# Table des matières

[Table des matières 1](#_Toc60923488)

[Présentation 2](#_Toc60923489)

[Rôle du noyau 2](#_Toc60923490)

[Noyau standard d'installation 3](#_Toc60923491)

[Noyau monolithique 5](#_Toc60923492)

[Les modules 5](#_Toc60923493)

[Explication 5](#_Toc60923494)

[Les commandes 5](#_Toc60923495)

[Les fichiers 5](#_Toc60923496)

[La compilation 6](#_Toc60923497)

[Pourquoi compiler le noyau ? 6](#_Toc60923498)

[Pourquoi compiler un module ? 6](#_Toc60923499)

[Les outils nécessaires 6](#_Toc60923500)

[Introduction à make 6](#_Toc60923501)

[Exemple d’utilisation 7](#_Toc60923502)

[Variables 8](#_Toc60923503)

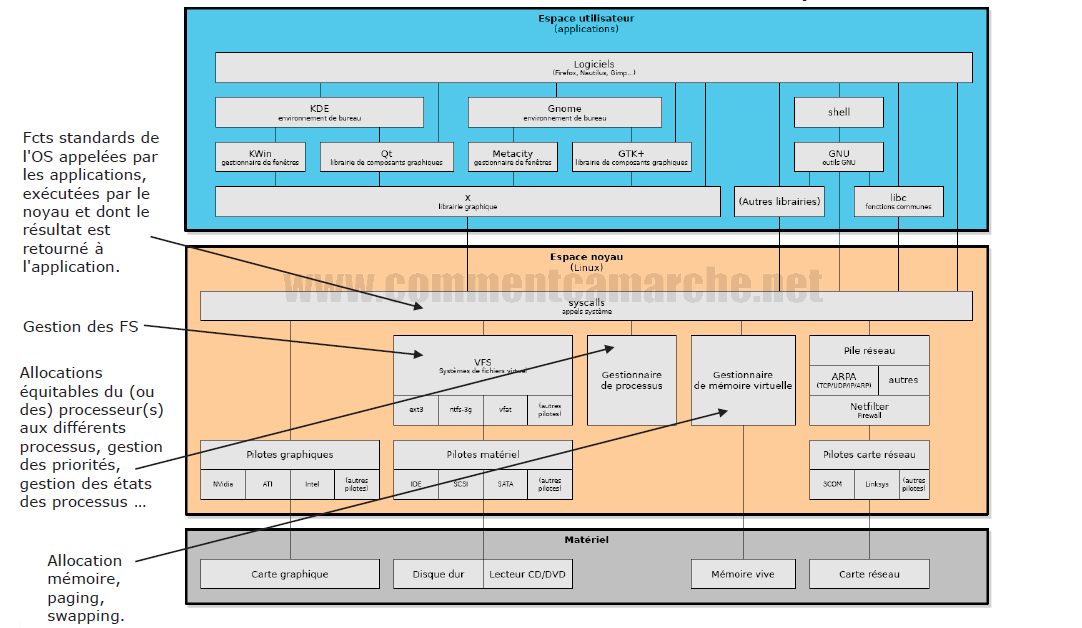
[Ecrire proprement un Makefile 8](#_Toc60923504)

[Avantages de make 9](#_Toc60923505)

[Les grandes étapes de la compilation 9](#_Toc60923506)

