmes générales nécessaires à l'amorçage. Ce répertoire doit être impérativement placé dans le système de fichiers racine. Tous les utilisateurs peuvent utiliser les commandes de ce répertoire.
/boot/	Répertoire contenant le noyau de Linux et ses informations de symboles.
/usr/	Répertoire contenant les fichiers du système partageables en réseau et en lecture seule.
/etc/share/	Répertoire contenant tous les fichiers de configuration du système. Ce répertoire doit être impérativement placé dans le système de fichiers racine.
/etc/src/	Répertoire contenant les fichiers sources du noyau et des applications de la distribution. Normalement, ce répertoire ne doit contenir que le code source des applications dépendantes de la distribution qu'on utilise.
/home/	Répertoire contenant les répertoires personnels des utilisateurs.

/opt/	Répertoire contenant les applications. C'est dans ce répertoire que les applications qui ne font pas réellement partie du système doivent être installées. Les applications graphiques devraient être installées dans ce répertoire. C'est en particulier le cas des gestionnaires de bureau. Les seules applications graphiques considérées comme faisant partie du système sont les applications de X11, qui sont donc stockées dans /usr/X11R6/. Il est recommandé que ce répertoire soit placé sur un système de fichiers en lecture seule, et que les applications utilisent le répertoire /var/opt/ pour travailler.
/etc/	Répertoire contenant tous les fichiers de configuration du système. Ce répertoire doit être impérativement placé dans le système de fichiers racine.
/tmp/	Répertoire permettant de stocker des données temporaires.
/sbin/	Répertoire contenant les commandes systèmes nécessaires à l'amorçage et réservées à l'administrateur. Ce répertoire doit être impérativement placé dans le système de fichiers racine. En général, seul l'administrateur utilise ces commandes.
/lib/	Répertoire contenant les bibliothèques partagées (« DLL » en anglais, pour « Dynamic Link Library ») utilisées par les commandes du système des répertoires /bin/ et /sbin/. Ce répertoire doit être impérativement placé dans le système de fichiers racine.
/mnt/	Répertoire réservé au montage des systèmes de fichiers non permanents (CD-ROM, disquettes, etc.). Ce répertoire peut contenir plusieurs sous-répertoires pour chaque périphérique amovible, afin de permettre d'en monter plusieurs simultanément. Notez qu'il est assez courant de disposer de liens symboliques dans la racine référençant les principaux systèmes de fichiers, afin d'en simplifier l'accès. Par exemple, il est courant d'avoir un répertoire /floppy référençant le lecteur de disquette et un répertoire /cdrom/ référençant le lecteur de CD-ROM.
/root/	Répertoire contenant le répertoire personnel de l'administrateur. Il est donc recommandé que le répertoire personnel de l'administrateur soit placé en dehors de /home/ pour éviter qu'un problème sur le système de fichiers des utilisateurs ne l'empêche de travailler. Toutefois, il est important que l'administrateur puisse travailler même si les répertoires /root/ et /home/root/ ne sont pas présents. Dans ce cas, son répertoire personnel devra être le répertoire racine.
lost+found	Répertoire contenant les données récupérées lors de la réparation d'un système de fichiers endommagé.

A.2 Outils de développement sous Linux

A.2.1 Illustration du fonctionnement de Gcc

Programme à plusieurs fichiers

$vi multi.h

```
void compilation();
```

$vi multi.c

```
#include <stdio.h>
#include "multi.h"
int main()
{
                                        compilation();
   return (0);
}
```

$vi bon.c

```
#include<stdio.h>
void compilation()
{
printf("compilation avec gcc\n");
}
```

Compilation:$gcc -c multi.c bon.c

Edition de lien: $ gcc -o mult multi.o bon.o

Execution :. /mult

compilation avec gcc

A.2.2 Illustration d'utilisation de make

Nous allons compiler le programme de l'exemple précédant.

D'abord nous créons le Makefile: $vi Makefile

```
esx :  multi.o bon.o
        gcc -o esx multi.o bon.o
multi.o : multi.c multi.h
        gcc -c multi.c
bon.o : bon.c multi.h
        gcc -c bon.c
clean:
        rm *.o esx
```

Puis nous lançons la compilation ($make -f Makefile) .maintenant nous pouvons exécuter le programme (. /esx).

