Buildroot

1. D'expliquer les principaux répertoires de buildroot

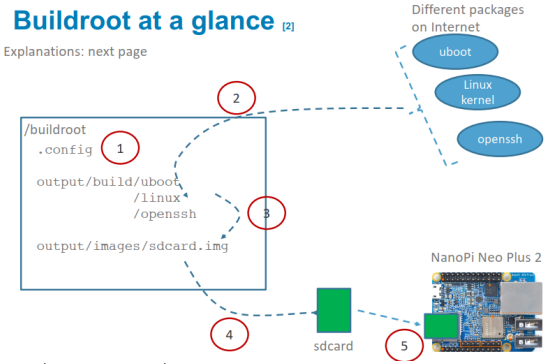
```
├── board
│   └── friendlyarm
│       └── nanopi-neo-plus2
│           ├── boot.cmd                // used by uboot
│           ├── genimage.cfg            // sdcard configuration
│           ├── nanopi-neo-plus2.dts    // Flatted device tree
│           ├── rootfs_overlay          // Add files to rootfs
│           │   └── etc
│           │       ├── network
│           │       │   └── interfaces
│           │       └── ssh
│           │           └── sshd_config
│           ├── ses_linux_defconfig     // Linux kernel configuration
│           └── uboot-extras.config     // uboot fragments configuration
└── configs
    └── ses_defconfig                   // SeS Buildroot configuration
```

/buildroot
- /board/friendlyarm/nanopi-neo-plus2 → own configuration
- /dl → downloaded « tared » packets: e.g. Busybox-1.30.1.tar.bz2
- /system/skeleton → Rootfs skeleton
- /output
  - /build → source codes and compiled packets, e.g.: linux-5.8.5
  - /images → sdcard.img, Image, nanopi-neo-plus2.dtb, u-boot.itb, boot.scr, sunxi-spl.bin
  - /target → rootfs not "tared"
  - /host/usr/bin → cross-compiler: aarch64-linux-gnu-gcc, ...

Ce qui est manquant dans le dossier output sera recompil´e lorsque la commande make est lancée (ou alors en faisant la commande make <package>-rebuild

2. D'expliquer le principe de fonctionnement de buildroot

Générateur de Linux embarqué avec le système de cross-compilation



1) The /buildroot/.config file contains the NanoPi Neo Plus 2 buildroot configuration
2) During the make, the configured packages sources files are downloaded to the directories output/uboot, ...
3) At the end of the make, the sdcard.img file is created. This file containts the bootloader, kernel, rootfs, ...
4) The sdcard is flashed with the sdcard.img file
5) The sdcard is put in the NanoPi Neo Plus 2

3. D'expliquer la configuration de buildroot pour un hardware donné

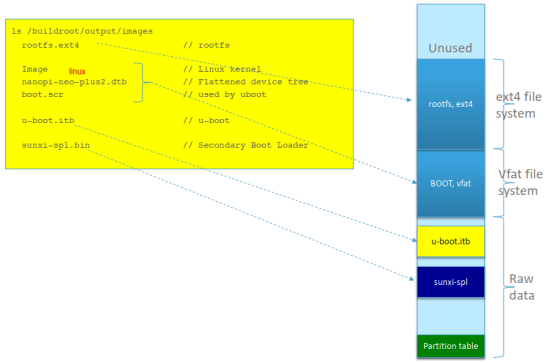
The Buildroot configuration is contained in two files : .config and xxx_defconfig
- Buildroot uses Kconfig like the Linux kernel
- The .config file is a full default config file with more 4000 lines
- The defconfig stores only the values for options for which the non-default value is chosen. It is a small file

4. D'expliquer comment faire un patch et appliquer ce patch dans buildroot

5. D'expliquer comment configurer, compiler buildroot, u-boot, kernel

make menuconfig to config buildroot
make linux-menuconfig to config linux kernel
make uboot-menuconfig to config uboot
La commande make permet de compiler u-boot et buildroot
make linux-rebuild compile le linux