# Présentation

## Rôle du noyau



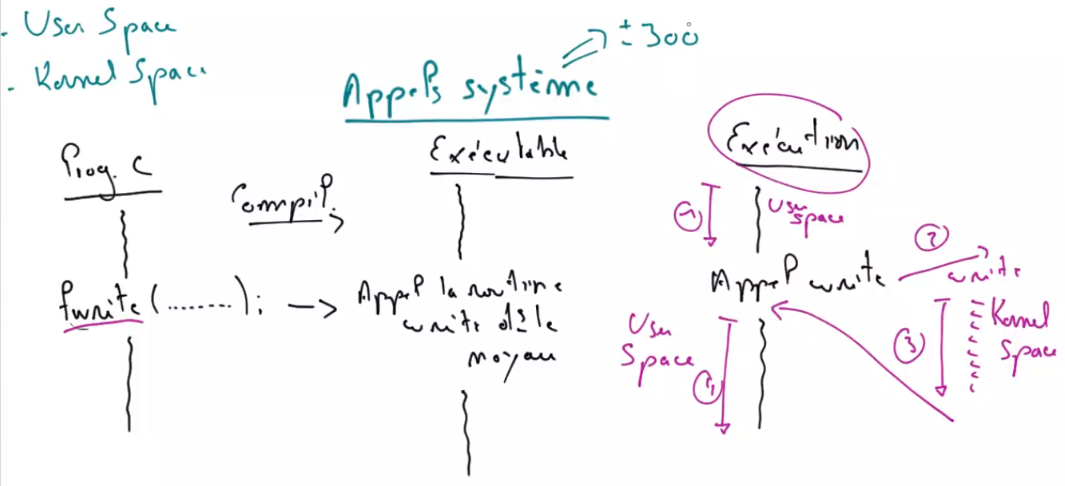
Un système informatique est fait en 3 composantes  
Les blocs sont des routines.   
Lorsqu'on a des routines qui s'exécute dans l'espace noyau, elles ont le droit de faire ce qu'elles veulent sur le matériel.   
Un logiciel qui tourne en espace utilisateur ne peut faire ce qu'il veut. Ex, un traitement de texte ne peut directement l'écrire sur le disque, il doit passer par l'espace noyau.

Espace utilisateur : routine non privilégiée

Lors du démarrage de la machine, le noyau et ses routines sont chargés en ram. Certaines routines doivent être en permanence en ram ex : gestionnaire de processus

Avec un ps ax on ne voit que les processus de l'user-space

syscall : appel système



Sous Windows ou linux, l'exécutable sera différent

## Noyau standard d'installation

bootload : grub2

Son but : installer en mémoire 2 choses

* noyau vmlinuz
* initramfs

Le fichier grub.cfg et grubenv ne doit pas être éditer à la main et vont être générer lors de la compilation de grub

La commande pour les générer : grub2-mkconfig

Il s'appelle vmlinuz-`uname –r`et est installé dans /bootpar l'outil d'installation de la distribution.

Il est bien souvent modulaire. La plupart des pilotes (dont ceux nécessaires au démarrage) sont souvent compilés dans des modules séparés dont les binaires d'extension.ko (ou .ko.xz à version compressée) sont enregistrés dans le dossier /lib/modules/`uname –r`

Il est souvent lancé par le chargeur de démarrage (Grub2).

# cat /boot/grub2/grub.cfg **(\*)**

…  
menuentry 'CentOS Linux (4.18.0-147.el8.x86\_64) 8 (Core)'…  
…

à titre affiché et sélectionnable dans le menu de Grub2

(\*) Après avoir mis à **'false'** la variable **GRUB\_ENABLE\_BLS\_CFG** du fichier /etc/default/grubet régénérer le fichier de configuration de Grub par la commande grub2\_mkconfig –o /boot/grub2/grub.cfg

# cat /boot/grub2/grub.cfg

…  
### BEGIN /etc/grub.d/10\_linux ###  
set root='hd0,msdos1'  
…

* hd0 : 1er disque du Bios rencontré par Grub2
* msdos : la table des partitions logée dans une MBR  
   Pour une table des partition gpt, on trouve à la place de msdos : gpt
* 1 : 1ère partition de cette table (nomenclature Grub2 rien n'est encore chargé ni monté !)

à ici cette partition correspond à /dev/sda1 et contient toute l'arborescence /boot.

