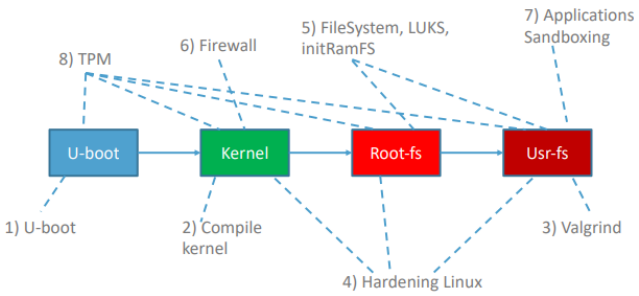
# 1 Introduction




## 1.1 Attaques

1. Attaques de surface
   (a) Utilisateurs des ports de debug
   (b) Connecteurs
   (c) Alimentations
2. Vecteurs d'attaque
   (a) Réseau (Ethernet, Wifi)
   (b) Application
   (c) Port série
   (d) USB, I2C, Flash, Bluetooth, GPS, etc...

## 1.2 Compilation pour nanopi

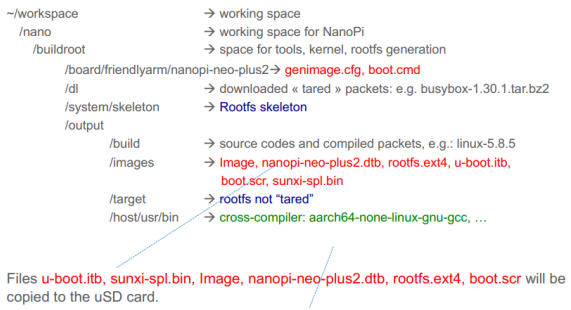
Cross-compilation (ARM) effectuée sur un système x86/x64. Buildroot est le toolchain utilisé. Les éléments suivants sont compilés :

1. Bootloader
2. Kernel
3. Rootfs

Puis les images sont copiées sur la carte SD

# 2 Buildroot

## 2.1 Répertoires



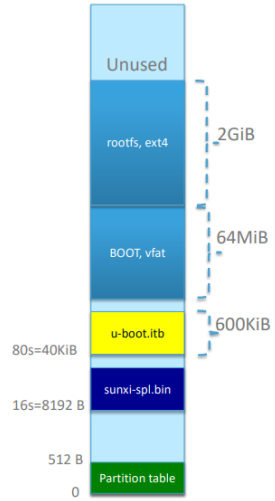

Ce qui est manquant dans le dossier output sera recompilé lorsque la commande `make` est lancée (ou alors en faisant la commande
`make <package>-rebuild`.
Le dossier `rootfs_overlay` permet d'ajouter des fichiers au `rootfs`
(`/workspace/nano/buildrootboard/friendlyarm/nanopi-neo-plus2/rootfs_overlay`)

## 2.2 Compilation

Dans le répertoire `buildroot`, effectuer la commande `make menuconfig` puis `make`. `make clean` pour effacer tous les fichiers compilés.
La configuration permet notamment de

1. Modifier le rootfs

## 2.3 Carte SD




Sector size = 512 Bytes

| Area Name | From (Sector #) | To (Sector #) | Size |
|---|---|---|---|
| rootfs, ext4 | | | 2GiB |
| BOOT, vfat | | | 64MiB |
| U-boot-itb | 80 | | 600KiB |
| sunxi-spl.bin | 16 | 79 | 32KiB |
| MBR (partition table) | 0 | 15 | 512B |

`genimage.cfg` ⟶ `genimage` ⟶ `sdcard.img` ⟶ `dd` ⟶ carte SD

Les fichiers pour l'initialisation sont

| | |
|---|---|
| `rootfs.ext` | Root file system |
| `Image` | Noyau Linux |
| `nanopo-neo-plus2.dtb` | Flattened device tree |
| `boot.scr` | Commandes boot compilées utilis |
| `boot.vfat` | Partition boot |
| `u-boot.itb` | Boot loader |
| `sunxi-spl.bin` | Secondary Program Loader |

`boot.vfat` contient `Image`, `nanopi-neo-plus2.dtb` et `boot.scr`. `boot.vfat` (ou `boot.ext4`) permet de créer `BOOT` sur la carte SD

#### 2.3.1 rootfs

Contient `/bin`, `/sbin`, `/root`, `/etc`, etc...