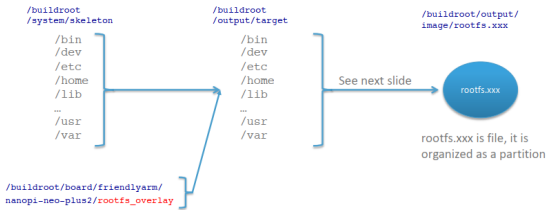
6. D'expliquer comment la SD-Card est générée



Pour créer sdcard.img, buildroot utilise le script genimage.sh → fichier : /buildroot/board/friendlyarm/scripts/genimage file

7. D'expliquer comment le rootfs est généré

- A rootfs skeleton is in the directory /buildroot/system/skeleton
- This skeleton is copied to the pseudo rootfs directory /buildroot/output/target
- It is possible to add files, directories with rootfs_overlay
- After the make command, the pseudo rootfs is populated and copied to one file in this directory : /buildroot/output/image/rootfs.xxx (xxx can be ext4, squashfs, ...)



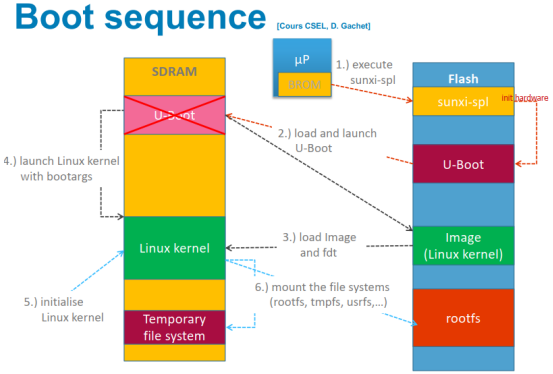
8. D'expliquer le rootfs_overlay

Le dossier rootfs_overlay permet d'ajouter des fichiers au rootfs

9. Savoir installer un nouveau package dans buildroot

U-boot

10.D'expliquer le démarrage du NanoPi



Le démarrage du NanoPi NEO Plus2 se décompose en 6 phases : - Lorsque le μP 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.
- Le firmware « sunxi-spl » (Secondary Program Loader) initialise les couches basses du μP, puis charge l'U-Boot dans la RAM du μP avant de le lancer.
- 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).
- L'U-Boot lancera le noyau Linux en lui passant les arguments de boot (bootargs).
- 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-neoplus2.dtb).
- Le noyau Linux attachera les systèmes de fichiers (rootfs, tmpfs, usrfs, ...) et poursuivra son exécution.

11. De connaître, expliquer les principales commandes de u-boot utilisées durant le démarrage

Si on appuie sur une touche pendant le démarrage, on entre dans le mode u-boot
boot load the Linux kernel, Image file, the FDT (Flattened Device Tree) and start Linux *booti* permet de lancer d'image linux.
mmc mmc(MultiMediaCard) sub system
printenv print environement variables

```
ext2load- load binary file from a Ext2 filesystem
ext2ls  - list files in a directory (default /)
ext4load- load binary file from a Ext4 filesystem
ext4ls  - list files in a directory (default /)
ext4size- determine a file's size

fatinfo - print information about filesystem
fatload - load binary file from a dos filesystem
fatls   - list files in a directory (default /)
fatmkdir- create a directory
fatrm   - delete a file
fatsize - determine a file's size
fatwrite- write file into a dos filesystem
```

12. De savoir comment configurer u-boot

On configure avec make uboot-menuconfig puis on effectue la compilation avec une des deux mani'eres : 1. make uboot-rebuild 2. supprimer les fichiers puis make La configuration de u-boot est stock´ee dans .config

13. D'expliquer comment améliorer la sécurité de u-boot

14. De connaître les différentes étapes pour la création de l'image de u-boot.itb

15. Savoir ce que fait la commande strip sur un fichier elf

16. De connaître les différentes étapes pour la création de uImage

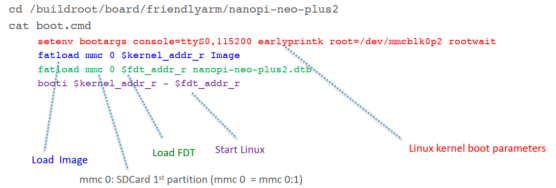
17. De connaître l'utilité du Flattened Device Tree