à par la suite, tout chemin de nomenclature Grub2 commençant par / correspondra à la racine de cette partition et donc, dans ce cas, au chemin logique /boot.

grubenv

… root=/dev/mapper ro …  
ro = read only

# cat /boot/grub2/grub.cfg

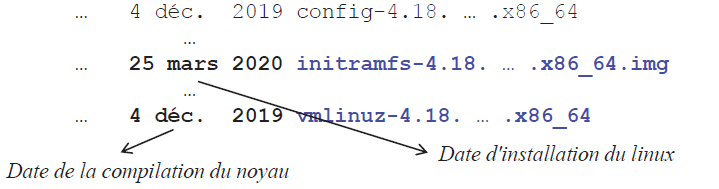
…  
linux /vmlinuz-4.18.0-147.el8.x86\_64 …  
…

Grub2 chargera le noyau 'vmlinuz-4.18… .x86\_64' se trouvant à la racine (/) de cette partition en lui passant les arguments *root=nom\_du\_fs\_princ. ro*…

initrd /initramfs-4.18.0-147.el8.x86\_64.img

Grub2 chargera et montera ce FS virtuel initial. Celui-ci contient les pilotes nécessaires afin que le FS principal puisse être monté par le noyau au bootage du système (contrôleur IDE ou SATA…, type du FS ext2 ou 3 ou 4 ou xfs …).  
Les pilotes stockés dans cette image dépendent de l'architecture du système.  
Elle sera donc générée lors de l'installation du système.

# ls –l /boot



|  |
| --- |
| cat /boot/loader/entries/…4.18.0-147-el8.x86\_64.conf  Celui-ci on peut le mettre à jour à la main |
| 1. initramfs mis en ram 2. vmlinuz mis en ram 3. monter en read only dans / |

## Noyau monolithique

Il est d'un seul bloc et ne contient donc aucun module externe (tous les pilotes sont dans le noyau).

* Pour des noyaux légers.
* Il est bien souvent utilisé dans les systèmes embarqués (faible besoin en pilotes).

Rem: Ce sont les pilotes du bootloader (ex. /boot/grub2/i386-pc/xfs.mod) qui permettront le chargement du noyau en RAM du système embarqué.

* Les pilotes (en nombre limité) sont compilés au sein du noyau.
* Pas de fichier initramfs.

## Les modules

### Explication

* La plupart des pilotes (et quelques sous-systèmes) peuvent être compilés sous forme de module (LKM: Loadable Kernel Module).
* L'ajout d'un périphérique se traduit donc par l'installation de son module binaire natif ou recompilé au départ de son code source.

à pas besoin de recompiler le noyau

à mise à jour aisée du noyau par le remplacement, la suppression ou l'ajout de modules

à chargement en mémoire uniquement des modules utilisés

* Un module ne pourra être chargé en mémoire que si toutes ses dépendances le sont également.
* Un module pourra être déchargé de la mémoire si celui-ci n'est pas une dépendance d'un ou de plusieurs autres modules.
* Le noyau ne peut charger un module que si ce dernier a été compilé pour la version exacte de ce noyau.

### Les commandes

lsmod – insmod – modprobe – depmod – rmmod – modinfo

### Les fichiers

/etc/modprobe.d/\*.conf à fichiers de configuration des modules

/lib/modules/$(uname –r) à arborescence de stockage des modules (.ko.xz)

/lib/modules/$(uname –r)/modules.dep à fichier des dépendances entre modules.

/etc/sysconfig/modules à dossier contenant des fichiers permettant de forcer le chargement de modules au démarrage.

# La compilation

## Pourquoi compiler le noyau ?

* Ajuster au mieux ses fonctionnalités au matériel et aux besoins (suppression / ajout de diverses fonctionnalités)
* Réglage de ses performances :
  + Modules dynamiques (externes au noyau) ou statiques (intégrés au sein du noyau)
* Essais des fonctionnalités d'un nouveau noyau.

Remarques :

* Pour une version commerciale de Linux, la recompilation pourrait être interdite sous peine de perdre le support.
* Certains fabricants ne livrent leur pilote que sous forme binaire et uniquement pour telle ou telle version d'un noyau

à ces pilotes deviendraient inutilisables sous un nouveau noyau recompilé.

## Pourquoi compiler un module ?

Pour installer un pilote à partir de son code source et ce, quelle que soit la version du noyau.

## Les outils nécessaires

Le compilateur C (souvent gcc).

- Les bibliothèques de développement C standard (glibc).

