Linux embarqué sur Raspberry PI

Gilles Maire

2017

Plan de la formation

1. Présentation
2. Licences
3. Compilation
4. Le noyau
5. Busybox
6. Raspberry
7. Compilation croisée
8. Buildroot
9. Divers
10. gdb
11. Cas avancés

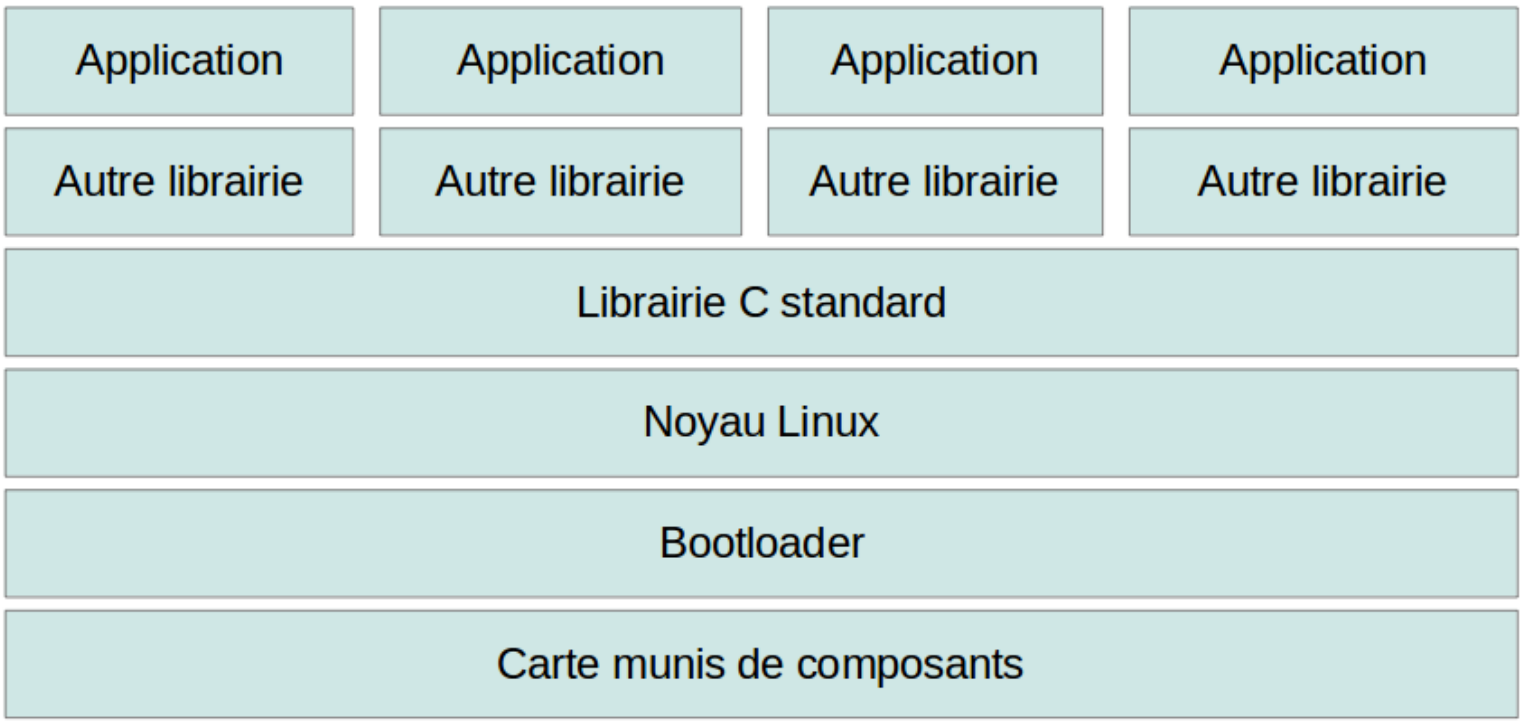
Présentation

Rubriques

- Architecture
- Init et démons
- Init et shell

Architecture

Architecture Linux



Les différents éléments d'un système Linux

- Le Bios / le Boot
- Le noyau
- La librairie libc
- L'init
- Le shell
- Le gestionnaire de fenêtres
- Les distributions
- Les outils de mise à jour

Le BIOS et le boot

- Tous les PC commencent par exécuter un programme appelé BIOS présent dans une EEPROM ou CMOS c'est à dire en mémoire ROM.
- Ce Bios démarre puis appelle un bootloader sur un des disques. Ce disque sera appelé le BOOT DISK
- Ce Bios se configure en tapant sur F1 ou ESC au démarrage (notamment pour indiquer le disque sur le quel le bootloader sera installé)
- Il existe plusieurs distributions de de BIOS : AMI, AWARD, MSI ou les BIOS des constructeurs
- Cette partie est très spécifique au monde PC, en embarqué nous n'utiliserons pas le même mécanisme.

Le boot

- Le BIOS appelle un média : disque, clé USB, CDROM, Disquette
- Au secteur 0 de ce média se trouve le bootloader qui est en charge de l'amorce de l'OS c'est à dire du chargement du noyau Linux
- Chacun de ces médias est organisé suivant un système de fichiers
- doit savoir ouvrir le système de fichier et les médias pour charger le noyau
- Cette partie est très spécifique au monde PC, en embarqué nous n'utiliserons pas le même mécanisme.

Introduction au Noyau

- Le noyau ou kernel est responsable du fonctionnement des couches basses de Linux.
- C'est ce noyau qui a été développé en 1991 par Linus Torvalds qui en est toujours le mainteneur en 2017
- Des milliers de personnes contribuent via un serveur git à l'amélioration de chaque release et à l'ajout des drivers des nouvelles cartes
- Le noyau est en licence GNU GPL V2 (sources disponibles avec droit de modification pour toute vente d'un système contenant Linux)
- Une fois le noyau lancé, le programme init est appelé
- En embarqué, le noyau utilisé est le même que sur les postes PC avec une compilation pour un processeur adapté.

La librairie LibC

- Très proche du noyau cette librairie permet aux programmes systèmes fonctionnant sous Linux d'appeler des primitives du noyau
- Elle permet les **appels systèmes** suivants :
  - malloc, free, memset, strcpy, strlchr, strlen
  - gestion des Threads & gestion des Mutex
  - gestion de l'encodage décodage UTF8
  - stdio : getc, printf etc
  - regex
  - qsort
  - calculs mathématiques flottants
  - gestion des timezones
- Il existe plusieurs souches de la librairie : glibc, uclibc, eglibc, dietlibc, newlibc
- La libc est la librairie contenant tous les appels système
- La commande `time` précédent une commande donne le temps passé par une commande à exécuter des appels systèmes, autrement dit des primitives lancées par la libc.

Init et démons

Inittab et init

- Une fois le noyau chargé en mémoire, le programme `/sbin/init` est appelé (`ps awx` donne le numéro de processus 1 pour l'init)
- Le fichier de configuration de l'init est /etc/inittab remplacé par /etc/init/rc-sysinit.conf sous Ubuntu. Ce fichier donne le niveau d'utilisation de l'équipement
- `/etc/init.d` contient les programmes de démarrage de chacun des démons
- Chaque niveau d'utilisation est matérialisé par un répertoire `/etc/rc0.d` `/etc/rc1.d ... /etc/rc6.d` chacun de ces répertoires comprenant un lien symbolique vers le programme de démarrage contenu dans `/etc/init.d`
- Les choix de ces différents programmes dépend de l'utilisation que l'on souhaite faire de son équipement

Niveau d'exécution

- Dans le fichier `/etc/inittab` on trouve des lignes sous la syntaxe:
  - `id : niveau(x)_exécution` : action : processus
  - une étiquette indiquant le niveau par défaut
- Les niveaux sont les suivants :
  - **Niveau 0 :** arrêt du système,
  - **Niveau 1 :** mode de maintenance (single user)
  - **Niveau 2 :** souvent non utilisé
  - **Niveau 3 :** système réseau sans interface graphique
  - **Niveau 4 :** souvent non utilisé
  - **Niveau 5 :** avec interface graphique
  - **Niveau 6 :** redémarrage du système
- Pour passer à un niveau n, il faut être root et entrer la commande

```
init n
```

Exemple fichier /etc/inittab

```
# Le niveau d'exécution par défaut
id:2:initdefault:
# Initialisation du système
si::sysinit:/etc/rc.d/rc.sysinit
# Les différents niveaux d'exécution
l0:0:wait:/etc/rc.d/rc 0
l1:1:wait:/etc/rc.d/rc 1
l2:2:wait:/etc/rc.d/rc 2
l3:3:wait:/etc/rc.d/rc 3
l4:4:wait:/etc/rc.d/rc 4
l5:5:wait:/etc/rc.d/rc 5
l6:6:wait:/etc/rc.d/rc 6
# Intercepter les touches CTRL-ALT-DELETE
ca::ctrlaltdel:/sbin/shutdown -t3 -r now
1:2345:respawn:/sbin/mingetty tty1
```

Les différents types de démon

- Les classiques
  - Gestion des fichiers
  - Exécution des tâches
  - Protocole réseau
  - Gestion de l'heure et des logs
  - Système
- Les desktop
  - Gestion l'énergie
  - Gestion du multimédia
  - Gestion de la communication
  - Gestion de l'affichage

Init et shell

Les commandes de base et programme

- Une fois l'init appelée elle peut à son tour appeler un simple shell script ou un serveur Web
- On peut décider d'adjoindre d'autres programmes personnalisés
- En général en embarqué on utilise une source appelée Busybox qui permet de compiler l'ensemble des outils que l'on souhaite fournir à son équipement
- On fournit un shell permettant des commandes de maintenance

Le Shell

- Le shell est un programme qui interprète des commandes lancées par un opérateur
- Ces commandes peuvent être également empilées dans un fichier qui les lancera sous contrôle de conditions (if while etc) appelé ShellScript
- Il existe plusieurs shell : sh (bourne shell), bash, ksh (korn shell), csh, tcsh
- Chacun a ses avantages : rappels des dernières commandes, commandes internes évoluées etc..
- Le plus courant en environnement Linux est bash
- Si on programme des ShellScript on doit choisir entre un développement en sh qui est le plus répandu et bash qui est le plus puissant

Les gestionnaires de fenêtre

- s'appuient sur la couche X11
- Il en existe des légers comme XFCE ou des plus lourds comme KDE
- Le choix médian sur les distribution pour PC se porte souvent sous gnome
- En embarqué, on n'utilise pas de couche X11 et si on possède un écran on l'adressera en mode framebuffer soit avec Qt soit avec Android

Qt

- est une librairie C++ fournissant 1200 classes et fonctionnant sous Windows, Linux, Windows et aussi en mode framebuffer
- fonctionne également en mode Android et en mode IOS, permettant avec un même développement de fournir un logiciel dans tous ces environnements.
- existe depuis 1993 et un gestionnaire graphique basé sur Maemo est moins utilisé qu'Android mais est très présent en embarqué graphique.

Licences

Rubriques

- Les distributions
- Les licences
- Différences entre les licences

Les distributions

Les distributions

- Linux courantes sont Debian, Redhat, Ubuntu, CentOS, Arch Linux, Gentoo et une centaine de dérivations en fonction des besoins spécifiques.
- En embarqué ces distributions ne sont pas utilisées même si on trouve des dérivés comme Raspbian ou ArchLinux pour RaspBerry.
- Les distribution pour Olimexe sont Yocto, Debian ou Buildroot
- Souvent ces distributions fournissent un environnement de bureau complet sous carte ARM ce qui n'est généralement pas le but des installations embarquées

Les outils de mise à jour

- Il existe plusieurs familles de logiciels de mise à jour couplés avec des serveurs de mise à jour appelé des dépôts.
- Sur les souches RedHat, les commandes se font au moyen d'instructions rpm
- Sous Debian, les commandes se font au moyen de l'instruction apt-get
- En embarqué, la problématique est quelque peu différente : on travaille sur des équipement pas forcément connectés en milieu industriel. On sera souvent amené à:
  - gérer plusieurs versions de carte dans le temps
  - gérer un environnement de mise à jour par retour atelier en fonction des pannes

Les licences

Objectif

- Nous allons déterminer l'ensemble des licences que l'on trouve dans le monde du libre ainsi que les termes d'utilisation de logiciels
- Il est important de connaître la philosophie des licences car certaines d'entre elles limiteront la diffusion de vos logiciels
- Il faut également savoir les limitations de chaque licences et procéder au choix correspondant à vos aspirations.

Free vs Open Source

- **Free** veut dire gratuit
- **Open source** veut dire que vous mettez à disposition les sources
- Mais attention :
  - Free et Open Source ne veulent pas dire que vous avez le droit de distribuer le logiciel
  - Le mode de distribution se fait généralement dans le cadre d'une licence commerciale ou d'une licence dite libre
- Certains logiciels libres ne sont régis par aucune licence ce qui rend floue leur utilisation dans un cadre professionnel
- Un auteur peut très bien inclure des clauses fantaisistes en lieu et place de licences
- Les licences sont faites pour éviter de convoquer un juriste pour étudier les clauses de chaque logiciel open source utilisé.

Avertissement

- Les définitions des diapositives suivantes présentent les concepts pour aider le développeur à comprendre les champs d'utilisation des licences.
- L'exposé ne couvre pas :
  - le droit de propriété industrielle
  - la notion de brevet
  - les preuves d'antériorité
- En outre, des jurisprudences par pays rendent certaines licences plus vulnérables dans certains pays
- Il existe des juristes spécialisés dans les droits des logiciels libres.

L'objectif

- **Licence :** est un contrat «par lequel le titulaire des droits du logiciel autorise un tiers à poser des gestes qui autrement les enfreindraient.»
- Le terme Contrat de Licence Utilisateur Final (**CLUF**) est une traduction du terme anglais **EULA**, End User License Agreement. C'est un contrat liant une personne installant un logiciel affecté par ce type de licence sur un/son ordinateur et l'éditeur du logiciel
- Protection contre les vices cachés : **As is** L'éditeur du logiciel décline donc toute garantie en cas de fonctionnement défectueux et se réserve le droit de faire payer les corrections.