#### 2.3.2 boot.scr

Le fichier `boot.scr` est utilisé par u-boot pour charger le kernel Linux. Il est créé avec la commande `mkimage`

#### 2.3.3 boot.cmd

`boot.cmd` contient des informations de démarrage, notamment les emplacements des différents l'emplacement de `nanopi-neo-plus2.dtb`, du kernel et (si présent) de l'initramfs

### 2.4 Séquence de démarrage (6 phases)

1. Lorsque le $\mu$P est mis sous tension, le code stocké dans son BROM va charger dans ses 32KiB de SRAM interne le firmware `sunxi-spl` stocké dans le secteur no 16 de la carte SD / eMMC et l'exécuter.

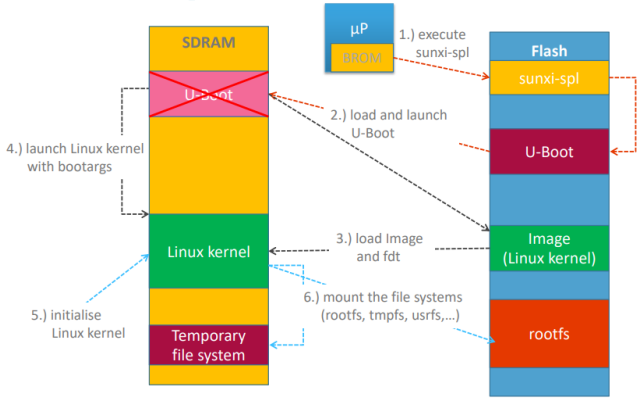
2. Le firmware `sunxi-spl` (Secondary Program Loader) initialise les couches basses du $\mu$P, puis charge l'U-Boot dans la RAM du $\mu$P avant de le lancer.

3. L'U-Boot va effectuer les initialisations hardware nécessaires (horloges, contrôleurs, ...) avant de charger l'image non compressées du noyau Linux dans la RAM, le fichier `Image`, ainsi que le fichier de configuration FDT (flattened device tree).

4. L'U-Boot lancera le noyau Linux en lui passant les arguments de boot (bootargs)

5. Le noyau Linux procédera à son initialisation sur la base des bootargs et des éléments de configuration contenus dans le fichier FDT (sun50i-h5-nanopi-neo plus2.dtb).

6. Le noyau Linux attachera les systèmes de fichiers (rootfs, tmpfs, usrfs, ...) et poursuivra son exécution.




# 3 U-boot

## 3.1 Compilation

On configure avec `make ubiit-menuconfig` puis on effectue la compilation avec une des deux manières :

1. `make uboot-rebuild`

2. supprimer les fichiers puis `make`

## 3.2 Démarrage

Si on appuie sur une touche, on entre en mode u-boot. La commande `booti` permet de lancer l'image linux (`boot` tout court va aussi lancer l'image Linux).
Avec les commandes présentes dans `boot.cmd`, on indique l'emplacement dans la ram de `Image` et `nanopi-neo-plus.dtb`

# 4 Kernel

## 4.1 Compilation

On configure avec `make linux-menuconfig` (ou `make linux-xconfig`) puis on lance une compilation avec `make linux-rebuild`

## 4.2 Busybox

Busybox est un éxécutable qui combine beaucoup de fonctions de base (ls, mv, rm, cat, etc...). En mettant toutes ces commandes dans un seul programme, on réduit énormément les redondances et par conséquent la taille de l'éxécutable.
On peut également configurer busybox avec `make busybox-menuconfig` puis le compiler avec `make busybox-rebuild`

# 5 Valgrind

## 5.1 Outils de Valgrind

- Memcheck : Détection d'erreur mémoire
- Cachegrind : Profiler de mémoire cache
- Callgrind : Profiler de cache avec infos supp et graph
- Helgrind : Détection d'erreur de threads 1
- DRD : Détection d'erreur de threads 2
- Massif : Profiler de heap et stack (memory leak)
- DHAT : Profiler de bloc dans le heap

