Université de Bretagne Occidentale

Systèmes d‘Exploitation pour l‘Embarqué

Cours 3 : Construction d’un Linux embarqué

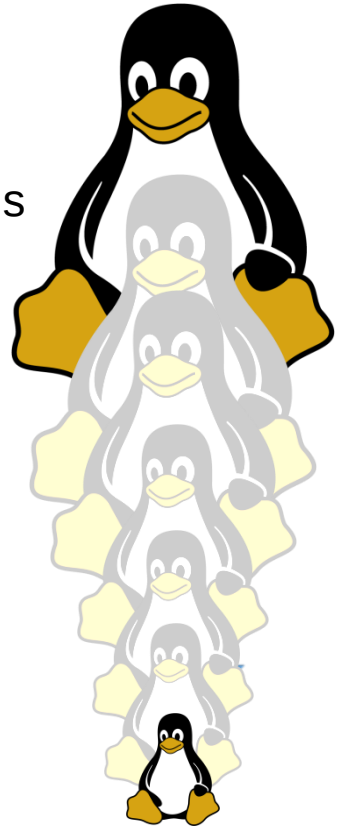
https://tinyurl.com/m2lsesee

Hai Nam TRAN
UBO – M2 LSE

Le CM est inspiré du cours précédemment dispensé par Jalil Boukhobza

Réduire l'empreinte mémoire !

- **Mémoire (rapide) est très couteuse**
- **3 façons de réduire l'empreinte mémoire:**
  1. **Optimiser le noyau**
     - Enlever le code dont on n'a pas besoin
     - Optimiser la compilation -O
     - Enlever le swap
     - Voir le « *Linux tiny kernel project* »: plusieurs optimisations sous forme de patch.
  2. **Optimiser l'espace de l'applicatif**
     - Optimiser son **code**
     - Optimiser son utilisation de la **librairie**
       - Optimiser les librairies partagées dans une application après développement
       - Utiliser des librairies réduites (uClibc, diet libc, etc)
     - Utiliser des **applications optimisées**:
       - *BusyBox*
       - *TinyLogin*
       - Serveur web *BOA*, *mini_httpd*, *GoAhead*
  3. **Compresser le système de fichiers**
     - Certains sont compressés: JFFS2, CRAMFS



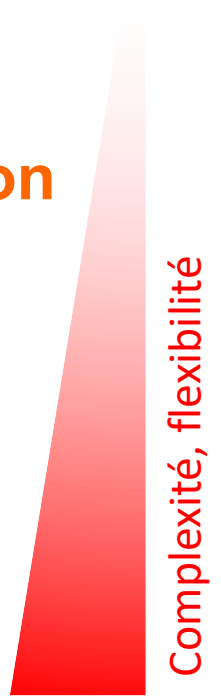
Réduire l'empreinte mémoire !

	Capacity	Latency	Cost/GB
Register	1000s of bits	20 ps	$$$$
SRAM	~10 KB-10 MB	1-10 ns	~$1000
DRAM	~10 GB	80 ns	~$10
Flash	~100 GB	100 us	~$1
Hard disk	~1 TB	10 ms	~$0.10

Linux pour l'embarqué

1. **Solutions sur étagère**

2. **Construction d'une distribution (compilation croisée)**
   1. **Utiliser un(e) chaîne/framework déjà produit(e)**
   2. **Construire son environnement de compilation croisée**



- **Plusieurs solutions**
  - **Solutions sur étagère**
    - Dégrossir une distribution pour PC Fedora, Debian
    - **TP3** : nous utiliserons l'image du Debian 10 (Buster) à partir du site beagleboard.org/latest-images
      - Espace mémoire important
      - All-in-one : OS, Integrated Development Environment (Cloud 9 IDE), ssh, ftp, ...
  - **Construction d'une distribution (compilation croisée)**
    - Utiliser un(e) chaîne/framework déjà produit(e)
      - **TP3** : Buildroot
    - ~~Construire son environnement de compilation croisée~~

Framework pour Linux embarqué

- **Donne des possibilités proches du « *Do It Yourself* »**
  - **Outils de construction de l'environnement et du système: compilateur croisé, outils d'intégration, bibliothèque C, version du noyau, exécutables, etc**
  - **Résolutions des dépendances: compilées automatiquement**
  - **Mises à jour: intégration des dernières versions dans la chaîne**
- **Buildroot https://buildroot.org/ : ensemble de *makefile* et de patches permettant de générer**
  - **Une chaîne de compilation croisée**
  - **Un système de fichiers racine**
  - **Une image de noyau Linux**

Compilation croisée

- **Tout faire à la main → fastidieux et souvent inutile**
- **Utiliser un outil permettant d'aider à faire cela:**
  - **Crosstool-NG:**
    - Outil de configuration simple (menuconfig)
    - Construction d'une chaîne de compilation croisée
- **Différentes étapes:**
  - **Décider de la cible**
  - **Décider de la version de noyaux/gcc/glibc/binutils**
  - **Patches si nécessaire**
  - **Définir les préfixes (PREFIX, KERNEL_SOURCE_DIR, NATIVE, etc.)**
  - **Compiler binutils**
  - **Compiler un gcc minimal (pour la cible)**
  - **Construire la glibc (pour la cible)**
  - **Recompiler gcc complet (pour la cible)**