Les formes de licence

- **Licence fixe :** est conçue pour être installée sur un ordinateur particulier
- **Licence nominative :** est attribuée à un utilisateur particulier, qui peut l'installer sur tout ordinateur, mais est le seul utilisateur agréé à l'utiliser.
- **Licence flottante :** fonctionne avec un ordinateur serveur de licence(s) : celui-ci décompte le nombre de licences utilisées à un instant T sur le réseau
- **Shareware :** (ou partagiciel) attribue un droit temporaire et/ou avec des fonctionnalités limitées d'utilisation
- **Licences libres :** sont une forme particulière de licence

Les principales licences de logiciels libres

- GPL
- LGPL
- Apache
- Licence X11
- Licence Publique Eclipse
- BSD
- Licences sur les contenus

Différences entre les licences

GPL

Nom	GPL
Appellation	GNU Public Licence
Version	3
Modification	**oui sous licence GPL**
Diffuser	**oui**
Version dérivée	**oui sous licence GPL**
Remarque	la plus répandue
Livraison sources	**obligatoire**
Tarif	**Frais d'acte de fabrication des supports**
Contrat de garantie	OK par une société de logiciel libre

LGPL

Nom	LGPL
Appellation	Lesser GNU Public Licence
Version	3
Modification	oui + adjonction parties non GPL
Diffuser	**oui**
Version dérivée	**oui**
Remarque	répandue
Livraison sources	**obligatoire**
Contrat de garantie	OK par une société de logiciel libre

Apache

Nom	Apache
Appellation	Software Foundation
Version	2
Modification	oui + ajout autre licence
Diffuser	**oui**
Version dérivée	on garde + adjonction
Remarque	Ok pour adjonctions commerciales
Livraison sources	**obligatoire**
Tarif	Libres
Contrat de garantie	Peut être fourni par une société de logiciel libre

X11

Nom	X11 ou MIT
Appellation	Licence MIT
Version	2
Modification	oui + ajout autre licence
Diffuser	**oui**
Version dérivée	oui en changeant la licence
Remarque	Ok pour adjonctions commerciales
Livraison sources	**obligatoire**
Tarif	Libres
Contrat de garantie	Peut être fourni par une société de logiciel libre

Eclipse

Nom	EPL
Appellation	Public Licence
Version	1
Modification	oui + ajout autre licence
Diffuser	oui
Version dérivée	oui en gardant la licence
Remarque	Ok pour adjonctions commerciales
Livraison sources	**obligatoire**
Tarif	Libres
Contrat de garantie	Peut être fourni par une société de logiciel libre

BSD

Nom	BSD
Appellation	Berkeley Software Distribution
Version	1999
Modification	oui + modification licence
Diffuser	oui
Version dérivée	oui sans respect de la licence
Remarque	Ok pour adjonctions commerciales
Livraison sources	Non obligatoire
Tarif	Libres
Contrat de garantie	Peut être fourni par une société de logiciel libre

Licences sur les contenus

- Il existe des licences qui ne s'appliquent pas aux logiciels mais aux documents et œuvres artistiques
- **CC :** (Creative Common) : se base sur des paramètres :
  - BY ou anonyme : recopiable en citant le nom
  - Commercial / Non commercial : si pas de commerce symbole euro ou dollar barré.
  - Modifiable / Non modifiable : si pas de modification symbole =
  - Partage des conditions identiques : symbole reload si oui.
- **GFDL :** de la FSF c'est le pendant de la licence GPL (copyleft) Elle permet la restriction à la modification.
- **LAL :**(Licence Art Libre) : Elle autorise tout tiers à procéder à la copie, la diffusion ou la transformation de l'œuvre. Elle est valide dans les pays ayant signés la convention de Berne. Elle est l'équivalent d'une licence CC : BY C

Compilation

Rubriques

- Compilation des sources
- Les librairies
- La librairie libc

Compilation des sources

Vérification d'intégrité

- Lorsqu'on récupère des sources on se verra proposer :
  - un fichier d'archive au format tgz bz2 ou xz
  - un fichier signature permettant de contrôler l'intégrité du fichier
- Une fois les deux fichiers chargés :
  - On lance la commande

```
md5sum fichier.tgz
```

- On peut le comparer avec l'étiquette md5sum contenue dans le fichier signature
- La signature sha1sum s'obtient de façon analogue à la signature md5sum

Décompression des sources

- On peut être amené à récupérer des sources sous plusieurs formats
  - TGZ : fichier.tgz (ou fichier.tar.gz) : se décompresse par

```
tar zxvf fichier.tgz
```

```
- BZ2 :  fichier.bz2 (fichier.tar.bz) : se décompresse par
```

```
tar jxvf fichier.bjz
```

```
- XZ : fichier.xz : se décompresse par
```

```
tar Jxf fichier.xz
```

- Sur les versions récentes on peut utiliser le paramètre automatique pour la décompression

```
tar axvf fichier.xz
```

Git SVN CVS

- **Git**, **SVN** ou **CVS** sont des gestionnaires de version qui permettent au développeurs de travailler ensemble sur les mêmes sources :
  - en gérant les modifications des autres développeurs
  - en pouvant récupérer la version d'une certaine date pour la tester
  - les serveurs git, svn ou cvs servent également de backup des sources
- On peut récupérer également les dernières sources en cours de développement avec une commande svn, cvs ou git
  - avantage : avoir des sources up to date
  - inconvénient : avoir des sources moins testés

Commandes de base CVS/SVN/git

- Avec chacun des gestionnaires, vous aurez à récupérer les sources et éventuellement de temps à autres à updater les sources ou au moins à vérifier qu'il n'y a pas de nouveautés
  - CVS :

```
cvs checkout adresse
cvs update
```

- SVN :

```
svn checkout adresse
svn update
```

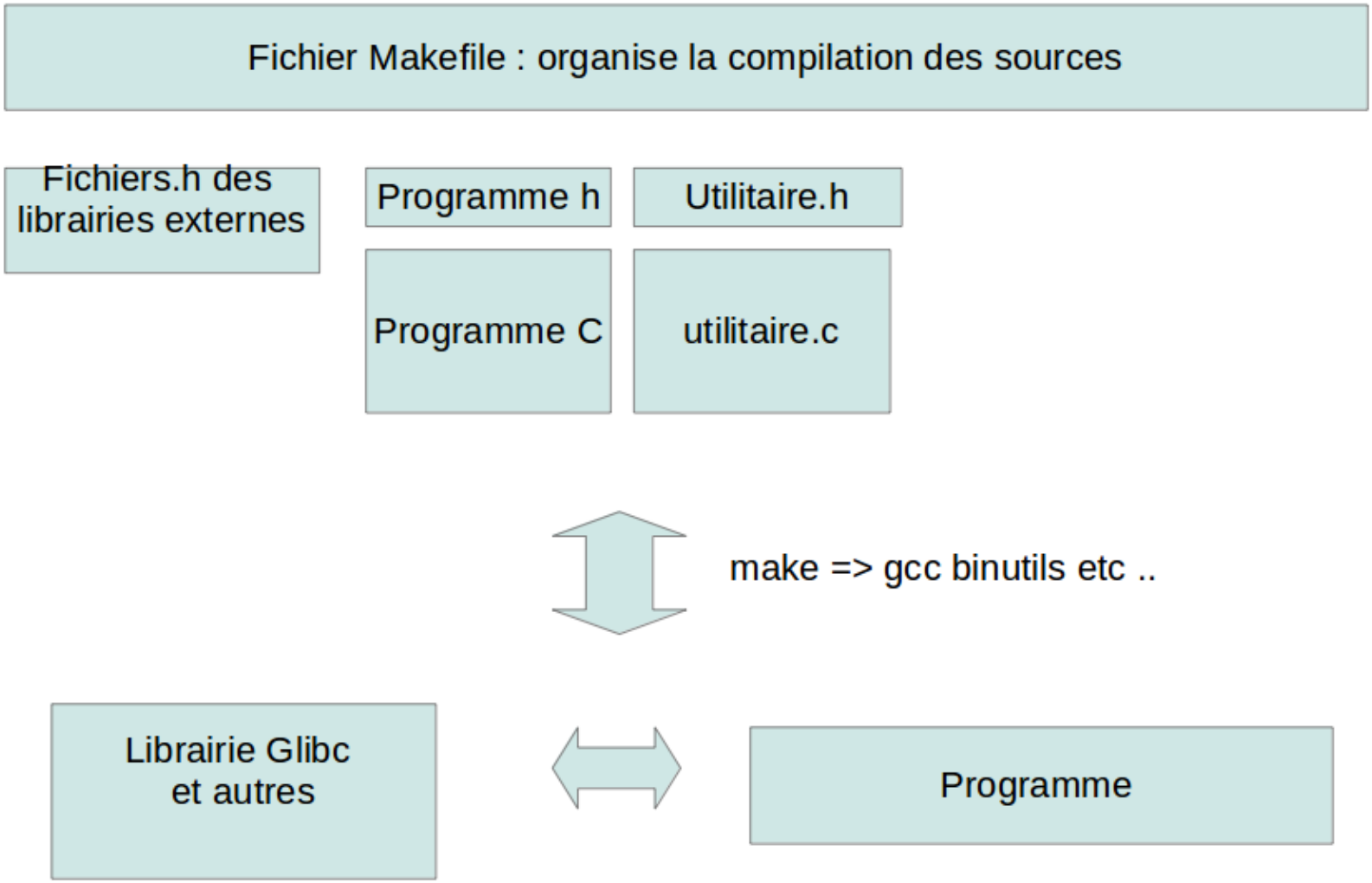
- Git

```
git clone adresse
git pull
```

Morphologies des programmes

- les programmes interprétés (Basic, Perl, Python, ShellSript)
  - le source analysé à la volée est transformé en code microprocesseur
  - l'interpréteur doit être présent sur l'équipement
  - interprétation est lente ne peut convenir à des tâches critiques
  - C'est très portable
- les programmes bytes compilé (Java)
  - le source compilé en «presque assembleur»
  - le résultat est transformé en code microprocesseur à la volée
  - C'est très portable et plus rapide
- les programmes compilés (C, C++)
  - nécessitent une transformation du code microprocesseur
  - c'est le plus rapide
  - il faut recompiler sur chaque processeur

Rappel mécanisme de compilation



Les librairies

Le principe des librairies

- Quand on code un projet on utilise parfois des librairies pour lesquelles on dispose:
  - des fichiers binaires rangés en principe dans /usr/lib avec une extension de fichier .so ou .a
  - de fichiers includes appelés header rangés en principe dans /usr/include
  - de documentations pour comprendre comment utiliser la librairie
- si ces librairies sont open source, on peut disposer des sources et les modifier mais ce n'est généralement pas nécessaire
- Dans son programme on inclut le header et quelque part on appelle la fonction voulue de la librairie

```
#include <malibrairiemp3.h>
fi=decodeMp3ToWav(file);
```

Édition de lien

- La compilation d'un programme produit un fichier .o

```
gcc -c exemple.o exemple.c
```

- Imaginons que ce programme ait besoin d'une librairie malibrairie est rangée dans /usr/lib/libmalibrairie.a
- l'éditeur de lien ld rassemble la librairie et exemple.o pour en faire un exécutable exemple

```
ld -L/usr/lib -lmalibrairie exemple.o -o exemple
```

- l'option -static force l'édition de lien à ne prendre que des librairies statiques.
- On peut aussi appeler l'édition de lien directement par gcc ou g++

Librairies dynamiques

- L'utilisation des librairies statiques produit un fichier binaire plus gros
  - si 1000 programmes utilisent la librairie malibrairie.a on aura 1000 fois le code de la librairie inclus dans des programmes
  - cela provoque une perte d'espace disque
- on peut préférer utiliser des librairies dynamiques d'extension so
  - dans ce cas, il faut être sûr que la librairie est bien installée sur la machine cible.
  - (rappelez vous des dll not found, ces dll sont des librairies dynamiques sous windows)
- Ainsi le développeur choisit le bon compromis entre les deux librairies :
  - un code compact mais il faut s'assurer que les librairies sont présentes
  - ou un code plus lourd qui englobe toutes les librairies et alourdit le système.

Utilisation des librairies

- Pour voir les librairies utilisées par un programme :

```
ldd programme
```

- Pour lancer un programme en indiquant un chemin temporaire pour une librairie :

```
LD_LIBRARY_PATH=/chemin/vers/la/librairie  programme
```

- Pour installer définitivement une librairie dans un emplacement déclarer le chemin dans un fichier conf présent dans /etc/ld.so.conf.d et ne pas oublier de lancer la commande

```
ldconfig
```

Librairies C utilisées en C++

Les Makefiles

- Ce sont des fichiers nommés **M**akefile (avec un M majuscule)
- Ils permettent de lancement de la compilation et de l'installation des programmes
- Ils agissent en cascade de répertoire en répertoire
- Ils contiennent des variables parfois en grand nombre
- À la fin de ces fichiers on trouve des étiquettes de la forme :
  - all : help : clean : distclean : install : uninstall :
  - ces noms d'étiquettes sont des conventions qui peuvent varier
- `make clean` : efface tous les fichiers compilés
- `make all` : effectue toutes les compilations
- `sudo make install` : installe les logiciels une fois ceux-ci compilés

Configure

- Parfois vous disposez des sources d'un logiciel et vous ne verrez pas le fichier Makefile mais uniquement un fichier configure.
- Cette commande configure doit être lancé par la syntaxe `./configure` éventuellement suivie d'options (./configure –help)
- Elle va vérifier que les composants nécessaires à l'élaboration du fichier Makefile sont présents sur votre machine de développement et si tous les composants sont présents, il va fabriquer le Makefile
- Mais attention les fichiers configure ne contiennent que les dépendances que le développeur à pensé y inscrire.
- L'option - `-prefix` permet d'indiquer où le logiciel sera installé (par défaut /usr/local/)
- Une fois que la commande ./configure ne donnera plus de message d'erreur, le fichier Makefile sera créé, on pourra alors lancer la commande

```
make
```

Problèmes de compilation

- Souvent on obtient des erreurs de compilation de programmes très éprouvés et réputés fonctionner.
- Ces erreurs peuvent être dues :
  - À votre version de Linux plus récente que celle qui a servi à mettre au point le programme et qui introduit de nouvelles instructions incompatibles
  - À la version de Linux qui utilise d'autres librairies que la version Linux qui a servi à l'élaboration du logiciel.
  - À des fichiers h ou des librairies que le compilateur s'attend à avoir sur votre système et qui ne sont pas présents (il faut alors les installer). Souvent on les trouve dans des packages nom-dev

Installation d'une librairie sous Debian

- Pour compiler un programme qui a besoin de la librairie libtruc on devra
  - installer la librairie libtruc par une commande :
    - `apt-get install libtruc` (sur les souches Debian)
  - installer les fichiers includes :
    - `apt-get install libtruc-dev` (sur les souches Debian)
- Sous les systèmes à paquetage rpm, les paquetage include sont nommés non pas dev mais devel
- Il peut arriver que les paquetages sous Debian se nomment devel s'ils proviennent d'un rpm converti en deb par l'utilitaire `alien`
- Exercice : installer Extreme tux racer à partir des sources

La librairie libc

Glibc

Nom	glibc
Licence	LGPL
Projet	GNU
Caractéristiques	standard, performance
Popularité	élevée
Taille en Mo	2,5
Source	http://www.gnu.org/software/libc/

uclibc

Nom	uclibc
Licence	LGPL
Caractéristiques	configurable, moins rapide
Popularité	élevée
Taille en Mo	0,5
Source	http://uclibc.org

eglibc

Nom	eglibc
Licence	LGPL
Projet	glibc
Caractéristiques	facile à compiler
Popularité	bonne mais projet intégré à glibc depuis 2014
Taille en Mo	1,5
Source	http://www.eglibc.org/home

dietlibc

Nom	dietlibc
Licence	LGPL
Projet	glibc
Caractéristiques	facile à compiler
Popularité	pas de mise à jour depuis 2013
Taille en Mo	0,2
Source	http://www.fefe.de/

Newlib

Nom	newlib
Licence	LGPL
Projet	sourceware
Caractéristiques	facile à compiler
Popularité	pas de mise à jour depuis 2013
Taille en Mo	0,5
Source	http://sourceware.org/newlib/

Exercice : construction de la librairie eglibc

- Télécharger les sources de la librairie eglibc
- Décompresser l'archive
- Suivre la documentation de compilation
- make
- **Ne pas installer**

Le noyau

Rubriques

- Présentation
- Configuration
- Options de compilation

Présentation

Présentation

- On peut compiler le noyau avec un maximum de modules drivers destinés à fonctionner sur toutes les plate-formes : c'est l'optique retenue par les distributions comme Ubuntu, Mandriva, RedHat etc.
- Le noyau sera de taille acceptable et sera fourni avec un grand nombre de modules correspondant aux drivers
- Les modules se chargent en mémoire dynamiquement donc ils ne prennent pas beaucoup de place mémoire, mais ils en prennent beaucoup sur le périphérique de stockage
- On peut compiler le noyau avec un nombre minimum de drivers correspondants aux caractéristiques de l'équipement.
- Ceci engendre un noyau prenant moins de place, plus rapide à compiler et également plus rapide à l'exécution.
- On se passe des modules dans ce cas là, sauf si certains drivers ne sont fournis que sous forme de module par les constructeurs

Versions

- Sur les systèmes de version antérieurs à 3
  - les versions de décimale paires sont stables (2.6)
  - les versions de décimales impaires ne le sont pas (2.7)
- Sur les systèmes embarqués, la version encore rencontrée est la version 2.6
- Sur les systèmes actuels
  - sur les systèmes les plus modernes la version est 4.xx (commande uname -r)
  - Les versions rc sont les releases candidates c'est à dire les versions non encore officielles

Les sources

- les sources sont disponibles sur le site http://www.kernel.org
- on peut récupérer les sources via ftp http ou par une commande git.

```
git clone git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git \
linux-git
```