## 5.2 Utilisation des outils

`Figures/Valgrind/outilsMem.png`

## 5.3 Trouver le bon outil

L'outil Memcheck regroupe beaucoup de fonctionnalité. C'est lui qu'il faut utilisé en priorité

# 6 Hardening

# 7 File system

## 7.1 Génération

Squelette de rootfs dans `workspace/nano/buildroot/system/ske`
Il est ensuite copié dans `buildroot/output/target` et les fichiers nécessaires y sont ensuite ajoutés.
Une fois que tous les fichiers sont ajoutés, une image `rootfs.xxx` est créé (xxx est ext4, squashfs, etc...)

## 7.2 1. De connaître les différents types de systèmes de fichiers ainsi que leurs applications

Pour les systèmes embarqués, il existe deux catégories de systèmes de fichiers :

1. Volatiles en RAM

2. Persitants sur des Flash (NOR et de plus en plus NAND)

Deux technologies principales sont disponible sur les Flash :

- MTD (Memory Technology Device)
- MMC/SD-Card (Multi-Media-Card / Secure Digital Card)

### 7.2.1 FS types

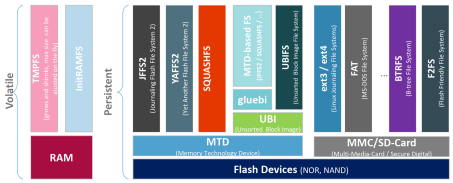



Figure 1: FS type

### 7.2.2 Choix d'un FS

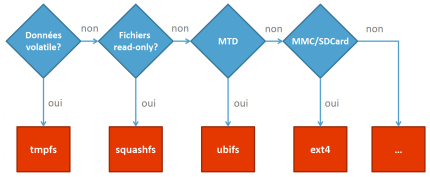



Figure 2: FS type

### 7.2.3 MMC technologies

MMC-eMMC-SD Card is composed by 3 elements

- MMC interface: handle communication with host
- FTL (Flash translation layer)
- Storage area: array of NAND chips

### 7.2.4 FTL

FTL is a small controller running a firmware. Its main purpose is to transform logical sector addressing into NAND addressing. It also handles:

- Bad block management
- Garbage collection.
- Wear levelling

## 7.3 2. De connaître les caractéristiques des filesystems ext2-3-4, ainsi que les commandes associées

"Filesystem considerations for embedded devices" is a good study about filesystems used on embedded systems.
This file system is very used in different Linux distribution.

- EXT filesystem was created in April 1992 and is a file system for the Linux kernel
  - Ext2 is not a journaled file system
  - Ext2 uses block mapping in order to reduce file fragmentation (it allocates several free blocks)
  - After an unexpected power failure or system crash (also called an unclean system shutdown), each mounted ext2 file system on the machine must be checked for consistency with the e2fsck program
- EXT2 replaced it in 1993
  - It was merged in the 2.4.15 kernel on November 2001
  - Ext3 is compatible with ext2
  - Ext3 is a journaled file system
  - The ext3 file system prevents loss of data integrity even when an unclean system shutdown occurs
- EXT4 arrived as a stable version in the Linux kernel in 2008
  - ext4 is backward compatible with ext3 and ext2, making it possible to mount ext3 and ext2 as ext4
  - Ext4 is included in the kernel 2.6.28
  - Ext4 supports Large file system:
    * Volume max: $2^60$ bytes
    * File max: $2^40$ bytes
  - Ext4 uses extents (as opposed to the traditional block mapping scheme used by ext2 and ext3), which improves performance when using large files and reduces metadata overhead for large files

### 7.3.1 Ext4 commands

```
# Create a partition (rootfs), start 64MB,
 length 256MB
sudo parted /dev/sdb mkpart primary ext4
131072s 655359s
# Format the partition with the volume
label = rootfs
sudo mkfs.ext4 /dev/sdb1 -L rootfs
# Modify (on the fly) the ext4
configuration
sudo tune2fs <options> /dev/sdb1
# check the ext4 configuration
mount
sudo tune2fs -l /dev/sdb1
sudo dumpe2fs /dev/sdb1
# mount an ext4 file system
mount  t  ext4 /dev/sdb1 /mnt/test //
with default options
```

```
mount t ext4 o defaults,noatime,
discard,nodiratime,data=writeback,acl,
user_xattr
/dev/sdb1 /mnt/test
```