The Flattened Device-Tree (FDT) was introduced in kernel 2.6. It is a file which contains the hardware description. Linux uses it for its configuration
FDT has two files :
§ .dts : Device Tree Source, it is an ascii file
§ .dtb : Device Tree Blob, it is a binary file
After the introduction of FTD with the kernel 2.6, a new binary file format was created : FIT (Flattened Image Tree). This format allows to insert different files into a single file

18. De connaître de manière générale le mapping de la SDCard

19. D'expliquer le fichier boot.scr

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



Compilation du noyau

20. De connaître les principaux répertoires du noyau Linux

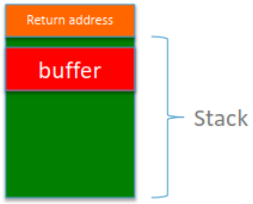
This directory has these main sub-directories :
arch Hardware dependent code
block Generic functions for the block devices
crypto Cryptographic algo. used in the kernel
Documentation Documentation about the kernel
drivers All drivers known by the kernel
fs All filesystem know by the kernel
include kernel include files

init Init code (function start_kernel) *ipc* Interprocess communication
kernel Kernel code, scheduler, mutex, ...
lib different libraries used by the kernel
mm Memory management
net Different protocols, IPv4, IPv6, bluetooth, ...
samples Different examples, kobject, kfifo, ...
security Encrypted keys, SELinux, ...
sound Sound support for Linux kernel
virt Kernel-based virtual machine
This directory has these main files
vmlinux Linux kernel, ELF format, ARM aarch64
.config Linux kernel configuration
.config.old Old Linux kernel configuration
Kconfig Configuration for the make linux-xconfig
Makefile makefile

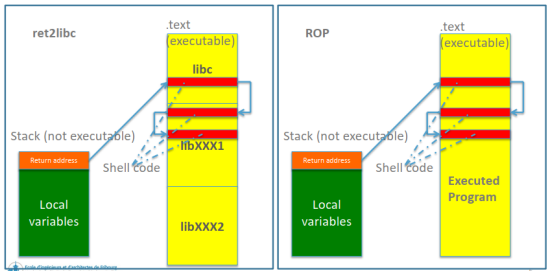
21. De connaître les principales méthodes pour sécuriser le noyau Linux
Enable -fstack-protector buffer overflow detection

22. D'expliquer le principe des software attacks : buffer overflow, ret2libc, ROP
A buffer overflow attack can insert and executes a shell code in an executable stack.

```
void main () {
char buffer [4];
    strcpy (&buffer[0], "123456");
}
```



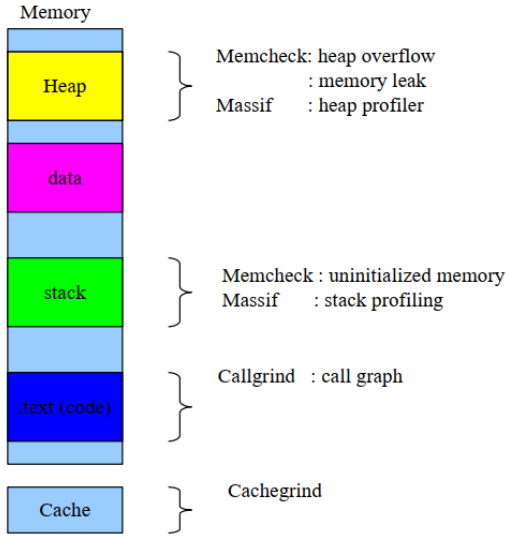
Now, the stack memory is no longer executable
§ However, a technique called ret2libc could be used to bypass non executable stack memory.
The main idea is to execute code in an executable memory like libc() or other libraries.
§ Another technique called ROP, or Return-Oriented Programming allows also to bypass non executable stack memory. The main idea is to execute code in the program itself



23. D'expliquer le principe des protections contre les softwares attacks : ASLR, PIE, canary
ASLR (Address Space Layout Randomization) randomize_va_space randomizes the stack and heap addresses
The PIE (Position Independent Executable) avoids the ret2lib and ROP problems (because code addresses change)