- Si vous souhaitez contribuer, la mise à jour se fait par des patches qui sont envoyés et pris en compte par les lieutenants chargés du contrôle des versions
- la liste des nouveautés se lit dans la liste des logs de git

```
git log
```

- Il n'est pas recommandé d'utiliser les versions en cours de développement
- La dernière version stable est la version 4.9
- les sources compressés font une taille approchant 100 Mo, le noyau comprend les drivers de toutes les cartes reconnues par Linux

Configuration du noyau

- On décompresse le noyau par la commande tar idoine de préférence dans /usr/src

```
cd linux-version
vi README
make mrproper
```

- Ensuite au choix (ces différents choix seront explicités un peu plus loin)

```
make config
make nconfig
make menuconfig
make xconfig
make gconfig
```

- on verra qu'on peut aussi modifier le fichier `.config`, mais avant nous devons parler des modules
- **NB :** Tout ce qui est explicité dans la configuration du noyau, sera utile pour la compilation de Busybox et de BuildRoot

Module / Kernel / None

- Pour chaque option de driver rencontrée dans la configuration du noyau on a le choix entre :
  - mettre le driver dans le noyau ;
  - le mettre en module ;
  - ne pas le mettre du tout.
- Il faut faire attention car un driver supprimé peut entraîner des erreurs s'il est nécessaire pour un autre driver
- On peut répondre à chacune des options par :
  - **true** (dans le kernel)
  - **m** (en module)
  - **false** (ne pas inclure)

Avantage / Désavantage des modules

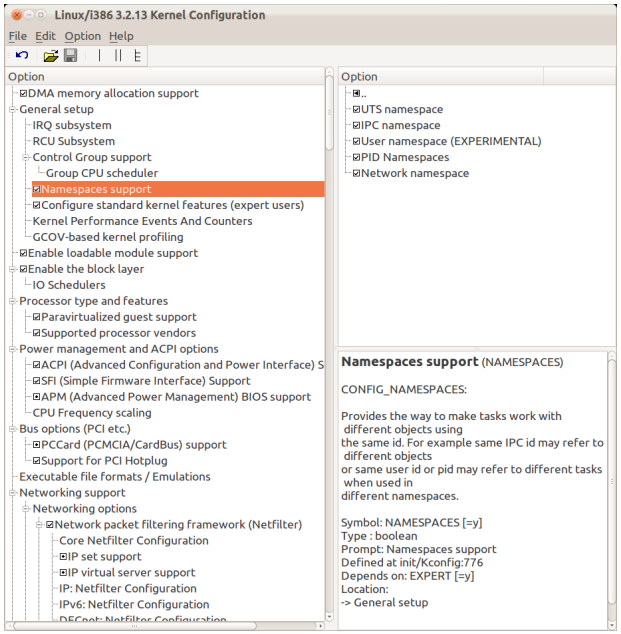
- Il existe des milliers de modules que l'on met en œuvre sur les distributions «tout terrain» (Ubuntu etc).
- Mais leur temps de compilation est important (>1 heure)
- Pour une version Linux propre à une carte embarquée on aura tendance à minimiser l'espace du noyau en flash et à refuser un maximum de modules pour ne garder que ceux vraiment utiles.
- Les noyaux sans module sont plus rapides à l'exécution

Commandes sur les modules

- La gestion des modules est automatisée par le système via la signature de chaque matériel
  - par son identifiant constructeur
  - son modèle
- On peut gérer les modules de façon manuelle pour des besoins de débugage:
  - **lsmod :** affiche la liste des modules montés
  - **modprobe module :** charge un module identifié par son nom partiel ou son nom avec chemin
  - **insmod :** installe un module identifié par son chemin
  - **rmmod :** désinstalle un module
  - **modinfo :** donne une information sur un module

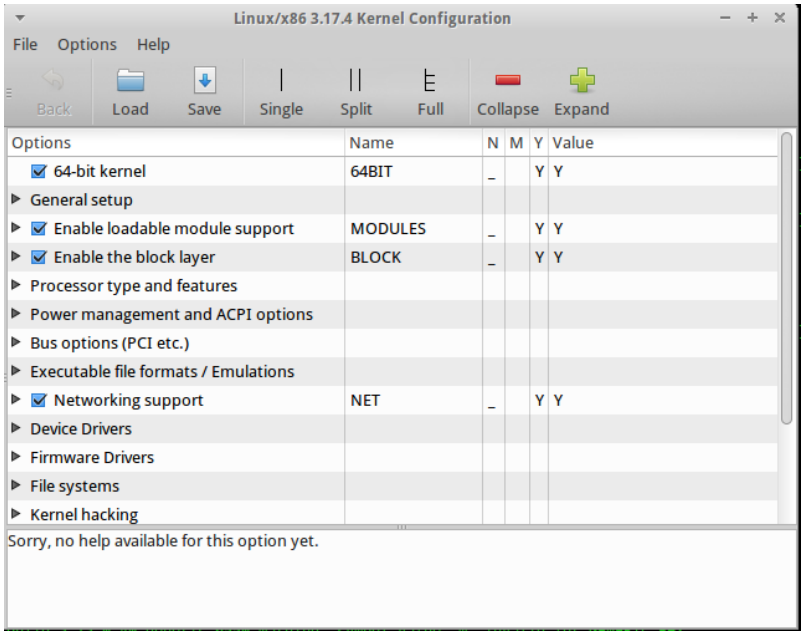
Configuration

make xconfig



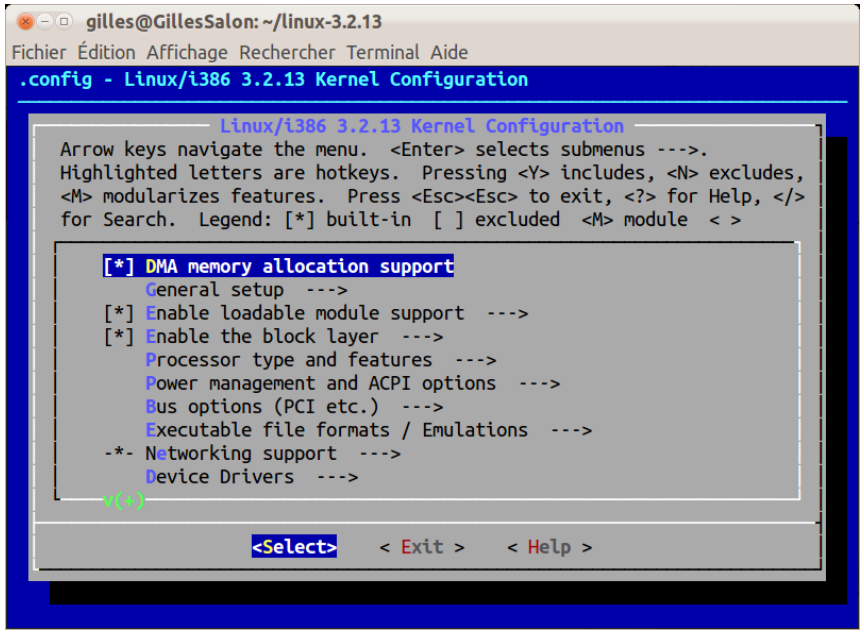
- est la façon la plus courante de configurer un noyau
- permet la recherche par mot clé d'un module ou driver ce qui est plus pratique que de rechercher dans l'arbre des catégories
- requiert la librairie libqt4-dev

make gconfig



- pas de recherche
- similaire à xconfig
- requiert libglade2-dev

make menuconfig



- quand on n'a pas x11 et requiert libncurses-dev
- La recherche se fait par la touche /

make nconfig

- ne requiert pas x
- requiert libncurses-dev
- la recherche d'un module se fait par la touche /

Autres possibilités

- .config : les options sont rangées dans le fichier .config qui est modifiable via un éditeur (dans les distributions, on trouve le fichier configure qui a servi à construire le noyau dans /boot)
- Pour positionner toutes les options du fichier .config et demande uniquement les nouvelles options qui n'existaient pas auparavant

```
make oldconfig
```

- Pour positionner toutes les options du fichier .config et positionner les nouvelles à leurs valeurs par défaut

```
make olddefconfig
```

- Pour positionner toutes les options aux valeurs par défaut

```
make defconfig
```

- Pour positionner toutes les options à oui (respectivement à non ou au maximum à module)

```
make allyesconfig
make allnoconfig
make allmodconfig
```

Les options de compilation du noyau

- **Prompt for development/incomplete code :** affiche ou masque les drivers qui ne sont pas considérés comme stables
- **Local version - append to kernel release :** permet d'ajouter une chaîne de caractères arbitraire au nom du noyau qui sera affichée avec l'option uname -r
- **Support for swap :** le swap est un mécanisme qui utilise le disque pour augmenter la mémoire. Doit être mis à NO sur les systèmes embarqués.
- **Configure standard kernel features (for small systems) :** permet de supprimer des options du noyau pour réduire sa taille. A ne pas trop utiliser
- **Loadable module support :** peut être supprimé pour avoir un noyau monolithique sans module dynamique
- **Enable block layer :** en mode CONFIG_EXPERT peut être supprimé si on utilise le mode FLASH NAND
- **Processor type and features :** à mettre à ARM avec les variantes correspondant au processeur
- **Preemptible Kernel :** (ON)
- **Network Support :** Unix sockets, TCP/IP, DCCP, SCTP, TIPC, ATM, Ethernet ;Bridging, QoS
- **SCSI :** est utile pour USB et pour les drivers IDE SATA et PATA
- **Input device support :** Mice, joystick, touchscreen, tablets, keyboard

Options de compilation (fin)

- **Characters devices :** ports séries, PTY driver pour SSH
- **I2C, SPI, 1Wire :** sont utiles
- Thermal sensor, Watchdog support, Multifonction drivers
- **Multimedia Drivers :** V4L, DVB, ALSA et OSS pour le son
- **Graphics supports :** Framebuffer
- **HID devices :** utiles
- **USB Support :** infrastructure, Host controller, devices drivers, gadget
- MMC/SD/SDIO, LED, RealTime Clock driver, Voltage an current régulator

Options de compilation

La compilation

- **make :** compile le noyau pour générer :
  - **vmlinux :** noyau au format RAW au format ELF utile pour débuguer mais ne fonctionne pas
  - **arch/arm/boot/Image :** l'image finale au format uimage
  - tous les modules au format *.ko
- Installation du noyau

```
make install
```

- Installation des modules

```
make modules_install
```

- Supprimer les fichiers générés :

```
make clean
```

- Supprimer également le .config

```
make mrproper
```

-Supprimer les backup produits par les patches

```
make distclean
```

Exercice : compiler un noyau

- Récupérer un noyau sur le site http://www.kernel.org
- Le décompresser
- Lire le fichier README.txt
- Récupérer l'ensemble des modules nécessaires via lsmod
- Lancer make xconfig
- Régler les options
- Lancer make

Installation du noyau

- Sur un système hôte pour installer un nouveau noyau on se contente de faire

```
make install
```

- quand on charge le noyau via le gestionnaire de paquetages
- Pour installer le noyau dans le répertoire cible en embarqué ou sur un média autre que /boot doit faire des opérations manuelles :

```
cp arch/i386/boot/bzImage /boot/vmlinuz-version
```

Installation des modules

- sur un système hôte :

```
make modules_install
```

- sur un système cible :

```
make modules_install INSTALL_MOD_PATH="
    ~/busybox-version/_install"
```

- Les modules de votre noyau seront installés dans le répertoire ~/busybox-1.19.4/_install/modules
- Les firmwares de votre noyau seront installés dans le répertoire ~/busybox-1.19.4/_install/firmware

Disque minimal en RAM

- Dans certains système Linux on créé un disque minimal contenant les drivers permettant de monter les périphériques nécessaires SCSI.
- on peut éviter cela en fabriquant ces périphériques dans le noyau et non pas en module (surtout vrai pour les modules SCSI)
- Sinon il faut créer un fichier initrd.img-3.2.9 par la commande

```
mkinitramfs  -o initrd-3.2.13 3.2.13
```

- En général on ne créé pas de disque RAM en embarqué.

Fichier config

- Une fois le noyau testé et installé, on copie le fichier .config qui a servi à la compilation dans le répertoire /boot avec
  - le nom du noyau.
  - ceci afin de pouvoir reprendre une compilation avec adjonction d'un driver

Busybox

Rubriques

- Busybox
- L'aide via Freenode
- Plus loin Avec Freenode

Busybox

Busybox : présentation

- Busybox contient l'ensemble des utilitaires minima pour Linux hormis le noyau, la librairie libC, les compilateurs et le lanceur
- On choisit les composants que l'on souhaite dans la configuration de Busybox qui se charge alors de les télécharger et de les compiler
- Busybox évite ainsi d'aller chercher l'ensemble de commandes gnu et autres sur les différentes ressources.
- On charge les sources de la dernière version sur le site http://www.busybox.net

Busybox : configuration

- Compilation de Busybox :

```
tar jxvf busybox-1.19.4.tar.bz2
cd busybox-1.19 .4
make menuconfig
```

- choisir les options souhaitées
- Ensuite
  - Save
  - Exit
- Enfin

```
make
make install
chmod 4755 ./_install/bin/busybox
```

Les commandes Busybox

- La liste des commandes fournies par busybox est données par la commande

```
./busybox --list
```

- La commande `./busybox --help` donne également la liste avec d'autres informations
- Pour appeler une commande par exemple `vi`

```
./busybox vi toto
```

- Pour obtenir de l'aide sur une commande

```
./busybox commande --help
./busybox wc --help
```

- Une fois installé les commandes sont accessibles via un lien symbolique :

```
./_install//usr/bin/cut -> ../../bin/busybox
```

Ajouter des commandes à Busybox

- On peut ajouter une commande à busybox
- Elle sera placée dans un répertoire des sources `./miscutils` :

```
 #include "busybox.h"
//usage:#define macommande_trivial_usage NOUSAGE_STR
//usage:#define macommande_full_usage ""
 # les lignes usages sont analysées
int macommande_main( int argc, char *argv[] ) {
  int i;
  printf("macommande :\n");
  for (i = 0 ; i < argc ; i++) {printf("arg[%d] = %s\n", i, argv[i]);  }
  return 0; }
```

- On ajoutera dans le fichier `./miscutils/Kbuild.src` l'entrée

```
lib-$(CONFIG_MACOMMANDE)     += macommande.o
```

- Ces syntaxes ayant tendance à changer on pourra chercher les lignes où apparaît une commande :

```
make clean
make cleandist
```

Finalisation d'ajout d'une commande

- On ajoute dans `./miscutims/Config.src` la commande (ordre alphabétique)

```
config MACOMMANDE
    bool "macommande"
    default n
    help
      macommande is a new test command.
```

- Dans le fichier `include/applets.src.h` ajouter (on demande que la commande soit dans /usr/bin et qu'elle n'ait pas le droit de passer en suid)

```
IF_MACOMMANDE(APPLET(macommande, BB_DIR_SBIN, BB_SUID_DROP))
```

- Ensuite la commande sera prise en compte par menuconfig

L'aide via Freenode

Aide

- En cas de problème, la principale ressource se trouve via IRC sur le réseau Freenode au canal #busybox (30 personnes environ)
- On trouvera également sur freenode une aide concernant le kernel sur le canal #kernel (300 personnes environ)
- Nous aborderons la partie RaspeberryPI qui dispose d'une aide sur le canal #raspberrypi (500 personnes environ)
- L'aide concernant les carte Olimex se trouve sur le canal #linux-sunxi
- Nous aborderons la partie Buildroot qui disponse d'une aide sur le canal #buildroot (120 personnes environ)
- D'autres aides peuvent être utiles dans le réseau Freenode que présentent les quelques transparents suivants

L'aide par IRC

- IRC est un protocole de chat qui est utilisé pour l'entraide dans les développements des logiciels libres
- Les réseaux IRC techniques Open Source se trouvent principalement sur Freenode pour la partie généraliste, mais on trouve également des serveurs IRC consacrés au langage Perl, à la distribution Debian (OTFC), aux Bitcoins
- L'utilisation d'IRC vient à remplacer les forums qui ont tendance à ne plus s'adresser qu'à des utilisateurs très novices.
- Nous allons apprendre à nous servir d'IRC et à respecter les usages en pratique sur ce type de réseau