A.3 Modèle OSI

La caractérisation donnée ici est tirée de la norme ISO 7498-1. La description originelle donne en plus pour chaque couche les fonctions de manipulation de commandes ou de données significatives parmi celles décrites plus bas.

1. La couche « physique » est chargée de la transmission effective des signaux entre les interlocuteurs. Son service est typiquement limité à l'émission et la réception d'un bit ou d'un train de bit continu (notamment pour les supports synchrones).
2. La couche « liaison de données » gère les communications entre 2 machines adjacentes, directement reliées entre elles par un support physique.
3. La couche « réseau » gère les communications de bout en bout, généralement entre machines : routage et adressage des paquets.(cf. note ci-dessous).
4. La couche « transport » gère les communications de bout en bout entre processus (programmes en cours d'exécution).
5. La couche « session » gère la synchronisation des échanges et les «transactions», permet l'ouverture et la fermeture de session.
6. La couche « présentation » est chargée du codage des données applicatives, précisément de la conversion entre données manipulées au niveau applicatif et chaînes d'octets effectivement transmises.
7. La couche « application » est le point d'accès aux services réseaux, elle n'a pas de service propre spécifique et entrant dans la portée de la norme.

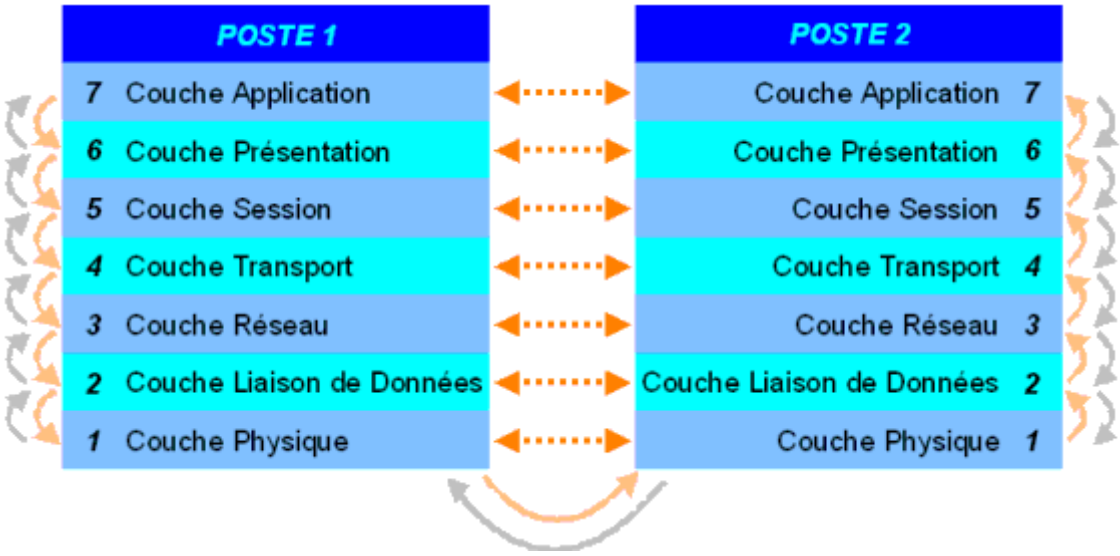

Figure A.3.1 Les différentes couches du modèle

A.4 Distribution Linux

Nous allons citer les distributions les plus connues.

Il s'agit d'une distribution relativement ancienne : les premières versions sont sorties en 1994.

L'apport principal de Red Hat est sans doute le concept de paquetage. Un paquetage comprend un logiciel, sa documentation ainsi que tous les utilitaires qui en simplifient l'installation, la désinstallation ou la mise à jour. Red Hat a ouvert aussi la voie à toutes les distributions utilisant des outils graphiques.

Le projet Fedora a d'abord eu pour but de développer des paquets RPM complémentaires pour la distribution d'entrée de gamme de *Red Hat*. Mais au mois de septembre 2003, *Red Hat*, décidé à se concentrer sur le marché des entreprises, laisse la charge de sa distribution grand public à Fedora.