- Les bibliothèques spécifiques à la manipulation des menus de configuration du noyau : ncurses (pour menuconfig), qt3 (pour xconfig)

- L'utilitaire make

- Le package kmod qui contient des programmes permettant la gestion des modules et leur (dé)chargement automatique.

- mkinitrd (du package dracut) qui crée un fichier initramfs associé à une version du noyau.

- lsinitrd (du package dracut) qui montre le contenu d'un fichier initramfs.

## Introduction à make

* C'est un puissant langage spécialisé dans la gestion de projets.
* Il obéit aux règles définies dans un fichier appelé Makefile devant se trouver à la racine du dossier dans lequel make doit opérer.
* Structure d'un Makefile : ensemble de règles

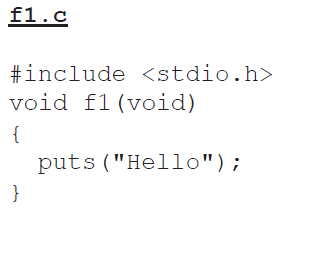
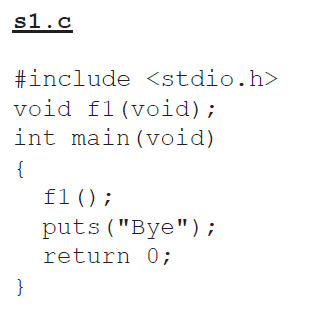
|  |  |
| --- | --- |
|  | Les dépendances sont analysées.  Si une dépendance est la cible d'une autre règle, cette règle est à son tour évaluée. |

Lorsque l'ensemble des dépendances est analysé et si la cible ne correspond pas à un fichier existant ou si un fichier dépendance est plus récent que la règle, les différentes commandes sont exécutées.

### Exemple d’utilisation

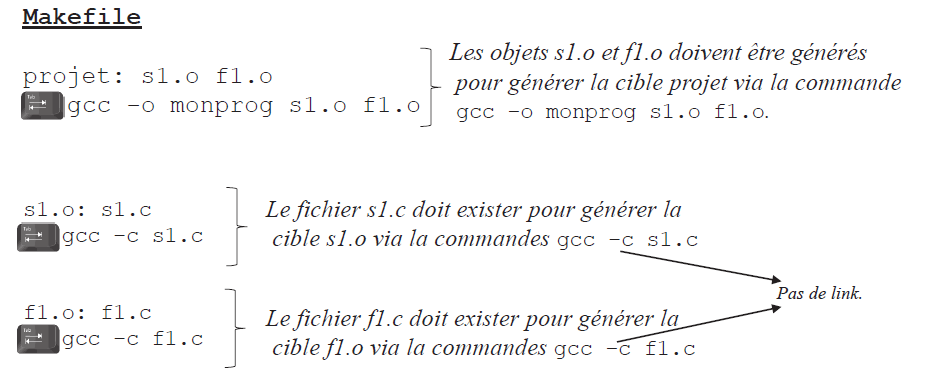
#### Composition du projet

Soit un projet C se composant de 2 fichiers sources :



#### Création du Makefile

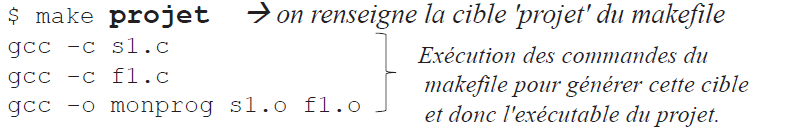
Création d'un Makefile qui permettra de générer un binaire monprog à partir de la compilation de ces 2 fichiers.



mcedit: (2 fois tab pour avoir 1 tabulation)

La commande make va aller rechercher la démarche dans le fichier Makefile  
projet : s1.o et f1.o doivent exister, s'il existe pas on va au 2 autres règles pour les générer