#### 7.3.2 Ext4 mount options and MMC/SD-Card

- filesystem options can be activated with the mount command (or to the /etc/fstab file)
- These options can be modified with tune2fs command
- Journaling: the journaling guarantees the data consistency, but it reduces the file system performances
- MMC/SD-Card constraints: In order to improve the longevity of MMC/SDCard, it is necessary to reduce the unnecessary writes
- Mount options to reduce the unnecessary writes (man mount) :
  - noatime: Do not update inode access times on this filesystem (e.g., for faster access on the news spool to speed up news servers)
  - nodiratime: Do not update directory inode access times on this filesystem
  - relatime: this option can replace the noatime and nodiratime if an application needs the access time information (like mutt)

Mount options for the journaling (man ext4):

- Data=journal: All data is committed into the journal prior to being written into the main filesystem (It is the safest option in terms of data integrity and reliability, though maybe not so much for performance
- Data=ordered: This is the default mode. All data is forced directly out to the main file system before the metadata being committed to the journal
- Data=writeback: Data ordering is not preserved - data may be written into the main filesystem after its metadata has been committed to the journal. It guarantees internal filesystem integrity, however it can allow old data to appear in files after a crash and journal recovery.
- Discard: Use discard requests to inform the storage that a given range of blocks is no longer in use. A MMC/SD-Card can use this information to free up space internally, using the free blocks for wear-levelling.
- acl: Support POSIX Access Control Lists
- $\text{user}_x attr$ : $Support" user." extended attributes default$ : $rw, suid, dev, exec, auto, nouser, and async$
- - rw : read-write
  - suid : Allow set-user-identifier or set-group-identifier bits
  - dev : Interpret character or block special devices on the filesystem
  - exec : Permit execution of binaries
  - auto : Can be mounted with the -a option (mount –a)
  - nouser : Forbid an ordinary (i.e., non-root) user to mount the filesystem
  - async : All I/O to the filesystem should be done asynchronously

#### 7.3.3 /etc/fstab file

File **/etc/fstab** contains descriptive information about the filesystems the system can mount

- ¡file system¿ : block special device or remote filesystem to be mounted
- ¡mount pt¿ : mount point for the filesystem
- ¡type¿ : the filesystem type
- ¡options¿ : mount options associated with the filesystem
- ¡dump¿ : used by the dump (backup filesystem) command to determine whichfilesystems need to be dumped (0 -¿ no backup)
- ¡pass¿ : used by the fsck (8) program to determine the order in which filesystem checks are done at reboot time. The root filesystem should be specified with 1, and other filesystems should have a 2. if ¡pass¿ is not present or equal 0 -¿ fsck willassume that the filesystem is not checked.
- Field options: It contains at least the type of mount plus any additional options appropriate to the filesystem type.

Common for all types of file system are the options (man mount) :

- auto : Can be mounted with the -a option (mount -a)
- defaults : Use default options: rw, suid, dev, exec, auto, nouser, and async
- nosuid : Do not allow set-user-identifier or set-group-identifier bits to take effect
- noexec : Do not allow direct execution of any binaries on the mounted file system
- nodev : Do not interpret character or block special devices on the file system

### 7.4 3. D'expliquer les différents "files systems" utilisés dans les systèmes embarqués (ext2-3-4, BTRFS, F2FS, NILFS2, XFS, ZFS, . . . )

#### 7.4.1 BTRFS (B-Tree filesystem)

- BTRFS is a "new" file system compared to EXT. It is originally created by Oracle in 2007, it is a B-Tree filesystem
- It is considered stable since 2014
- Since 2015 BTRFS is the default rootfs for openSUSE
- BTRFS inspires from both Reiserfs and ZFS
- Theodore Ts'o (ext3-ext4 main developer) said that BTRFS has a better direction than ext4 because "it offers improvements in scalability, reliability, and ease of management"

#### 7.4.2 F2FS (Flash-Friendly File System)

It is a log filesystem. It can be tuned using many parameters to allow best handling on different supports. F2FS features :

- Atomic operations
- Defragmentation
- TRIM support (reporting free blocks for reuse)

### 7.4.3 NILFS2 (New Implementation of a Log-structured File System)

- Developed by Nippon Telegraph and Telephone Corporation
- NILFS2 Merged in Linux kernel version 2.6.30
- NILFS2 is a log filesystem
- CoW for checkpoints and snapshots
- Userspace garbage collector

### 7.4.4 XFS (Flash-Friendly File System)

XFS was developed by SGI in 1993.