Mandriva (ancienne Mandarkesoft) est une société française éditrice de la distribution Mandriva Linux. Son installation est simple et ne requière aucune connaissance technique particulière : différents assistants se chargent d'expliquer à l'utilisateur les notions de base et le conseillent pour les choix délicats.

Fondée officiellement par Ian Murdock le 16 août 1993, le projet Debian fut l'une des premières distributions Linux.

Debian est toujours disponible en 3 versions (trois branches) qui sont :

- ***stable*** : version figée où les seules mises à jour sont des correctifs de sécurité.
- ***testing*** : future stable où seuls les paquets suffisamment matures peuvent rentrer.
- ***unstable* :** version active, constamment nourrie de nouveaux paquets ou de mises à jour de paquets déjà existants (surnommée ***Sid***).

Ubuntu est une distribution GNU/Linux basée sur Debian et destinée à proposer un système convivial, ergonomique, libre et gratuit y compris pour les entreprises. Initialement conçu pour tous les ordinateurs de bureau (fixe ou portable).

Les distributions Ubuntu sont toujours des variantes **stables** de la version avancée de *Debian* (SID). L'architecture générale est donc celle de la distribution Debian.

La différence principale est que la convivialité générale (procédure d'installation, choix de logiciels par défaut...) se fait parfois hors du cadre éthique très strict de Debian. Par contre, les versions stables de Debian suivantes intègrent ou adaptent certaines des avancées mises en place pour Ubuntu.

A la différence de la plupart des distributions Linux, le compte ***root*** ou administrateur est désactivé par défaut pour faciliter l'administration du système (choix donnant souvent lieu à des polémiques concernant la sécurité); c'est donc l'utilisateur qui effectue les tâches administratives temporairement et pour une tâche déterminée avec les droits d'administrateur (ou *super utilisateur*).

A.5 Les paquets installés

Pour la construction de LFS nous avons besoin des paquets suivants.

- Autoconf (2.61)
- Automake (1.10)
- Bash (3.2)
- Bash Documentation
- KB: Berkeley DB
- Binutils (2.17)
- Bison (2.3)
- Bzip2 (1.0.4
- Coreutils (6.9)
- DejaGNU (1.4.4)
- Diffutils (2.8.1)
- E2fsprogs (1.40.2)
- Expect (5.43.0
- File (4.21)
- Findutils (4.2.31)
- Flex (2.5.33)
- GCC (4.1.2)
- Gettext (0.16.1)
- Glibc (2.5.1)
- Glibc LibIDN add-on (2.5.1)
- Grep (2.5.1a)
- Groff (1.18.1.4)
- GRUB (0.97)
- Gzip (1.3.12)
- Iana-Etc (2.20)
- Inetutils (1.5)
- IPRoute2 (2.6.20-070313
- Kbd (1.12)
- Less (406)
- LFS-Bootscripts (6.3)
- Libtool (1.5.24)
- Linux (2.6.22.5)
- M4 (1.4.10)
- Make (3.81)
- Man-DB (2.4.4)
- Man-pages (2.63)
- Mktemp (1.5)

- Module-Init-Tools (3.2.2)
- Ncurses (5.6)
- Patch (2.5.4)
- Perl (5.8.8)
- Procps (3.2.7)
- Psmisc (22.5)
- Readline (5.2)
- Sed (4.1.5)
- Shadow (4.0.18.1)
- Sysklogd (1.4.1)
- Sysvinit (2.86)
- Tar (1.18)
- Tcl (8.4.15
- Texinfo (4.9)
- Udev (113)
- Udev Configuration Tarball
- Util-linux (2.12r)
- Vim (7.1)
- Vim (7.1) language files (optional)
- Zlib (1.2.3)

Les patchs utilisés sont :