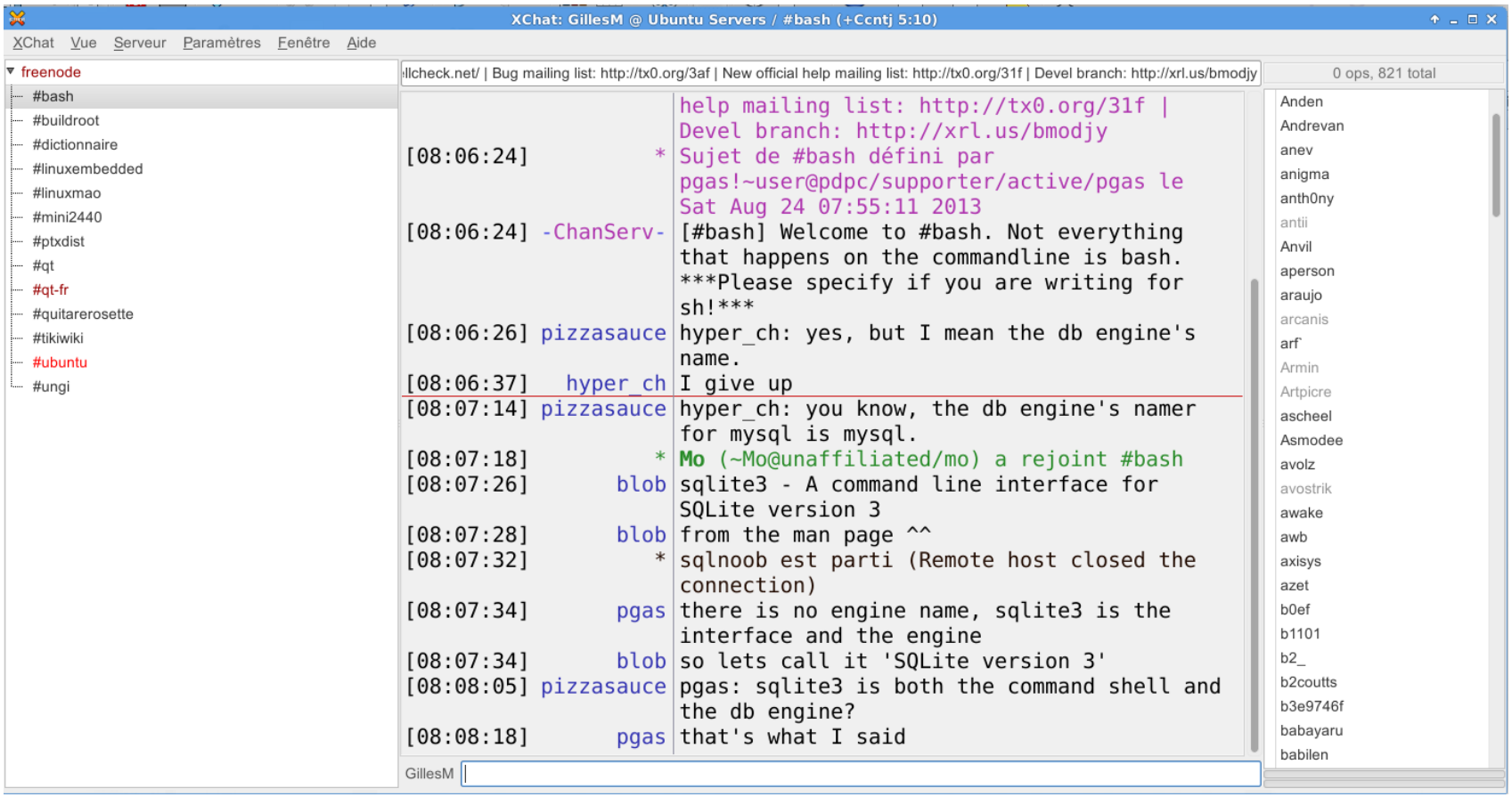
Logiciels IRC

- Logiciels spécialisés
  - On se connecte à Freenode par un logiciel spécialisé IRC (port 8001) : Linux(XChat, Quassel, Hexchat, Konversation, KVirc, Pidgin) Windows (mirc)
  - Ces logiciels, gardent la trace des précédentes discussions permettant de retrouver des informations contenues dans d'anciennes discussions, ils affichent la liste des canaux de discussions.
- On trouve pour certains réseaux notamment freenode une interface web http://webchat.freenode.net
  - qui nécessite de connaître le canal sur le quel on veut se connecter car la passserelle n'en fournit pas la liste.
  - c'est une interface qui passe par les ports 80 ce qui peut s'avérer pratique si on ne peut utiliser qu'un port http à cause d'un Firewall.

IRC : Présentation du réseau Freenode

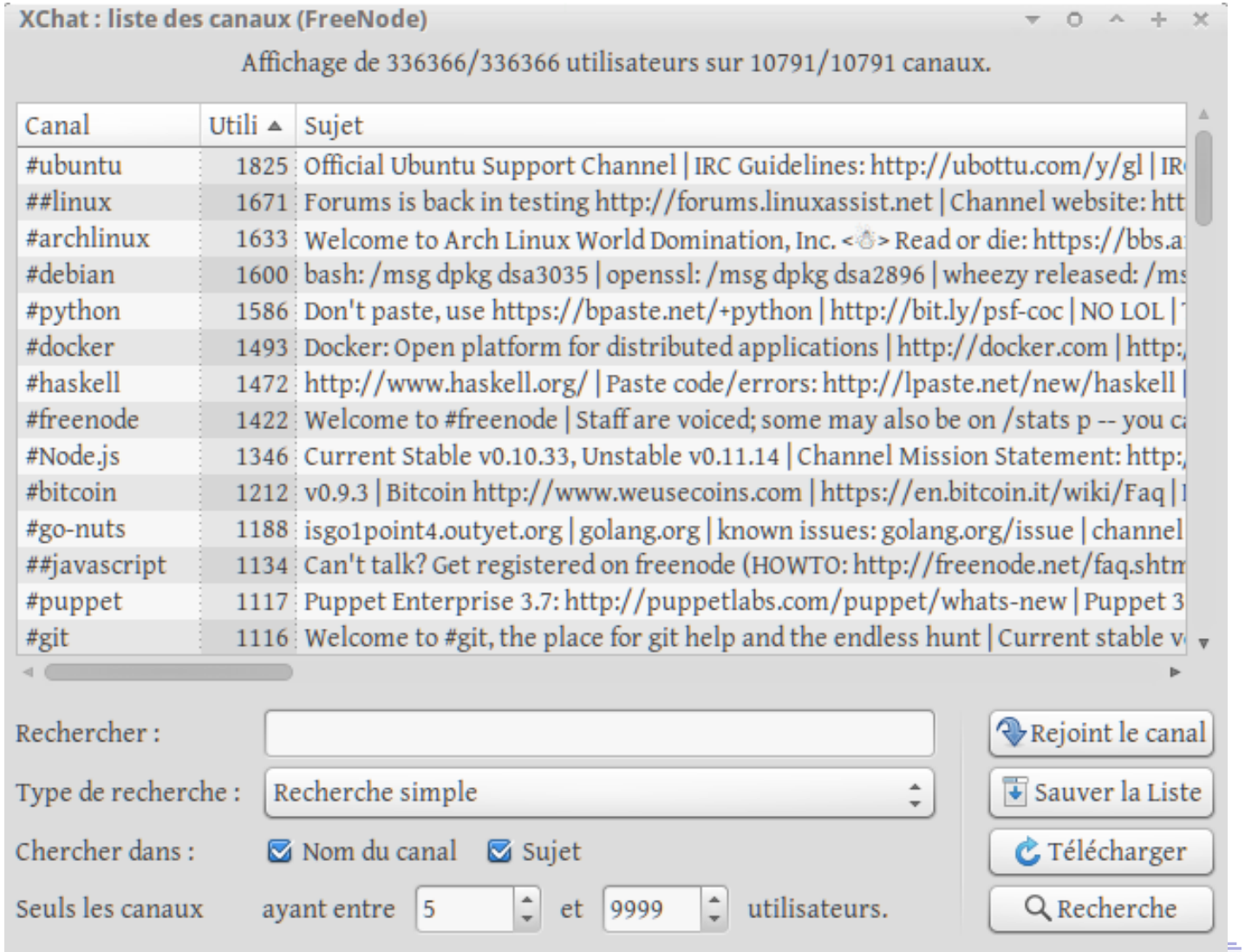
- **Freenode :** réseau de type IRC
  - 300 000 utilisateurs
  - plus de 10 000 canaux répartis par logiciel
- **Serveur :** irc.freenode.net
- **Port :** 8001
- Protection des mots de passe par mécanisme NICKSERV
- Utilisé comme canal d'aide par beaucoup de logiciels Open Sources.
- Passerelle Web à http://webchat.freenode.net

Exemple de canal



Plus loin Avec Freenode

Liste des canaux



Freenode : Nickserv

- Nickserv un service permettant de réserver son pseudo sur Freenode

```
/msg NickServ REGISTER password youremail@example.com
```

- Ensuite pour se connecter :

```
/msg NickServ IDENTIFY monnick password
```

- La connexion automatique se fait dans l'option de connexion du logiciel IRC

Quelques règles de bon sens

- Les personnes connectées ne sont pas obligées de vous répondre mais en général si elles sont devant leur écran elles vous répondront si elles connaissent la réponse à votre question
- À part si le canal se termine par fr ou si le topic (le sujet du salon) est en français la langue principale est l'anglais
- Ne vous présentez pas, n'indiquez pas que vous avez une question, posez la.
- Il est interdit d'envoyer des codes sources qui pollueraient l'affichage. Pour cela on utilise pastebin
- Le sujet de certains salons comprend des instructions précédées d'un point d'exclamation. Il s'agit de commandes robot qui vous donnent des premières aides

```
!getting_help
<knoba> GillesMM: "getting_help" : before asking your question, read the
!relevant_logs and !showconfig factoids, and prepare a single pastebin
containing all of that data. if you don't understand what this means,
or if you need help doing this, please let us know.
also see !pastebin
```

Pastebin

- Pastebin est un service Web permettant de copier du code pour ne pas polluer les fenêtres des salons
- Il est directement accessible depuis certains environnements de développement.
- Dans tous les cas, vous copiez les sources qui apparaîtront formatées dans le service pastebin ; une adresse URL vous est communiquée que vous indiquez dans IRC.
- Si vous accédez à pastebin par le web l'adresse est au choix : dpaste.org, fpaste.org ou pastebin.ca
- Le résultat à communiquer vous sera renvoyé par exemple

`http://pastebin.ca/3PYeZEgL`

- Les sites pastebin vous propose de conserver la trace pendant une durée de temps paramétrable. Pour des raisons de sécurité, ne laissez pas traîner à vie des fichiers de configuration qui pourraient finir sur google.

Imagebin

- Pour communiquer une copie d'écran explicitant un problème ou une solution vous utilisez imagebin à l'adresse : http://imagebin.org
- Sur le même principe que pastebin vous récupérez l'adresse de votre image que vous communiquez sur IRC
- Il est efficace de mettre des pastilles indiquant les zones que vous voulez souligner.(voir l'application gimp http://www.gillesmaire.com/tiki-index.php?page=draw-numbers)
- Dans l'exemple suivant, on montre aux développeurs de l'application Ardour un problème avec quelques pastilles numérotées via imagebin.org

Raspberry

Rubriques

- Architecture ARM
- Présentation de la carte
- Installation de base

Architecture ARM

Les processeurs ARM

- Architectures ARM, développées par ARM Ltd, sont des architectures RISC 32 bits (ARMv3 à ARMv7) et 64 bits (ARMv8)
- On les trouve dans les liseuses, les premiers smartphones, les tablettes, les consoles de jeu et tous les produits de basse consommation électrique
- Les processeurs sont vendus à des intégrateurs de cartes qui les revendent aux industriels/utilisateurs
- Les OS des processeurs ARM sont : Symbian, IOS, Linux, BlackBerry, Windows CE, Play Station Vita, Windows 10

Quelques termes à connaître

- **MMU :** permet l'adressage virtuel de la mémoire
- **MPU :** Protection de la mémoire
- **DSP :** composants électroniques pour les calculs Codec
- **FPU :** composants pour les calculs flottants avec l'extension VFP pour les calculs vectoriels
- **Jazelle DBX :** exécution des bytecodes Java (processeur portant l'extension J)
- **Thumb :** jeu d'instructions 16 bits sous ensemble des jeux 32 bits ou 64 bits permettant un gain de mémoire (processeur portant l'extension T)
- **Thumb2 :** jeu d'instructions 16 bits + instructions 32 bits minimales
- **MPCore :** Multi Processor Core
- **LPAE :** (Large Physical Address Extension) Adressage étendu que l'on trouve dans les processeurs ARM Cortex de 3 ème génération. Adressage 40 Bit 1To
- **big.LITTLE :** architecture permettant de faire fonctionner un processeur de très faible consommation la majorité du temps et d'utiliser un ou plusieurs cœurs plus puissants, mais consommant plus, lorsque les applications le demandent

Les familles de processeurs ARM

- **ARM1 :** 1985
- **ARM2 :** 1987 adressage 24bits
- **ARM3 :** 1988 FPU
- **ARM4 :** 1989
- **ARM5 :** 2000 Thumb ARM5T + DSP + gazelle avec ARM5J
- **ARM6 :** 1990 Jazelle 2001 - ARM1136J(F)-STM 2002 - ARM1156T2(F)-S 2003 - ARM1176JZ(F)-S 2003
- **ARM7 :** ARM720T MMU - ARM7TDMI - ARM7TDMI-S - ARM7EJ-S : DSP et Jazelle

Les processeurs ARM9 et ARM10

- **ARM9 :** 5 niveaux de pipeline sur les entiers, MMU) :
  - ARM920T (double cache de 16 Ko)
  - ARM922T (double cache de 8 Ko)
  - Famille ARM9E ARM946E-S : DSP, double cache, MPU, 1 port AHB
  - ARM926EJ-S: DSP, double cache, MMU, 2 ports AHB
  - ARM966E-S : DSP, double cache, MPU, 1 ports AHB
- **ARM10 :**
  - Famille ARM10E ARM1020E : DSP, 2x cache de 32 Ko, MMU
  - ARM1022E : identique au ARM1020E, sauf 2xcache de 16 Ko ARM1026EJ-S
- **ARM11 :**
  - Famille ARM11 : SIMD, Jazelle, DSP, Thumb-2

Les Cortex

- Famille Cortex-A,

1. génération (2009) Cortex-A8 (2009)
2. génération : Cortex-A5 MPCore 2 eme génération (très basse onsommation) et Cortex A9 MPCore (évolution du A8)
3. génération : Cortex-A7 MPCore (2013) - A15 MPCore (2012) A17 MPCore (2014)
4. génération : Cortext-A50 (A53 et A57) 2014

- Famille Cortex-R, processeur temps-réel: Architecture ARMv7-R Cortex-R4
- Famille Cortex-M, processeur embarqué : ARMv6-M et ARMv7-M

Présentation de la carte

Les cartes Raspberry

- Les cartes Raspberry Pi sont des cartes à processeur ARM conçues par David Braben, dans le cadre de sa fondation Raspberry Pi2.
- Il existent plusieurs cartes Raspberry
  - Modèle A : (ARM Cortex A6 700Mhz) 1 vidéo composite 1 HDMI 256 RAM 1 USB 25 €
  - Modèle A+ : SD => MicroSD 20 €
  - Modèle B : JTAG I2C RAM 512Mo 35 €
  - Modèle B+ : 4 USB Micro SD 3,5W => 3 W
  - Modèle Pi 2 : (ARM Cortex A7 900 Mhz 32 bits) 4 cœurs 1 Go RAM 35 €
  - Modèle Pi 3 : (ARM Cortex A53 1200 Mhz 64 bits) 4 cœurs 1 GO RAM Wifi Blutooth 45 €
- Doivent être ajoutés à la carte (PI2)
  - une alimentation micro USB
  - Une carte Micro SD
- Peuvent être ajoutés à la carte
  - USB Wifi
  - Écran HDMI
  - Boîtier
  - Caméra