Valgrind

24. De connaître les différents outils de Valgrind et leur utilisation
1. Memcheck is a memory error detector. It helps you make your programs, particularly those written in C and C++, more correct.
2. Cachegrind is a cache and branch-prediction profiler. It helps you make your programs run faster.
3. Callgrind is a call-graph generating cache profiler. It has some overlap with Cachegrind, but also gathers some information that Cachegrind does not.
4. Helgrind is a thread error detector. It helps you make your multi-threaded programs more correct.
5. Massif is a heap and stack profiler. It helps you make your programs use less memory.



25. Pour un code donné avec des erreurs, savoir quel-s outil-s de Valgrind utiliser

Hardening Linux

26. De contrôler l'intégrité d'un package, d'un programme

27. De configurer un nouveau package, programme

28. De cross-compiler un programme

29. De contrôler les services, les ports ouverts

30. De contrôler les « file systems »

31. De contrôler les permissions des fichiers, répertoires
By separating file systems into various partitions, it is possible to fine tune permissions and functionalities.
Ajournaling file system must be installed (e.g. ext3, ext4) and activated
Areas where users have "write privileges" should be kept on their own partition.
LVM (Logical Volume Manager) can be used when more than four partitions are required.
§ read access (r)
§ write access (w)
§ execute access (x)

32. De sécuriser le réseau

33. De contrôler-sécuriser les comptes utilisateurs

34. De limiter le login root

35. De sécuriser le noyau

36. De sécuriser une application

37. De contrôler le démarrage de Linux

38. D'appliquer la méthodologie OSSTMM simplifiée

Filesystem

39. De connaître les différents types de systèmes de fichiers ainsi que leurs applications

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

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

42. Expliquer les files system de type Journal, B_Tree/CoW, log filesystem

43. De connaître les caractéristiques du filesystem Squashfs, ainsi que les commandes associées

44. De connaître les caractéristiques du filesystem tmpfs, ainsi que les commandes associées

45. De connaître les caractéristiques du filesystem LUKS, ainsi que les commandes associées

46. Savoir expliquer la gestion des clés de LUKS

47. De connaître les caractéristiques du filesystem InitramFS, ainsi que les commandes associées

48. De savoir créer un initramFS

Filesystem security

49. De connaître les « files permissions » sous Linux

50. De contrôler et sécuriser les comptes utilisateurs sous Linux

51. De connaître les real-effective userID and groupID

52. De connaître les ACL

53. De connaître les attributs particuliers des filesystems ext2-3-4

54. De rechercher des permissions de fichier faibles

55. Comment sécuriser les répertoires temporaires

56. De savoir comment les mots de passe sont mémorisés sous Linux

57. De connaitre les différentes possibilités pour casser un mot de passe

58. De savoir utiliser hashcat pour casser un mot de passe

Firewall iptables

59. De connaître les principes de Netfilter, iptables

60. Savoir expliquer les notions de chain-tables

61. Savoir expliquer les différences entre les firewall Stateless et Stateful

62. Savoir configurer avec iptables un firewall simple de types Stateless (pages 17-19) et Stateful (pages 26-32)

63. Connaître le principe des NFQUEUE

TPM

64. Savoir expliquer uniquement le principe des chiffrements symétrique, asymétrique,fonctions de hachage, la signature digitale

65. Connaitre les différentes implémentations des TPM (discrete, integrated, Hypervisor,Software)

66. Connaitre l'architecture interne d'un TPM

67. Connaitre les différentes hiérarchies des TPM (endorsement, platform, owner, null)

68. Savoir créer, utiliser des clés avec un TPM

69. Connaitre les commandes principales d'un TPM (pas tous les paramètres, mais savoir expliquer ce que font ces commandes, être capable de dessiner ce que font les commandes)

70. Savoir encrypter-décrypter, signer-vérifier avec un TPM

71. Savoir utiliser les registres PCR

72. Savoir sauver des données sur le TPM

73. Savoir sauver des données et les protéger avec une PCR policy