- Bash Upstream Fixes Patch
- Bzip2 Documentation Patch
- Coreutils Internationalization Fixes Patch
- Coreutils Suppress Uptime, Kill, Su Patch
- Coreutils Uname Patch
- DB Fixes Patch
- Diffutils Internationalization Fixes Patch
- Expect Spawn Patch
- Gawk Segfault Patch
- GCC Specs Patch
- Grep RedHat Fixes Patch
- Groff Debian Patch
- GRUB Disk Geometry Patch
- Inetutils No-Server-Man-Pages Patch
- Kbd Backspace/Delete Fix Patch
- Kbd GCC-4.x Fix Patch
- Man-DB Fix Patch
- Mktemp Tempfile Patch
- Module-init-tools Patch
- Ncurses Coverity Patch
- Perl Libc Patch
- Readline Fixes Patch
- Shadow Useradd Patch
- Sysklogd 8-Bit Cleanness Patch
- Sysklogd Fixes Patch
- Texinfo Multibyte Fixes Patch

- Texinfo Tempfile Fix Patch
- Util-linux Cramfs Patch
- Util-linux Lseek Patch
- Vim Fixes Patch
- Vim Man Directories Patch

Dans la partie BLFS, nous avons installé les paquets suivants :

- CrackLib-2.8.9
- Cyrus SASL-2.1.21
- Heimdal-0.7.2
- Iptables-1.3.6
- GnuPG-1.4.3
- Linux-PAM-0.99.4.0
- Linux-PAM-0.99.4.0-docs.tar.bz2
- NSS-3.11.3
- OpenSSL-0.9.8d
- Shadow-4.0.15
- Stunnel-4.15
- Tripwire-2.4.0.1

Les correctifs utilisés pour la personnalisation de notre distribution sont :

- cyrus-sasl-2.1.21-openldap23-1.patch
- cyrus-sasl-2.1.21-openssl98-1.patch
- heimdal-0.7.2-setuid-patch.txt
- heimdal-0.7.2-fhs_compliance-1.patch
- heimdal-0.7.2-cracklib-1.patch
- openssl-0.9.8d-fix_manpages-1.patch
- tripwire-2.4.0.1-gcc4_build_fixes-1.patch

Pour installer le shorewall nous avons utilisé les paquets suivants :

- shorewall.common-4.0.0.tar.bz2 (common files).

- shorewall.shel-4.0.0.tar.bz2 (shel compiler)

- shorewall. perl-4.0.0.tar.bz2 (perl compiler).

Bibliographie

[1]Franck HUET, Linux : sécurité du système, sécurité des données pare-feu, chiffrement, authentification, eni, PARIS, 2004.

[2]Jean-François BOUCHAUDY&Abdelmadjid BERLAT, Linux administration, Eyrolles, Juillet 2002.

[3] Philippe LAVLEVEE and Christian SCHULLER, Mon système Linux, Ellipses, Février 2002.

[4] Daniel.P.BOVET, Le noyau Linux, Eclipses, 2003.

[5] Ronaldo GARCIA, Linux un système d'exploitation Unix, Eyrolles, 2000.

Webographie

www.wormux.org/wiki/howto/en/compile_source_code.php

www.**shorewall**.net

www.kernel.org/pub/**linux**/libs/**pam**/**linux**fr.org/

www.**linux**.com/base/ldp/howto/**Kerberos**-Infrastructure-HOWTO

www.picard.ups-tlse.fr/~dbonnafo/slides/**kerberos**-SSO

www.**stunnel**.org/**stunnel**.mirt.net/pipermail/**stunnel**-users

www.**tripwire**.org

www.**linux**journal.com/article/8758

www.ibiblio.org/pub/**Linux**/docs/HOWTO/translations/fr/html-1page/

www.**linux**fromscratch.org

www.kernel.org

www.generation-nt.com/**noyau-linuxgcc**.gnu.org

www.**makelinux**.net/**linux**.softpedia.com/get/Programming/Compilers/GNU-**Automake**

Glossaire

A

AES : Advanced Encryption Standard

B

BASH : Boume Again Shell

BLFS : Byond Linux From Scratch

C

CPU : Central Processor Unit

D

DMZ : Zone Démilitarisée

F

FS : Fichier Système

G

GRUB : GRand Unified Boot loader

GCC : GNU C Compiler

I

IPV4 : IP version 4

IPV6 : IP version 6

L

LILO : LInux LOader

LFS : Linux From Scratch

M

MBR : Master Boot Record

N

NAT : Network Address Translation

O

OSI : Open System Interconnection

P

PAM : Pluggable Authentication Modules

S

Stunnel : Secure tunnel

Shorewall : SOREline fireWALL