Carte Raspberry PI 2

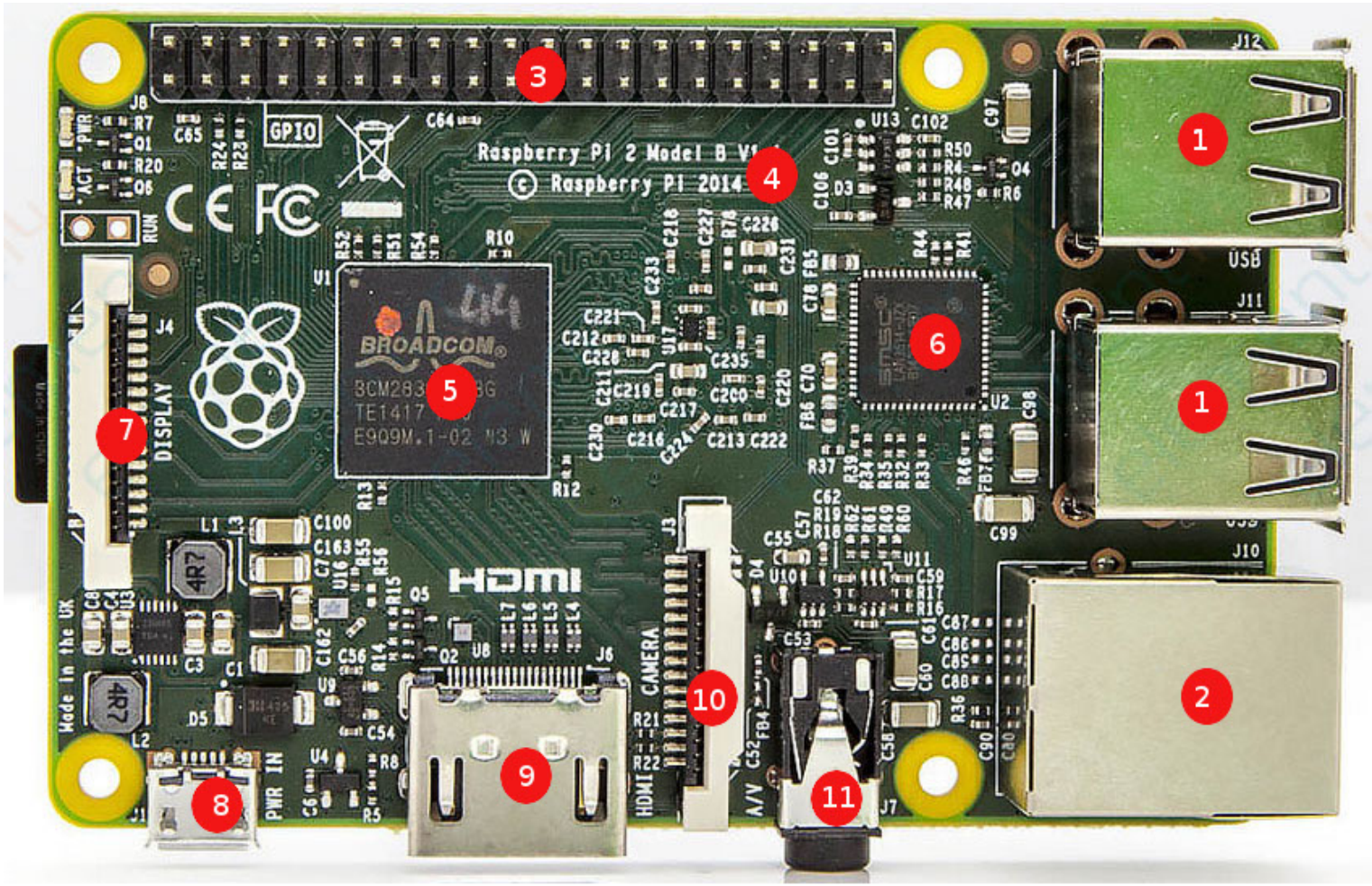


Figure 1: La carte de face

Carte raspberry PI 2

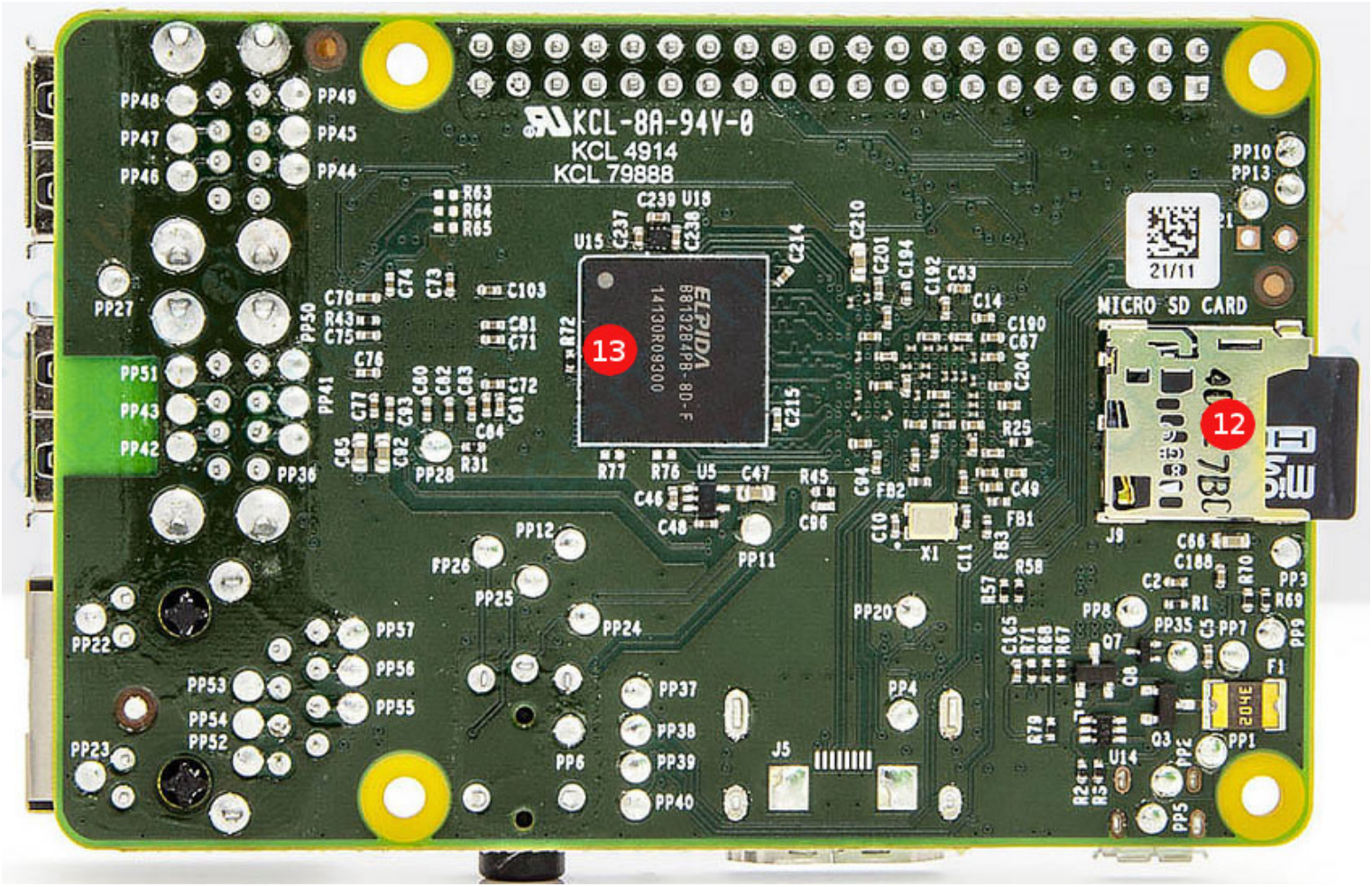


Figure 2: La carte de dos

Légendes

1. 4 ports USB 2
2. 10/100 Ethernet
3. 40pin extended GPIO
4. Références de la carte
5. Broadcom BCM2836, quatre cœurs ARMv7 à 900 MHz (Cortex A7)
6. Composant SMSC de la carte réseau
7. Interface Touch Screen
8. Port d'alimentation micro USB
9. HDMI
10. Port CSI Camera
11. Port audio
12. Slot Micro SD
13. RAM chip Elpida EDB8132B4PB-8D-F

OS disponibles

- NOOBS : utilitaire permettant de faciliter l'installation d'un OS pour les débutants
- RASPBIAN : OS basé sur la distribution Debian, c'est l'OS de référence de la carte
- ArchLinux : optimisée pour la carte
- Gentoo : version de base qui se configure pour la carte
- Raspbmc : distribution orientée média center
- Buildroot : fabrication d'une distribution sur mesure pour l'embarqué
- Windows 10 : nouveauté de Microsoft

Installation de base

Principe d'installation

- L'installation des Raspberry ne nécessite pas de mécanisme de flashage de la mémoire Flash :
  - pas d'Uboot
  - pas d'accès à la carte par un port série
- L'installation se fait une image sur la carte micro SD
- Nous allons apprendre à fabriquer une image et à l'installer dans un deuxième temps
- Dans un premier temps on peut installer une image toute faite : celle de raspbian que l'on récupère généralement sous la forme d'un fichier zip
- On insère la carte Micro SD dans un Socket SD que l'on insère dans un lecteur de carte SD d'un ordinateur sous Debian
- Une commande `mount` nous montre le point de montage qui est généralement de type `vfat`

```
/dev/sdc1 on /media/gilles/3232-3838 type vfat
```

- Dans ce cas, le périphérique à formater est `/dev/sdc`

Formatage via fdisk

- La commande de formatage permet de restituer une carte vierge en format VFAT, elle est également utile pour vérifier que la carte fonctionne correctement.
- Cette étape peut être sautée car la commande dd formate directement la sdcard
- Pour formater une sdcard en mode root faire :

```
umount /dev/sdc1
fdisk /dev/sdc
```

- Nous sommes en mode interactif

```
m : affiche l'aide
p : affiche la liste des partitions
d : supprime la dernière partition
n : crée une partition (à l'invite laisser les options par défauts)
w : sauvegarde les opérations
```

- Dans notre cas faire p,d,p,n,w
- Ensuite on formate la partition au type ext3 par exemple :

```
mkfs.ext3 /dev/sdc1
```

Copie de Raspbian sur la carte

- Toujours sur l'ordinateur de bureau sous Debian faire

```
dd bs=4M if=chemin_vers_le_img_de_raspbian of=/dev/sdc
```

- retirer la carte SD
- retirer la carte micro SD du support SD
- insérer à carte Raspberry éteinte la micro SD sur la carte
- La commande raspi-config permet de configurer l'OS au démarrage ou dans une console une fois Raspbian lancé

Compilation croisée

Rubriques

- Présentation
- Compilation du compilateur
- Appels compilateurs/compilateurs croisés

Présentation

Compilation croisée

- Compiler c'est transformer du code en langage humain C, C++ ou autre vers le langage de la carte
- Cela nécessite
  - un compilateur,
  - un éditeur de lien,
  - un outil de configuration,
  - un outil d'automatisation de la compilation make
  - la librairie glibc
  - d'autres librairies additionnelles
- Il est impossible de charger tous ces outils sur la machine cible
- Donc on compile sur un PC puissant sous Linux ou sous Windows et on transfère les programmes sur la carte cible
- c'est ce qu'on appelle la compilation croisée
- nous nous intéresserons à deux langages le C et le C++

Machine Hôte

- La machine Hôte est généralement sous Linux mais peut être sous Windows aussi bien en natif, qu'en mode Linux Virtualisé
- Nous basons cette formation sur l'utilisation d'une machine hôte sous Linux Ubuntu pour les raisons suivantes
  - ubuntu offre les dernières versions des logiciels
  - les documentations et forums sont en grand nombre
  - La machine hôte sert à préparer les logiciels c'est à dire à les compiler

Les compilateurs C et C++

- http://gcc.gnu.org/
- Le nom du compilateur C est gcc, le nom du compilateur C++ est g++
- Le compilateur fournit les fichiers entêtes pour ses fonctions
- gcc regroupe les compilateurs suivants : C, C++, Objective-C, Fortran, Java, Ada, and Go
- quand on compile gcc, on spécifie au moment de la configuration la liste des langages supportés par l'option –enable-languages=c,c++ etc
- Gcc est assez compliqué à compiler
- on peut par argument intégrer as et ld dans la compilation

Le compilateur GCC (suite)

- Le compilateur sait produire du code natif pour les processeurs suivants :
- ARM, AVR,Blackfin, CRIS, FRV, M32, MIPS, MN10300,
- PowerPC, SH, v850, i386, x86_64, IA64, Xtensa
- Pour chacun de ces processeurs des sous variantes sont possibles. Par exemple
- un processeur ARM peut contenir les extensions thumb ou un jeu d'instructions plus ancien comme armv4t armv5t
- pour la famille des processeurs x86 on a les architectures pentium, i88, i38 etc

Contenu d'une chaîne croisée

- Les binutils qui sont les programmes indépendants du C
- Le compilateur C
- Le compilateur C++
- Les entêtes des fonctions du C
- Les entêtes des classes du C++
- Les entêtes d'une version du noyau
- Les entêtes des librairies additionnelles
- Les librairies de calcul mathématique si votre processeur cible ne sait pas faire des calculs mathématiques
- La librairie système C

Les binutils

- sont les utilitaires permettant de traiter les objets une fois la compilation effectuée (http://www.gnu.org/software/binutils)
- **addr2line :** utile pour le débugage permet de connaître la ligne et le programme correspondant à une adresse dans un exécutable
- **ar :** permet de retrouver, de créer, de modifier les fichiers d'une archive (utilisé pour la gestion des librairies)
- **as :** l'assembleur qui sait gérer beaucoup de processeurs ainsi que leurs variantes (faire man as est instructif)
- **c++filt :** ce programme, utilisé pour les programmes en Java et C++, permet de reconstituer les noms des méthodes surchargées (même nom arguments différents) car en assembleur les noms de ces méthodes ont été différenciés

Binutils (suite)

- **elfedit :** met à jour les entêtes des fichiers ELF
- **gprof :** lit le profil d'exécution permettant de connaître le temps passé dans chaque procédure (option -pg au compilateur)
- **ld :** c'est le linker qui rassemble les différents objets issus du C ou C++ ainsi que les objets issus des librairies
- **nm :** liste les symboles d'un fichier objet
- **objjcopy, objdump :** recopie ou dump les fichiers obj en permettant leur relocation
- **ranlib :** génère l'index d'une librairie et la stocke dans celle-ci
- **readelf :** affiche des informations au sujet des fichiers ELF
- **size :** affiche les sections taille d'un objet ou de chaque objet d'une librairie
- **strings :** affiche les chaînes de plus de quatre caractères présentes dans un fichier
- **strip :** supprime les symboles d'un programme, ce qui le rend plus compact mais impossible à débuguer

Les librairies de traitement mathématique

- Certain processeurs fournissent des jeux de calculs numériques d'autres pas.
- Pour ceux qui ne les fournissent pas on a deux solutions
  - demander au noyau de faire les calculs, ce qui reste très lent (hard float code)
  - utiliser des librairies spécialisées GMP (http://gmplib.org) et MPFR (http://www.mpfr.org) qui corrige les arrondis.
  - En fait on utilise MPFR de l'INRIA qui utilise elle même GMP

Les entêtes du noyau

- Le compilateur GNU C étant proche du système Linux, il comporte un certain nombre d'appels système appelant le noyau.
- Donc le compilateur GCC fournit des entêtes propres à un noyau Linux
- Ainsi on doit savoir quand on installe une chaîne de compilation pour quels entêtes de noyau elle a été prévue.
- Si une chaîne de compilation a été prévue pour un noyau différent que celui qu'on désire utiliser, on peut utiliser un noyau plus récent ou plus ancien mais on court des risques d'erreurs de compilation théoriques sur les structures de données différentes.