- Added to Linux kernel in 2001
- On disk format updated in Linux version 3.10
- XFS is a journaling filesystem
- Supports huge filesystems
- Designed for scalability
- Does not seem to be handling power loss (standby state) well

### 7.4.5 ZFS (Zettabyte ($10^21$)File System)

ZFS is a combined file system and logical volume manager designed by Sun Microsystems.

- ZFS is a B-Tree file system
- Provides strong data integrity
- Supports huge filesystems
- Not intended for embedded systems (requires RAM)
- License not compatible with Linux

### 7.4.6 Conclusion

Performances:

- EXT4 is currently the best solution for embedded systems using MMC
- F2FS and NILFS2 show impressive write performances

Features:

- BTRFS is a next generation filesystem
- NILFS2 provides simpler but similar features

Scalability:

- EXT4 clearly doesn't scale as well as BTRFS and F2FS

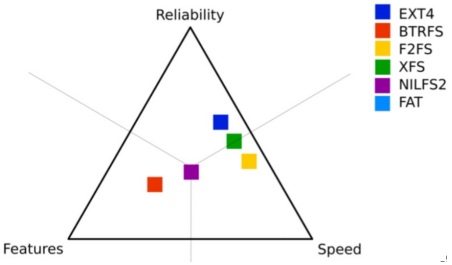



Figure 3: FS Comparaison

## 7.5 4. Expliquer les files system de type Journal, B-Tree/CoW, log filesystem

### 7.5.1 Journalized filesystem

A journalized filesystem keeps track of every modification in a journal in a dedicated area

- EXT3, EXT4, XFS, Reiser4
- Journal allows to restore a corrupted filesystem
- Modifications are first recorded in the journal
- Modifications are applied on the disk
- If a corruption occurs: The File System will either keep or drop the modifications
  - Journal is consistent : we replay the journal at mount time
  - Journal is not consistent : we drop the modifications

### 7.5.2 B-Tree filesystem

- ZFS, BTRFS, NILFS2
- B+ tree is a data structure that generalized binary trees
- CoW (Copy on Write) is used to ensure no corruption occurs at runtime :
  - The original storage is never modified. When a write request is made, data is written to a new storage area
  - Original storage is preserved until modifications are committed
  - If an interruption occurs during writing the new storage area, original storage can be used

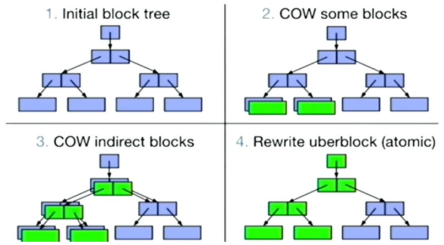



Figure 4: B-tree type FS execution

### 7.5.3 Log filesystem

Log-structured filesystems use the storage medium as circular buffer and new blocks are always written to the end.

- F2FS, NILFS2, JFFS2, UBIFS
- Log-structured filesystems are often used for flash media since they will naturally perform wear-levelling
- The log-structured approach is a specific form of copy-on-write behavior

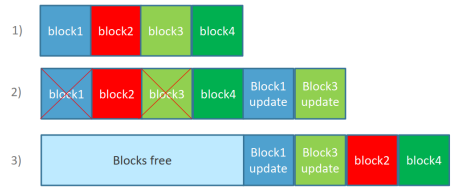



Figure 5: $\log_f stype FS execution$

1. Initial state
2. Block 1-3 are updated, old blocks 1-3 are not used
3. Garbage copies block2 and 4, and frees old block1-2-3-4