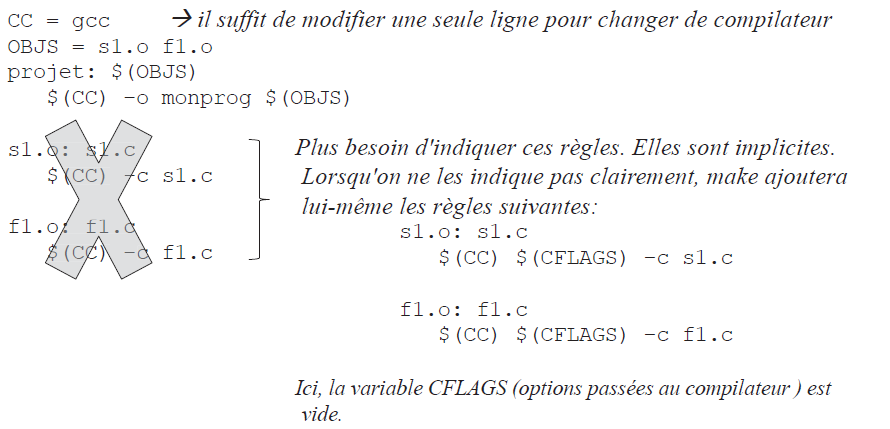
#### Appel à make et exécution



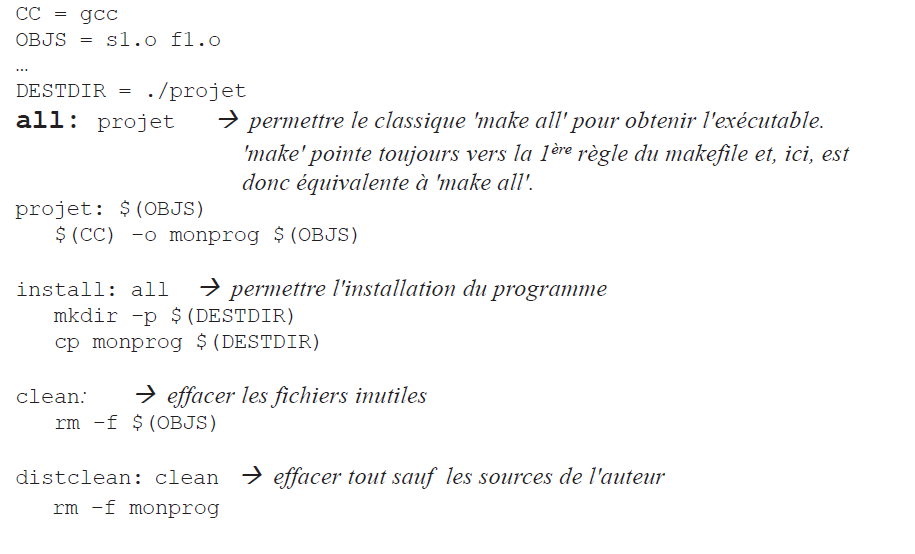
### Variables

make peut manipuler un grand nombre de variables

#### Réécriture du Makefile



### Ecrire proprement un Makefile



### Avantages de make

Lors de la recompilation complète du projet, make ne recompile que les sources dont la date est plus récente que leur fichier objet.

Exécutions (compilations) parallèles possibles sous des architectures multi-processeurs.

Exemple : make –j2 projet à2 compilations indépendantes seront lancées en même temps…

C'est un outil universel qui peut être utilisé dans le cadre d'une gestion de projet autre que la compilation multi-sources.

Son usage s'avère facile partout où il s'agit d'obtenir quelque chose à partir de fichiers existants

Exemple : encoder des CD en MP3, créer des images avec POV…

## Les grandes étapes de la compilation

1. Récupérer et installer les sources du noyau à partir du site officiel (kernel.org) ou à partir du site de votre distribution. (wget …)

2. Configurer le noyau. (make menuconfig)

C'est l'étape la plus importante et la plus longue.  
Concrètement, on va faire le choix des modules qui composent le noyau.  
On choisit aussi si l'inclusion est statique ou dynamique…

3. Compiler le noyau. (make)

4. Créer (éventuellement) un fichier initramfs, configurer le bootloader et installer le noyau dans /boot pour permettre le démarrage du noyau (make install)

5. Installer les modules dans /lib/modules/… (make modules\_install)

6. Redémarrer le système en utilisant le nouveau noyau.

7. Tests