Compilation du compilateur

Chaîne de compilation croisée

- Se mettent en œuvre sous Linux, bien qu'elles puissent être fabriquées sur MacOsX ou sous Windows
- Il existe plusieurs possibilités pour se procurer une chaîne de compilation de compilation croisée
- ex nihilo : on charge le compilateur sur le site GNU et on le prépare pour la cible (c'est assez complexe)
- on prend le compilateur du constructeur de la carte mais il est généralement peu récent et peut poser des problèmes pour compiler de nouveaux logiciels
- on prend une chaîne open source déjà prête ou une chaîne d'aide à la fabrication.

Alternatives à la compilation croisée

- On peut choisir de développer sur une carte peu puissante mais dotée d'un connecteur SD ou USB afin d'y placer le compilateur et les fichiers nécessaires à la compilation
- On peut choisir de compiler en mode émulation en utilisant qemu et en compilant sur une machine hôte
- Mais ces pratiques restent minoritaires

Les chaînes précompilées

- **CodeSourcery :** http://www.codesourcery.com/ chaîne commerciale très réputée mais tarde à délivrer les codes d'accès à son site
- **Linaro (ARM) :** http://linaro.org ne fournit que des chaînes pour les processeurs Cortex 18 A9 récents. Un bon choix
- **DENX ELDK** : http://www.denx.de/wiki/DULG/ELDK une bonne chaîne de compilation croisée, un uboot et autres utilitaires
- **Scratchbox :** http://www.scratchbox.org moins riche que les précédentes
- **Fedora ARM :** http://fedoraproject.org/wiki/Architectures/ARM tentative de Fedora
- **Embedded Debian cross-tools packages :** http://www.emdebian.org/tools/crosstools.html Chaîne pratique fournie par Debian
- **FreePascal :** http://www.freepascal.org/download.var chaine du projet d'IDE Lazarus

Les chaînes de fabrication de cross compilateurs

- **Crosstool-NG :** <http://crosstool-ng.org> populaire Supporte Glibc mais pas les versions plus légères uclibc et eglibc.
- **Buildroot :** <http://buildroot.uclibc.org> supporte de plus en plus de cartes et des librairies libc. Un bon choix.
- **Open Embedded :** <http://www.openembedded.org> un script de fabrication basé sur le script Bitbake toujours actif.

Les composants de la chaîne

- Header du noyau
- Outils de développement : binutils (qui assemble les fichiers objets), autoconf (qui fabrique les make)
- Compilateur : gcc g++
- Librairies : glibc (gestion fichiers, allocation mémoire) ou uclibc plus légère pour l'embarqué ou dietlibc ou newlib
- Librairies scientifiques : MPFR, GMP
- Souvent des patches

Processus de fabrication de la chaîne soi-même

- récupération des sources binutils gcc et noyau gmp mpfr
- compilation des librairies de précision arithmétiques
- compilation binutil en mode ARM
- récupération des header du noyau linux
- compilation gcc première passe
- compilation glibc
- production des entêtes de glibc
- compilation gcc avec les entêtes glibc mode statique
- compilation gcc mode dynamique

Appels compilateurs/compilateurs croisés

Édition de lien

- La compilation d'un programme produit un fichier .o

```
gcc -c exemple.o exemple.c
```

- Imaginons que ce programme ait besoin d'une librairie malibrairie est rangée dans /usr/lib/libmalibrairie.a
- l'éditeur de lien ld rassemble la librairie et exemple.o pour en faire un exécutable exemple

```
ld -L/usr/lib -lmalibrairie exemple.o -o exemple
```

- l'option -static force l'édition de lien à ne prendre que des librairies statiques.
- On peut aussi appeler l'édition de lien directement par gcc ou g++

Les étapes de compilation

- Quatre étapes
- Preprocessing
  - gcc -E programme.c > programme.i
- Compilation vers l'assembleur
  - gcc -S programme.i produit le fichier programme.s
- Assembleur vers le code machine
  - gcc -c programme.s produit le fichier programme.o
- Édition des liens
  - gcc -o programme programme.o produit l'exécutable programme
- Mais en général on fait les 4 en une étape raccourcie :
  - gcc -o programme programme.c

Exemples

- Production d'un fichier fichier.o (l'option -c s'arrête avant l'édition de liens)

```
gcc -c  fichier.c
```

- Deux exemples de production du fichier exécutable programme avec édition de liens

```
gcc -o programme  fichier1.o fichier2.o -lm
gcc -o prg fichier1.o fichier2.o -L/usr/local/lib -lpet -lreadline
```

- **Attention :** quand l'éditeur de lien est appelé par gcc ou g++ l'ordre des arguments passés est important : il faut que les ordres du linker soient à la fin.

Buildroot

Rubriques

- Présentation
- Utilisation
- Installation de la distribution

Présentation

Utilisation de BuildRoot

- Buildroot est un outil de fabrication d'un environnement Linux
- Le site Web de BuildRoot est http://buildroot.org/
- on peut récupérer la dernière version par git

```
git clone git://git.buildroot.net/buildroot
```

- **Attention :** l'adresse git a tendance à changer, en vérifier l'adresse sur le site
- On peut récupérer buildroot en format tgz sur le serveur
- Une documentation assez claire est disponible sur le site en langue anglaise
- On dispose d'une mailing liste et d'une aide freenode sur le canal #buildroot
- L'adresse du site buildroot.uclibc.org pourrait laisser penser que seule la librairie uclibc sera proposée, il n'en est rien comme nous le verrons.
- On peut utiliser Buildroot depuis son compte courant et son répertoire de travail sans être root

Buildroot : principe

- Buildroot est préconfiguré pour un ensemble de cartes du marché mais on peut fabriquer une configuration pour une carte non contenue dans Buildroot
- Buildroot automatise l'ensemble des tâches suivantes et permer de :
  - Downloader les fichiers sources nécessaires
  - Configurer, fabriquer et installer la chaîne de compilation croisée ou en utiliser une déjà installée
  - Compiler les composants demandés
  - Compiler un noyau
  - Compiler uboot ou équivalent pour les matériels fonctionnant avec uboot
  - Fabriquer différents formats de systèmes de fichier contenant la distribution
- Buildroot est donc utile pour maintenir l'ensemble des composants en permettant une recompilation régulière de l'OS et de ses composants

Paquetages nécessaires à Buildroot

- Obligatoires
  - which, sed, make (>version 3.81), binutils
  - build-essential (debian), gcc (>version 2.95), g++ (>version 2.95)
  - bash, patch, gzip, bzip2, perl, tar, cpio, python (version 2.6>), unzip, rsync
- Optionnels
  - git, ncurses5, qt, glib2, gtk, glade2, javac, jar, tools, asciidoc

Compilation de Buildroot

- Installer les paquetages nécessaires :

```
sudo apt-get install libncurses5-dev bison g++ flex \
gettext texinfo patch git-core libtool autoconf \
subversion
```

- La liste des fichiers de configuration des cartes supportées se trouve dans le répertoire `configs`. On peut également l'obtenir par la commande `make list-defconfigs`
- La configuration de Buildroot se fait comme celle de busybox ou du noyau au moyen des commandes `make menuconfig` ou autres

```
make raspberrypi2_defconfig
make raspberrypi3_defconfig
```

- ou bien copier le fichier `configs/raspberrypi3_defconfig` dans .config pour en faire le fichier de configuration par défaut

```
make xconfig
```

- ou lancer `make menuconfig` pour une configuration en mode texte
- La documentation pour la carte Raspberrypi3 se trouve dans le répertoire

Les paramètres qu'on peut ajouter

- Toolchain
  - Enable toolchain locale/i18n support OUI (localization)
  - Enable C++ support OUI (pour compiler Qt si on veut piloter un écran TV)
  - VCHAR
- Graphic libraries and applications (graphic/text)
  - Qt5
  - Graphics drivers
    - Qt Virtual Framebuffer (OUI si on veut Qt en framebuffer)

Utilisation

Exécution

- Fabrication de l'ensemble des composants demandés
- Les options make -jN pour activer la compilation parallèle **ne sont pas** supportées

```
make
```

- Si on souhaite récupérer l'ensemble des sources pour une compilation en mode déconnectée :

```
make source
```

- Une fois compilé on peut demander une configuration plus fine d'un des éléments :

```
make busybox-menuconfig
make uclibc-menuconfig
make linux-menuconfig
make barebox-menuconfig
make uboot-menuconfig
```

- On peut générer des graphes de dépendances inter paquet, de temps de génération ou de taille (nécessite l'installation de graphviz,python-matplotlib)

```
make graph-depends ; make graph-build ; make graph-size
```

Les fichiers générés

- Le répertoire buildroot/output contient des répertoires certains contiennent

  - Les répertoires utilisables
  - **output/images/ :** contient les images et le bootloader
  - **output/build/ :** contient tous les programmes composants sauf la toolchain
  - **output/host/ :** contient tous les binaires utiles pour la fabrication de buildroot. Ces binaires seront inclus dans le le filesystem qui sera envoyé sur votre carte. Seule la chaine de compilation a des raisons d'être utilisée sur votre poste de travail.
  - La chaine de compilation si vous l'avez générée se trouve dans

```
output/host/usr/bin/
```

  - Les répertoire non utilisables
    - **staging/ :** contient des fichiers intermédiaires
    - **target/ :** contient presque toute l'arborescence / de la cible (pas de /dev). Ces fichiers ne doivent pas être utilisés
    - **output/graphs :** contient les graphes

Les fichiers .config

- le fichier .config pour make menuconfig est situé dans `buildroot`
- le fichier .config pour make linux-menuconfig est situé dans `buildroot/output/build/linux-<version>/`
- le fichier .config pour make uclibc-menuconfig est situé dans `buildroot/output/build/uclibc-<version>/`
- le fichier .config pour male busybox-menuconfig est situé dans `buildroot/output/build/busybox-<version>/`

Utiliser la toolchain générée

- On souhaite compiler d'autres programmes en dehors de buildroot :
- la chaîne de compilation de buildroot est située par défaut dans `output/host/usr/bin`
- On y trouve les compilateurs g++, gcc, l'éditeur de lien ld, ainsi que l'ensemble des binutils
- Pour les utiliser, il suffit de déclarer :
  - `export PATH=$PATH:xxx/buildroot/output/host/usr/bin`
- L'exécutable de gcc s'appelle indifféremment:
  - arm-unknown-linux-uclibcgnueabi-gcc (permettant d'identifier la marque du compilateur)
  - arm-linux-gcc (un lien permettant d'utiliser le compilateur sans se soucier de sa provenance)

Utiliser une chaîne de compilation externe

- On peut choisir d'utiliser une chaîne externe pour gagner de temps de fabrication ou suivre des recommandations d'entreprise :

1. BuildRoot peut utiliser la chaine Linaro, Sourcery et BlackFin
2. BuildRoot ne peut pas utiliser les chaînes OpenEmbedded ou Yocto
3. Les autres chaînes n'ont pas été testées par l'équipe de BuildRoot

- Pour utiliser une chaîne réputée fonctionner on peut agir de trois façons :

1. Demander l'utilisation d'une chaîne externe et demander a BuildRoot de downloader et d'installer la chaîne
2. Indiquer à BuildRoot l'emplacement où trouver la chaine déjà installée sur votre disque. Dans ce cas désactiver `automatic download`
3. Utiliser une chaîne de compilation customisée (en général BuildRoot ou crosstool-NG). Il faut dans ce cas renseigner le path, le préfixe et les librairies externes

Installation de la distribution

Préparation de la carte Micro SD

- Une fois la distribution compilée on insère la SD card dans un lecteur de carte sur la machine de développement puis en étant super utilisateur

```
dd if=output/images/sdcard.img of=/dev/sdg
```

- Cette opération de copie crée un système de fichiers que l'on peut contrôler via la commande

```
fdisk -l /dev/sdg
```

- En cas de problème d'écriture sur la carte SD on contrôlera l'image output de la façon suivante (on note la taille des secteurs en principe de 512 octets)

```
fdisk -l output/images/sdcard.img
```

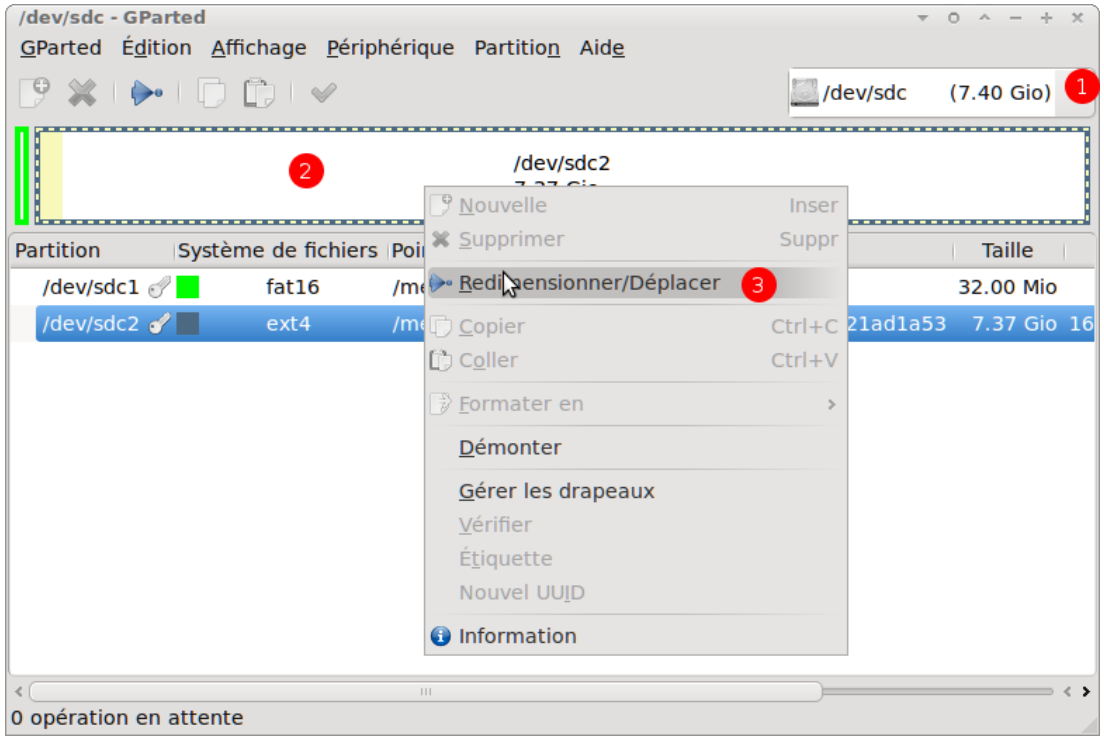
- Pour visualiser le contenu des partitions

```
mount -o loop,offset=512 output/images/sdcard.img /mnt
mount -o loop,offset=33554944 output/images/sdcard.img /mnt
```

- L'offset se calcule par : debut du bloc * 512
- Le début du bloc est donné par la commande fdisk

Agrandir la partition à toute la mémoire SD

- Le plus rapide est de passer par la commande gparted



1. Sélectionner la clé
2. Via le bouton droit de la souris faire apparaître le menu
3. Selectionner agrandir qui permet d'agrandir la zone à la souris.