## 7.6 5. De connaître les caractéristiques du filesystem Squashfs, ainsi que les commandes associées

beginitemize

Squashfs is a compressed read-only filesystem for Linux

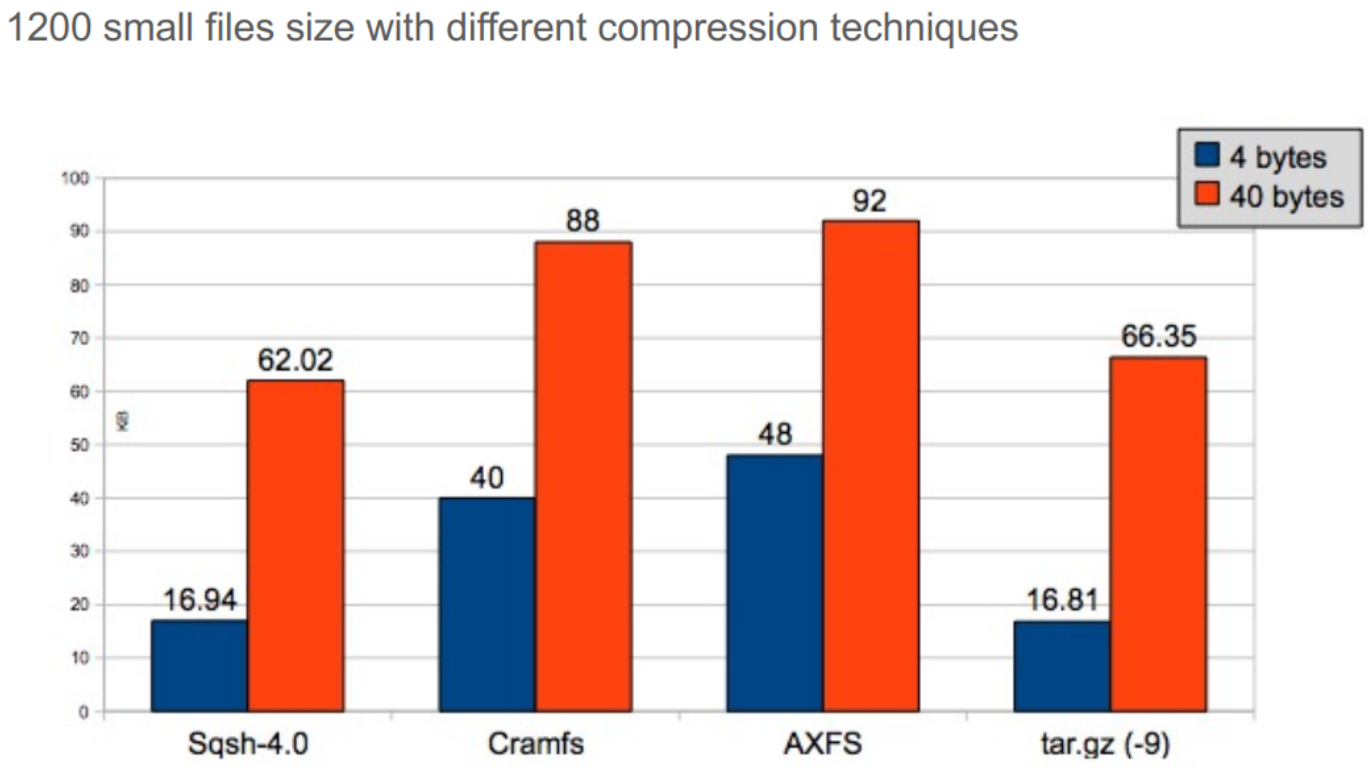
Squashfs versions

- Squashfs 2.0 and squashfs 2.1: 2004, kernel 2.2
- Squashfs 3.0: 2006, kernel 2.6.12
- Squashfs 4.2: 2011, kernel 2.6.29

It uses gzip, lzma, lzo, lz4 and xz compression to compress files, inodes and directories

SquashFS 4.0 supports 64 bit filesystems and files (larger than 4GB), full uid/gid information, hard links and timestamps

Squashfs is intended for general read-only filesystem use, for archival use, and in embedded systems with small processors where low overhead is needed




### 7.6.1 Create and use squashed file systems