- Ne pas oublier de sauvegarder

Divers réglages

- Si vous disposez d'un écran HDMI et d'un clavier vous pouvez vous connecter en donnant le nom de login root qui est fourni de base sans mot de passe
- Si vous ne disposez ni de l'un ni de l'autre :
  1. Vous pouvez installer un serveur ssh appelé `dropbear` qui est disponible dans BuildRoot
  2. Par contre le compte root n'ayant pas de mot de passe, vous ne pourrez pas vous connecter. Le plus simple est de donner un mot de passe à root (enable root password)
  3. Ensuite soit vous pouvez configurer une adresse IP fixe, soit la carte obtient une adresse d'un serveur DHCP (il faut que le client udhcpc soit installé et avoir spécifié etho comme interface). Vous pouvez installer un serveur DHCP et fournir une adresse IP connue pour pouvoir vous connecter. Si vous utilisez un serveur DHCP d'entreprise, la meilleure chose à faire est de dire à la carte d'envoyer en tftp l'adresse sur votre machine de développement.
- La carte dispose d'un port série sur le Bus GPIO (TxD sur port 8 GPIO14, RxD sur port 10 GPIO15) alimenté en 3.3V et non en 12V, ce qui nécessite un adaptateur.
- On peut également deviner une adresse IP dynamique à l'aide de la commande

```
arp -a
```

Customiser la cible

- La documentation (chapitre 9) de BuildRoot donne beaucoup de possibilités (ajout de comptes, de patches etc)
- Deux méthodes permettent de modifier la cible : Root filesystem overlays (BR2-ROOFS_OVERLAY dans le config de buildroot) et les scripts Post Build (BR2_ROOTFS_POST_BUILD_SCRIPT)
- **Root filesystem overlays :**

1. Dans le menu System Configuration on vous permet de spécifier un ensemble de répertoires qui seront copiés quand le build est fini mais avant de faire l'image.
2. Mettre le répertoire `board/nomdelacarte/common/rootfs_overlay`
3. Suivant la recommandation d'arborescence donnée dans le chapitre 9.1, on mettra les fichiers dans `board/nomdelacarte/common/rootfs_overlay/etc/init.d`
4. Ainsi les fichiers seront recopiés avec leurs attributs dans le répertoire /etc/init.d

- **Scripts Post Build :**

1. Sur le même principe que précédemment, les scripts sont appelés entre le build et la fabrication de l'image (ce ne sont pas des scripts appelés sur la cible !)
2. `board/<company>/<boardname>/post_build.sh`

Installer la tool chain dans QtCreator

- Dans Qt Creator aller dans le menu outils/Options
- Dans l'onglet Compilateurs Appuyer sur le ComboBox Ajouter et sélectionner gcc
  - Donner le nom arm-rasbperry-gcc
  - Renseigner le chemin du compilateur avec le compilateur croisé
- Ajouter le compilateur g++ de la même façon via le ComboBox ajouter
- Ensuite dans l'onglet Kits ajouter et choisir
  - le compilateur gcc et g++ que l'on vient d'ajouter
  - le qmake disponible dans le buildroot/output/bin
- Ensuite choisir ce kit dans le sélecteur de de kit (Release/Debug)

Divers

Rubriques

- Paramètres spécifiques
- Services réseau supplémentaires
- Ajout de programmes tiers

Paramètres spécifiques

Configuration de l'ordonnanceur

- L'ordonnanceur des tâches se fait au moyen de la crontab ancienne version.
- Elle est activable par Busybox dans la partie `Miscellaneous Utilities`
- Le répertoire /var/spool/cron/crontabs doit être créé
- Le programme /usr/sbin/crond doit être lancé par un service créé dans /etc/init.d (S80Cron)
- L'ordonnancement est créé par la commande crontab sous la forme

```
crontab -e
```

- On entre dans le fichier des lignes de la forme (ici toutes les minutes)

```
* * * * * /bin/date >> /tmp/date.txt
```

- le process sera lancé par l'utilisateur ayant lancé la cron.
- Les 5 étoiles peuvent être remplacées par des valeurs spécifiques sous la forme (voir page suivante)

```
mm hh jj MMM JJJ
```

Exemples lignes cron

- Tous les premiers du mois à 23 h 30

```
30 23 1 * *
```

- Tous les lundis à 22 h 28 :

```
28 22 * * 1
```

- Du 2 au 5 de chaque mois à 10 h 12 :

```
12 10 2-5 * *
```

- tous les jours pairs à 23 h 59

```
59 23 */2 * *
```

- Toutes les 5 minutes

```
*/5 * * * * df
```

Clavier français

- On charge le clavier français via la commande :

```
loadkmap </etc/azerty.map
```

- Pour générer le fichier azerty.map, il suffit de faire sur un système bien configuré doté du clavier :

```
sudo busybox dumpkmap >azerty.map
```

- Le programme dumpkmap se trouve dans busybox qu'on pourra compiler sur la machine de développement

Notes à propos des modifications de configuration buildroot

- Quand on modifie des options de génération par exemple par un make menuconfig, Buildroot doit dans certains cas reconstruire entièrement le système ou ne reconstruire que la partie concernée.
- La décision n'est pas prise automatiquement par BuildRoot.
- On doit tout reconstuire lorsqu'un élément d'architecture est modifié : processeur, format binaire, lorsqu'on change la chaîne de compilation croisée
- Lorsqu'on ajoute un paquet additionnel, on peut se contenter d'un make qui générera la cible de manière incrémentale sauf si l'élément modifié peut avoir des répercussions sur d'autres éléments, c'est parfois ou souvent le cas pour des librairies.
- Si vous supprimez un paquet, BuildRoot ne recompilera plus ce paquet mais ne le détruira pas physiquement de la cible, il faudra attendre la prochaine regénération complète. On peut dans ce cas supprimer le paquet dans output/target pour éviter que le paquet soit remis sur la cible et éviter de tout reconstruire.

Services réseau supplémentaires

Connexion sécurisée avec SSH et transferts de fichiers par SCP

- Les connexions ssh et l'envoi de fichiers via scp sont gérés par l'installation de `DropBear` dans BuildRoot
- De base l'utilisateur root n'a pas de mot de passe et donc la connexion ssh et scp sont refusées par le serveur
- Il faut dont ajouter un mot de passe à l'utilisateur root via la directive R2_TARGET_ENABLE_ROOT_LOGIN que l'on trouve dans la rubrique `System Configuration`
- Dans la rubrique BR2_TARGET_GENERIC_ROOT_PASSWD qui correspond à la case en dessous dans une interface de menuconfig, on doit entrer le mot de passe root.
- La connexion scp ne demande pas de réglage particulier, elle est intégrée dans `DropBear`

Serveur httpd de Busybox

- Le serveur de Busybox est le plus simple à mettre en œuvre
- Il se lance en démon ou en mode inetd avec l'option -i
- La Documentation se trouve en début du fichier output/build/busybox*/networking/httpd.c
- Les fichiers html doivent être dans la racine du serveur
- Les fichiers cgi doivent être dans le répertoire cgi-bin et être exécutables.
- Si ce sont des fichiers Perl la ligne shebang doit le préciser (#!/usr/bin/perl)
- Si ce sont des fichiers shell la ligne ser (#!/bin/sh)

```
#!/bin/sh
echo -e "Content-type: text/html\n\n"
echo -e "hello\n"
```

- D'autres serveurs plus compliqués à configurer sont disponibles via Buildroot : ligthttp, Boa, nGinx,Apache,tinyhttpd,thttpd

Ajustement d'horloge système avec NTP

- Le protocole client NTP est fournit via BR2_PACKAGE_NTP des NetWorking applications il permet de mettre à jour l'heure et la date par commande.
- Le protocole serveur NTPD permet de synchroniser à intervalle régulier le device via BR2_PACKAGE_NTP_NTPD sans passer la commande ntp dans un mécanisme de crontab
- Le serveur openntpd sert à la fois de client de mise à jour de l'heure mais il permet également aux autres équipement de se synchroniser.
- Le fichier de configuration est /etc/ntpd

Autres astuces et documentations

- Les logiciels que l'on trouve dans BuildRoot sont souvent des versions simplifiées des logiciels que l'ont trouve sous les distributions classiques
- Ils sont moins documentés et nous allons donner ici la méthode pour trouver la documentation pour chacun d'eux
- S'ils font partie de Busybox on a la documentation sommaires données par busybox command help
- Mais on peut trouver les sources dans le répertoire `output/build/busybox*/`. Par exemple networking/httpd.c donne la documentation du protocole httpd
- Pour les programmes issus de Buildroot, on doit aller dans le répertoire `output/build/`. Par exemple la documentation du protocole ntp est fournie dans le répertoire ntp-4.2.8p4

Ajout de programmes tiers

Ajout de programmes spécifiques

- Si on veut appliquer des modifications il serait maladroit de décompresser le système de fichier pour ajouer les différents composants, on peut demander à buildroot d'inclure ces modifications de façon automatique.
- On peut avoir à ajouter sur la carte un logiciel spécifique développé (BR2_ROOTFS_OVERLAY)
  - à partir d'un shell script ou autre langage de script
  - en C/C++ ou autre language avec compilation croisée
  - ou disposer de données sur le système de fichiers cibles
- On peut vouloir appliquer des patches sur un composant présent ou le supprimer (BR2_ROOTFS_POST_BUILD_SCRIPT)
- Plus généralement appeler une commande avant la génération du système de fichiers (BR2_ROOTFS_POST_BUILD_SCRIPT)
- On peut demander la modification des permissions et des propriétaires de fichiers (BR2_ROOTFS_DEVICE_TABLE)
- Ajouter des fichiers device (/dev) (BR2_ROOTFS_STATIC_DEVICE_TABLE)
- Appliquer des patches (BR2_GLOBAL_PATCH_DIR)

Les patches

- On peut être amené à charger des patches qui sont des fichiers différentiels entre deux sources (on regardera le contenu d'un fichier patch)
- Ces patches sont utiles que si on compile des versions de logiciels intermédiaires non encore stabilisés
- Un patch est obtenu par la commande

```
diff -Naur Repertoire.origine Repertoire.destination > patch1to2
```

- Pour appliquer un patch on fait

```
patch fichieroriginal  <patch1to2
patch -pn <patch1to2multiple
```

Niveaux de patch

- L'argument n est le niveau
- Si dans le fichier patch on a /usr/local/src/toto/toto.c pour le fichier original
  - -p0 indique le chemin complet
  - -p1 usr/local/src/toto/toto.c
  - -p2 local/src/toto/toto.c
  - -p3 src/toto/toto.c
  - -p4 toto/toto.c
  - -p5 toto.c ce qui est équivalent à ne pas mettre d'argument p

gdb

Rubriques

- Premiers pas
- Suivre l'exécution
- Affichage des variables

Premiers pas

Présentation de gdb

- Le programme central permettant de débuguer s'appelle *gdb*, c'est un débugueur en mode commande sans interface graphique.
- Il existe plusieurs débugueurs graphiques qui s'appuient sur gdb sans fournir toutes ses fonctionnalités.
- Il est donc intéressant de connaître les possibilités de gdb pour savoir se passer de débugueur graphique dans certains cas.
- Il est des fonctions que le débugueur *gdb* possède et qui sont rares dans les débugueurs graphiques :
  - sortie des traces sur fichier
  - interface avec le shell
  - historique avec rappel des commandes et complétion
- gdb peut être chargé sur un équipement embarqué en phase de test, là où les débugueur graphiques sont très gourmands en ressource
- gdb supporte les langages suivants : C, C++, D, Go, ObjectiveC, Fortran, Pascal, Mosula2, Ada

gdb : lancement

- il est nécessaire d'avoir compilé son programme avec l'option -g afin de disposer des tables des symboles
- On lance gdb suivi du nom du programme pour entrer dans le mode dgb interactif

```
gdb program
```

- On lance l'exécution du programme par `run` ou `start` qui lance uniquement la première ligne.
- Si on a pris soin de déclarer à l'extérieur du programme

```
export EDITOR=mousepad
```

- on pourra appeler l'éditeur pour modifier le code à l'intérieur du débugueur par la commande

```
edit fonction
edit numero de ligne
```

- dans le Debugueur la fonction `make` relance le makefile sans qu'on ait besoin de sortir de l'éditeur

Programme avec arguments

- on peut lancer un programme avec des arguments de deux façons

```
gdb --args programme argument1 argument2
run
ou
gdb programme
run argument1 argument2
```

- On peut voir les arguments à tous moment par la commande *show args*, on peut même les changer par la commande *set args valeur1 valeur2*
- *run* permet de lancer l'exécution mais généralement on pose des points d'arrêt avant
- pour arrêter un programme on tape *CTRL C* on peut alors le relancer par run
- une fois dans l'invite on peut entrer :
  - **help** affiche une aide de premier niveau
  - **help data** affiche une aide de deuxième niveau sur la rubrique choisie ici data
  - **help command** affiche l'aide sur une commande
- Pour quitter gdb on tape *quit* ou plus simplement *q*

Quelques astuces

- **! commande** ou **shell commande** lancent une commande shell
- On peut choisir le shell qui s'exécute en le définissant dans la variable *SHELL*
- Si vous souhaitez refaire un make dans gdb entrer seulement *make all* qui est équivalent à *!make all*
- On peut demander à gdb de loguer les sorties vers un fichier texte par

```
set logging on
set logging off
```

- Par défaut le fichier texte est gdb.txt mais on peut le spécifier via

```
set logging file fichier
```

- Chaque commande peut être entrée sous sa forme longue mais également sous sa forme courte. Par exemple *s* est *step*. *se* est ambiguë car peut être *search section* ou *set*.
- si on tape la touche *entrée* la dernière commande est généralement répétée
- Enfin la touche *TAB* donne toutes les complétions possibles, notamment pour les arguments des commandes

Gestion des variables d'environnement

- *path répertoire :* ajoute pour *gdb* le répertoire au PATH du shell qu'il connaît déjà
- *show paths :* affiche la variable d'environnement UNIX $PATH
- *show environment [variable] :* montre la variable d'environnement donnée ou toutes si elle n'est pas mentionnée
- *set environment variable=value :* change la variable d'environnement mais uniquement dans le débugueur
- de même on peut changer de répertoire dans *gdb* par *cd répertoire* ou afficher le répertoire courant par *pwd*
- *unset environment variable :* supprime la variable

Afficher le source d'un programme

- **list num:** affiche le programme source, précédé des numéros de ligne, centré sur le numéro de ligne donné en argument
- **list fonction:** affiche le programme source centré sur le début d'une fonction
- **list :** affiche les lignes suivantes
- **list - :** affiche les lignes précédentes
- **list debut,fin :** affiche les lignes de *début* à *fin*
- **list debut, :** affiche les lignes en commençant à *début*
- **list ,fin :** affiche les lignes finissant par *fin*
- on peut demander d'afficher un nombre de lignes plus important que celui proposé par la commande `set listsize valeur`
- Les fonctions `search regexp` et `reverse-search regexp` permettent de rechercher dans le sources la première occurrence de l'expression régulière passée en argument :

```
search mafonction
search mafonction[0-9]
```

Suivre l'exécution

Breakpoint - watchpoints - catchpoints

- un *breakpoint* ou point d'arrêt est posé sur une adresse pour ordonner au débugueur de s'arrêter lorsque il atteint cette adresse
  - un *breakpoint* peut être unitaire c'est à dire être supprimé après un seul passage (commande tbreak à la place de break)
  - il peut être régit par une expression régulière (syntaxe rbreak)
- un *watchpoint* ou point d'observation arrête le programme lorsqu'une valeur est modifiée ou juste accédée en lecture. Il n'est pas placé sur une ligne particulière
- un *catchpoint* permet d'arrêter un programme lorsqu'une exception C++ se produit. C'est une possibilité donnée pour mémoire.
- Chacun de ces points d'arrêt dispose d'un numéro qui permet de l'identifier

gdb : point d'arrêt

- On place des points d'arrêt par les commandes
  - **break** (b) nom de fonction (fonctionne sans table des symboles)
  - **break TestClass::testfunc()** : point d'arrêt dans une méthode d'une classe
  - **break (b) numéro de ligne** de programme (nécessite la table des symboles)
- on peut ajouter if condition après un break (condition utilise les variables du programme). Ceci permet de s'arrêter à un point donné que si une condition est remplie :

```
b 12 if ( compteur == 2 )
```

- La condition peut être ajoutée sur un point d'arrêt existant par la syntaxe

```
condition numerobreak expression
```

- Pour supprimer une condition `condition numerobreak`
- *numerobreak* n'est pas la ligne du programme mais le numéro du break donné par la commande `info break` qui liste les points d'arrêt.
- on peut aussi demander à passer sur un point d'arrêt un certain nombre de fois avant de s'arrêter par

```
ignore numbreak compteur
```

Watchpoint

- **watch expression** : arrête le programme dès que l'expression donnée par la variable est accédé en écriture

```
watch i
```

- on peut non plus se référer à une variable mais à l'écriture d'une adresse mémoire par :

```
whatch -l 0xdeadbeef
```

- qui revient à écrire

```
whatch *0xdeadbeef
```

- Un watchpoint peut être non plus à l'écriture d'une valeur mais à son simple accès en lecture avec les mêmes options que pour watch

```
rwatch expression
```

- La syntaxe rwatch permet de poser un watchpoint sur lecture ou écriture d'une adresse
- `info watchpoints` donne la liste des watchpoints

Gestion des points d'arrêt

- **info break :** lister les points d'arrêt avec leur numéro d'ordre sans distinction des *watchpoints* et des *breakpoints*
- suppression de points d'arrêt :
  - **clear :** suppression le point d'arrêt courant
  - **clear fonction** ou **clear ligne :** suppression d'un point d'arrêt
  - **delete numerobreak :** suppression du point d'arrêt par son numéro d'ordre
- Activation/désactivation de points d'arrêt :
  - **enable numerobreakpoint :** activation du point d'arrêt par son numéro d'ordre
  - **enable numerobreakpoint once :** activation du point d'arrêt par son numéro d'ordre une seule fois ensuite passe inactif
  - **disable numerobreakpoint n :** activation du point d'arrêt et désactivation après n passages
  - **enable :** inverse de disable avec les mêmes arguments
- Sauvegarde de points d'arrêt :
  - **save breakpoints fichier.txt :** sauvegarde des breakpoints
  - **source fichier.txt :** lecture des breakpoints

Les commandes pas à pas

- **continue :** continue jusqu'au prochain break ou jusqu'à la fin du programme
- **step :** le programme avance d'une instruction, son abréviation est s. Si l'instruction suivante est une fonction *step* provoque l'entrée dans la fonction.
- **step n :** avance de n instructions ;
- **next [n]:** avance d'une instruction mais n'entre pas dans les fonctions
- **finish :** continue jusqu'à sortir de la fonction en cours
- **until [ligne]:** continue jusqu'à ce qu'on passe à l'instruction suivante ou à la ligne donnée, c'est utile pour sortir des boucles.
- **layout asm:** passe en mode assembleur
- **nexti et stepi :** ont la même signification que next et step au niveau assembleur.
- **skip fonction :** permet de se positionner juste après la fonction
- **skip file fichier:** permet de se positionner systématiquement après toutes les fonctions contenues dans le fichier.
- **delete/enable/disable skip :** détruit/désactive/active tous les ordres de skip
- **delete/enable/disable skip numéro :** détruit/désactive/active le skip numéro
- **info skip [numero] :** affiche les information sur les skips en cours

Signals

- gdb détecte l'arrivée des signaux UNIX, on peut préparer le le comportement de gdb pour chaque signal
- **info signals [numero]:** affiche le comportement de gdb pour chacun des signaux
- **handle signal MOT CLE:** indique le comportement à avoir sur chacun des signaux. MOT CLE peut être :
  - **nostop :** gdb ne stoppe pas le programme quand le signal est reçu
  - **stop :** gdb stoppe le programme quand le signal est reçu
  - **print :** le signal provoque l'affichage d'un message
  - **printstop :** le signal ne provoque pas l'affichage d'un message
  - **pass :** gdb laisse le programme voir le message-
  - **passstop :** gdb ne laisse pas le programme voir le message

Affichage des variables

Revenir en arrière

- Tous les OS n'acceptent pas de revenir en arrière, car cela nécessite une gestion sophistiquée des registres. Linux accepte en mode Intel 32 bits ou 64 bits, VMWARE également
- D'autre part cette option n'est disponible qu'avec la version gdb 7.0 minimum
- Les distributions n'incluent pas systématiquement l'option arrière jugée non stable. Il faut donc recompiler gdb dans certains cas.
- Les commandes step, stepi, next, nexti, finish et continue admettent le préfix *reverse-* pour rembobiner la séquence en sens inverse avec les arguments de compteur identique à la version non reverse

```
reverse-step 2
```

La pile d'appel

- Quand le debugueur est arrêté sur une instruction, il peut être intéressant de déterminer comment on est arrivé à l'instruction.
- Chaque fois qu'une procédure est appelée, toutes les informations d'appel sont stockées dans un bloc d'information appelé le *stack frame* ou cadre de pile. Chaque appel provoque la création d'un nouveau frame.
- la commande `frame` remonte dans la liste des frames emmagasinées et disponibles, la commande `select-frame` produit le même effet sans afficher d'informations à l'écran

```
frame  2
select-frame 2
```

- **frame 0** : permet de revenir au niveau actuel
- **backtrace :** ou bt sans argument donne la liste des frames appelées pour arriver au point courant
- **bt n :** montre les n premières instances de pile
- **bt -n :** montre les n dernières instances de pile
- **up [n] :** remonte de n occurrences dans la pile
- **down [n] :** descend dans la pile
- **where :** affiche le niveau actuel dans la pile

Modification du contexte

- **print variable=valeur :** change la variable C ou C++ en valeur et affiche cette valeur. Valeur peut être calculée :

```
print i=j+2
```

- **set var variable=valeur :** la même chose sans affichage de la valeur.

```
set var i=2
set {int}0x83040 = 4
```

- la variable $pc est l'adresse d'exécution ou compteur de registre on peut l'altérer comme une variable, tout comme $sp qui est le pointeur de pile.
- On préfère cependant utilise la commande `jump` vers une adresse qui

provoque un continue à partir de l'adresse donnée en sautant les instructions entre le $pc et la ligne de code demandée.

```
break 32
jump 32
```

print variable

- La commande la plus basique pour afficher une variable est la commande *print* ou son abréviation *p*

```
p i
$1 = 1
```

- l'affichage concerne la frame en cours, ce qui fait qu'en cas de remontée dans la frame, on visualise la valeur de la variable dans la frame.
- la variable doit être accessible au moment où on demande son affichage
- les variables statiques peuvent être affichées sous la forme `fichier::variable` ou `fonction::variable`

```
print 'main.c::i'
```

- le symbole :: peut être utilisé pour donner la référence d'une fonction, c'est à dire différencier une variable d'une fonction à une variable de même nom. En cas de conflit de notation, la notation C++ prime.
- **print /f expression :** permet l'affichage d'une expression suivant le format f
- **f** peut être : x : impression hexadécimal, d : entier décimal, u : décimal signé, o: octal, a : format adresse, c: caractères, f : flottant, s : string ...
- sans expression, *print* affiche la dernière valeur affichée

Autres affichages

- La commande `display /fmt` affiche une variable après chaque instruction, ce qui est pratique pour suivre une ou plusieurs variables
- La commande *explore* permet d'afficher la valeur et le type d'une variable

```
explore valeur
```

- Affichage de zone mémoire : se fait par la commande x :

```
x [/nfu] adresse
```

- **n :** taille en unité à afficher
- **u :** b pour 1 octet, h pour 2 octets, w pour 4 octets, g 8 octets
- **f :** format x, d, u, o, t, a, c, f, s ou i pour voir l'instruction assembleur

```
x/3uh 0x54320
```

Printf dynamique

- *dprintf* combine un point d'arrêt à l'affichage de données le résultat d'un printf appels dans le programme, sans avoir à recompiler.
- La syntaxe est `dprintf adresse,format, expression1,...`

```
dprintf 45,"i=%d,j=%d\n",i,j
```

- La commande `save breackpoints fichier` qui sauvegarde l'ensemble des points d'arrêt sauvegarde également les printf dynamiques.
- La fonction `set dprintf-style style` permet de stipuler la fonction d'impression suivant la valeur de style :
  - `gdb` : la fonction printf de gdb
  - `call` : une fonction du programme déterminée par `set dprintf-function`
  - `agent` : en cas compilation à distance. Dans ce cas, il faut que gdbserver accepte le dprintf distant(>2014)
- `set dprintf-function fonction` : donne la fonction d'impression si style = call
- `set dprintf-channel canal` : permet de déporter le debug sur un log par exemple, il suffit que le IO/Stream standard soit assigné à la variable `canal`

Break avec liste de commandes

- les watchdogs, les breakpoints et les catchpoints peuvent être assignés à des suites de commandes
- les commandes sont encadrées par les mots clés `commands` et `end`
- à l'intérieur on utilise généralement le mot clé `silent` qui indique au break de rester en mode silencieux on inclue un ordre continue, step ou next, si on souhaite que le point d'arrêt se comporte comme un point de modification de code et non plus comme un point d'arrêt

```
break 567
commands
silent
set x = y + 5
continue
end
```

- Pour détruire une liste de commandes on déclare une séquence vide entre `commands` et `end`

Cas avancés

Rubriques

- Debuguer un programme en exécution
- Debuguer via un core dump
- Debuguer à distance

Debuguer un programme en exécution

Contraintes

- Si on souhaite voir les sources du programme en mode debug, il faut que le programme en cours d'exécution ait été compilé en mode -g
- Attention : certaines configurations Linux interdisent de tracer les processus non fils aux utilisateurs non root
- Si on utilise gdb on peut passer en mode root pour lever cette interdiction. Par contre lancer Eclipse ou ddd en mode root n'est pas toujours possible.
- Pour autoriser les utilisateurs non root à tracer des programmes on peut également procéder comme suit :
  - Pour un usage permanent éditer le fichier `/etc/sysctl.d/10-ptrace.conf` et positionner la ligne `kernel.yama.ptrace_scope = 1` à valeur 0
  - Pour un usage ponctuel exécuter `echo 0 | sudo tee /proc/sys/kernel/yama/ptrace_scope`
- on récupère le PID du programme tournant par

```
ps awx | grep programme
```

Sous gdb ou ddd

- on peut attraper le programme tournant de deux façons :

```
gdb -p 12122
```

ou

```
gdb
attach 12122
```

- une fois le processus attaché, l'exécution est stoppée et on peut prendre la main dans le débugueur.
- une commande `where` permet de connaître les frames et on exécute une commande up autant de fois que nécessaire pour se retrouver dans une fonction dont on possède les sources.
- une fois qu'on lancera la commande detach (ou qu'on quittera le Debuguer), l'exécution du processus reprendra normalement.

Debuguer via un core dump

Debug après plantage : core dump

- Sous Linux on peut facilement mettre en place un mécanisme permettant de débuguer un programme en analysant après coup un fichier trace appelé *core*.
- C'est via le débugueur que ce fichier va être rejoué.
- Il faut cependant s'assurer que ce fichier est généré en cas de plantage, ce qui n'est pas forcément le cas sur toutes les plate-formes.
- Il faut également connaître précisément la version du logiciel qui a planté et être en mesure de disposer des mêmes sources. Ceci est rendu possible par l'utilisation de gestionnaires de versions dont le plus répandu est aujourd'hui *git*.
- Il faut donc savoir précisément actionner la génération d'un fichier core (core dump)
- Il est conseillé de renseigner la version git à l'intérieur de l'exécutable afin de savoir la retrouver via l'exploration d'une variable par exemple.
- En outre, il faut garder à l'esprit que beaucoup de distributions génèrent ce fichier core mais laissent le programme Apport l'intercepter, en demandant l'autorisation à l'utilisateur d'envoyer le core au développeur. Il nous faut donc savoir désactiver ce mécanisme

Tuning de génération du core

- Vérification qu'un fichier core n'est pas intercepté

```
cat /proc/sys/kernel/core_pattern
```

- par exemple par `|/usr/share/apport/apport %p %s %c`
- que le fichier /etc/security/limits.conf contienne * soft core unlimited et * hard core unlimited
- Si on veut que la modification soit temporaire, on exécute

```
- en root : echo /tmp/core.prog_%e.sig_%s.proc_%p>/proc/sys/kernel/core_pattern
- en utilisateur : ulimit -c unlimited
```

- Si on veut qu'elle soit permanente :
  - ulimit se règle sous Debian en mettant dans le fichier `/etc/security/limits.conf` : `* soft core unlimited`
  - ajouter `kernel.core_pattern = /tmp/core.prog_%e.sig_%s.proc_%p` dans un fichier `/etc/sysctl.d/50-coredump.conf`
  - supprimer le programme apport (apt-get remove apport)
- On peut vérifier que le répertoire /tmp contient bien un fichier core après lancement de : `sleep 100 & kill -SIGSEGV %%` (%% est une variable bash représentant le PID de la dernière tâche suspendue)

Technique de débugage du core

- On s'assure que le programme qui a produit le core était bien compilé en mode débug
- Si on utilise gdb on lance :

```
gdb programme fichiercore
```

- le programme doit être généré avec les options de debug
- le fichier core doit être de la forme `/tmp/core.programm_bash.sig_11.proc_29255`
- Le débugueur montre l'instruction qui a provoqué le plantage
- on remonte via autant de commandes *up* que nécessaire jusqu'à trouver son code
- on inspecte sous le débugueur les variables afin de comprendre pourquoi l'instruction a provoqué le plantage

Debuguer à distance

Debug croisé

- On appelle debug à distance la possibilité de débuguer un logiciel tournant sur un équipement distant depuis un autre équipement.
- Ce cas est très utile en embarqué où les machines cibles sont pourvues de peu d'espace de stockage et munis de processeurs de petite capacité demandant des compilations croisées. C'est pourquoi cette technique s'appelle également debug croisé.
- *gdb* permet le débug croisé via un port série ou via TCP/IP
- *gdb* permet de débuguer un code distant sur un processeur différent du processeur de la machine possédant le débugueur. Il gère même les cas de byte ordering différent.
- Sur la carte distante, on rencontre deux cas :
  - un serveur gdb est installé car on est sur une machine unixlike, on communique de débugueur client à débugueur serveur
  - un serveur n'est pas installé dans ce cas on doit implémenter sur le logiciel de la carte le protocole stub de communication de gdb.
- Nous nous bornerons aux cas d'utilisation avec le protocole gdbserver en mode TCP/IP

Installation sur l'équipement distant

- nous supposons que ce serveur a l'adresse 192.168.0.237
- sur le serveur distant installer gdbserver
  - sur architecture debian : `apt-get install gdbserver`
  - sur embarqué busybox disponible dans les config
- La machine distante et la machine host doivent disposer d'un binaire construit à partir des mêmes sources avec les librairies idoines
- sur la machine distante : le binaire compilé sans option -g, stripé ou non stripé, les sources ne sont pas nécessaires
- Sur le serveur distant on lance le serveur gdbserver avec la syntaxe suivante :

```
gdbserver 192.168.0.32:2345 programme arguments
```

- 192.168.0.32 est l'adresse IP de la machine Host
- 2345 est le port TCP que vous choisissez
- programme est le programme que vous débuguez juste compilé

Mise en route sur l'équipement host

- Nous supposons que cet équipement a l'adresse 192.168.0.32
- Nous compilons le programme avec l'option -g
- Sur le poste de travail du débugueur on lance

```
gdb nom du programme
```

- On peut ainsi débuguer à distance après avoir informé le débugueur par la première commande :

```
target remote 192.168.0.237:2345
```

- le programme étant lancé on fait généralement un **break main** ou

un **continue**

Remarques sur le debug croisé

- Depuis fin 2013 de nouveaux protocoles ont fait leur apparition d'abord sur le client gdb 7.5 puis sur le serveur gdb.
- Ceci rend compliqué le protocole entre un équipement d'après 2014 et un équipement d'avant 2014
- Plus d'information sur https://sourceware.org/gdb/wiki/LinkerInterface
- En embarqué si vous utilisez busybox, les dernières versions permettent de choisir des versions compatibles en fonction de votre host.
- Utilisation avec ddd :
  - soit mettre `put target remote 192.168.0.237:3456` dans un fichier et utiliser l'option `--command fichier` pour l'utiliser
  - soit mettre `put target remote 92.168.0.237:3456` dans le fichier .gdbinit dans votre répertoire d'accueil
  - soit lancer

```
ddd --eval-command="target remote 192.168.0.237:1234" main
```