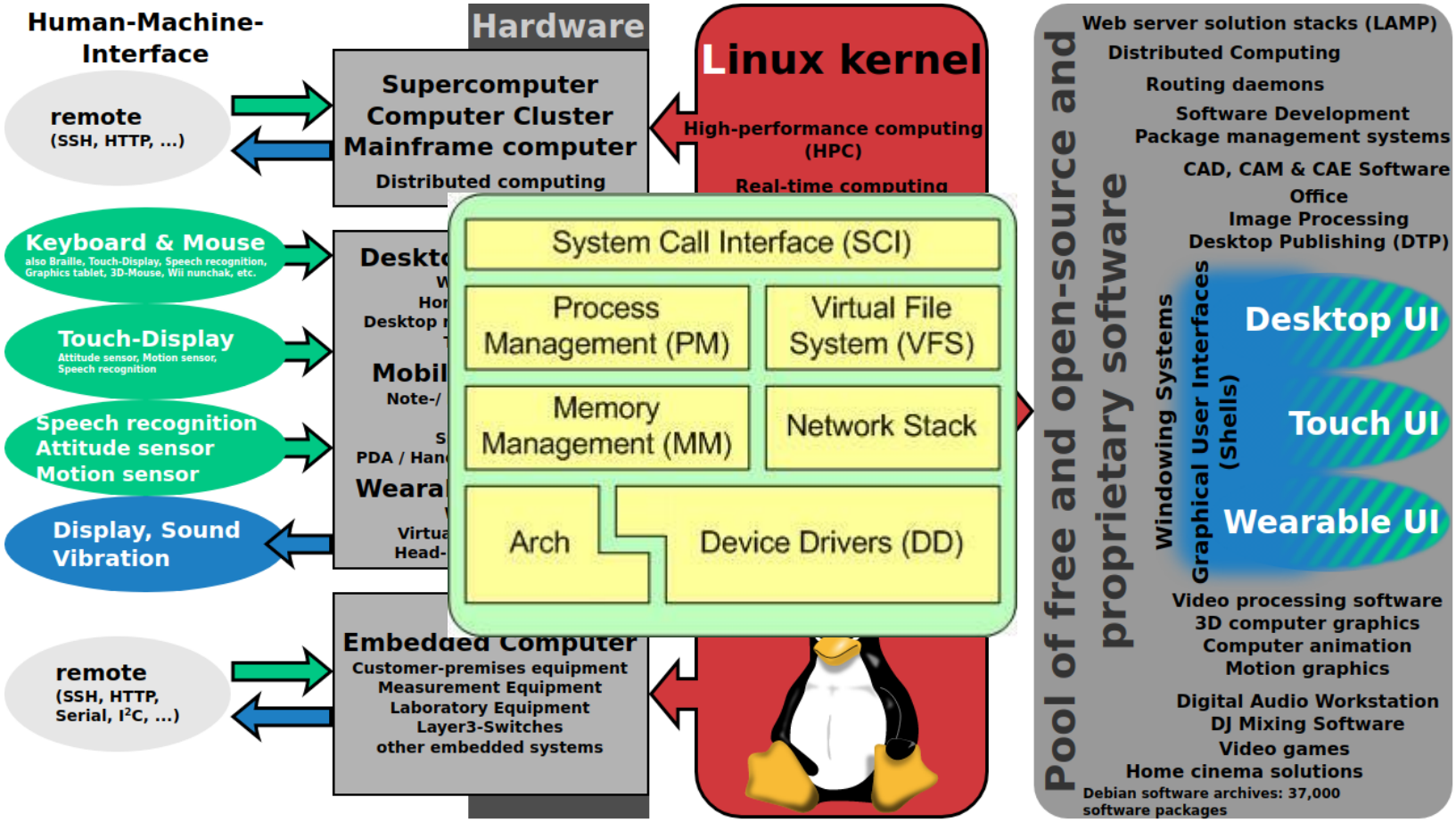
Binutils

- **Un ensemble d'outils permettant de générer et de manipuler des binaires pour une architecture donnée (CPU)**
  - **as, l'assembleur qui génère le code binaire d'un code source en assembleur**
  - **ld, l'éditeur des liens**
  - **ar, ranlib, génère l'archive .a, utilisée pour les librairies**
  - **objdump, readelf, size, nm, strings, pour inspecter les binaires**
  - **…**
- **http://www.gnu.org/software/binutils/**
- **GPL licence**

Etapes clés

1. Construction du noyau
2. Construction du système de fichiers racine
3. Construction des applications/commandes
4. Configuration du démarrage

1. Construction du noyau



1. Construction du noyau

- **La sélection du noyau**
  - **http://www.kernel.org/ : dépôt principal et officiel des (Vanilla ou mainstream) noyaux Linux.**
    - Pour l'embarqué … pas toujours été approprié! (question d'architecture), mais on utilise de plus en plus le dépôt principal.
    - Quelle version ?: généralement prendre la dernière (stable) !
      - Mais ce n'est pas évident ! Cela dépend des changements qu'il y a eu
      - Les patchs (fonctionnalités additionnelles)
      - Compatibilité avec d'autres outils …

Architecture de processeur	Ou télécharger le noyau	Comment ?
x86	**http://www.kernel.org/**	**ftp, http, rsync**
ARM	http://www.arm.linux.org.uk/developer/ … plus très à jour (-> kernel.org)	ftp, rsync
PowerPC	http://penguinppc.org/	ftp, http, rsync, bitkeeper
MIPS	http://www.linux-mips.org/	cvs
SuperH	http://www.linux-sh.org/shwiki/FrontPage	cvs
M68k	http://www.linux-m68k.org/ … pas très à jour	ftp, http

1. Construction du noyau

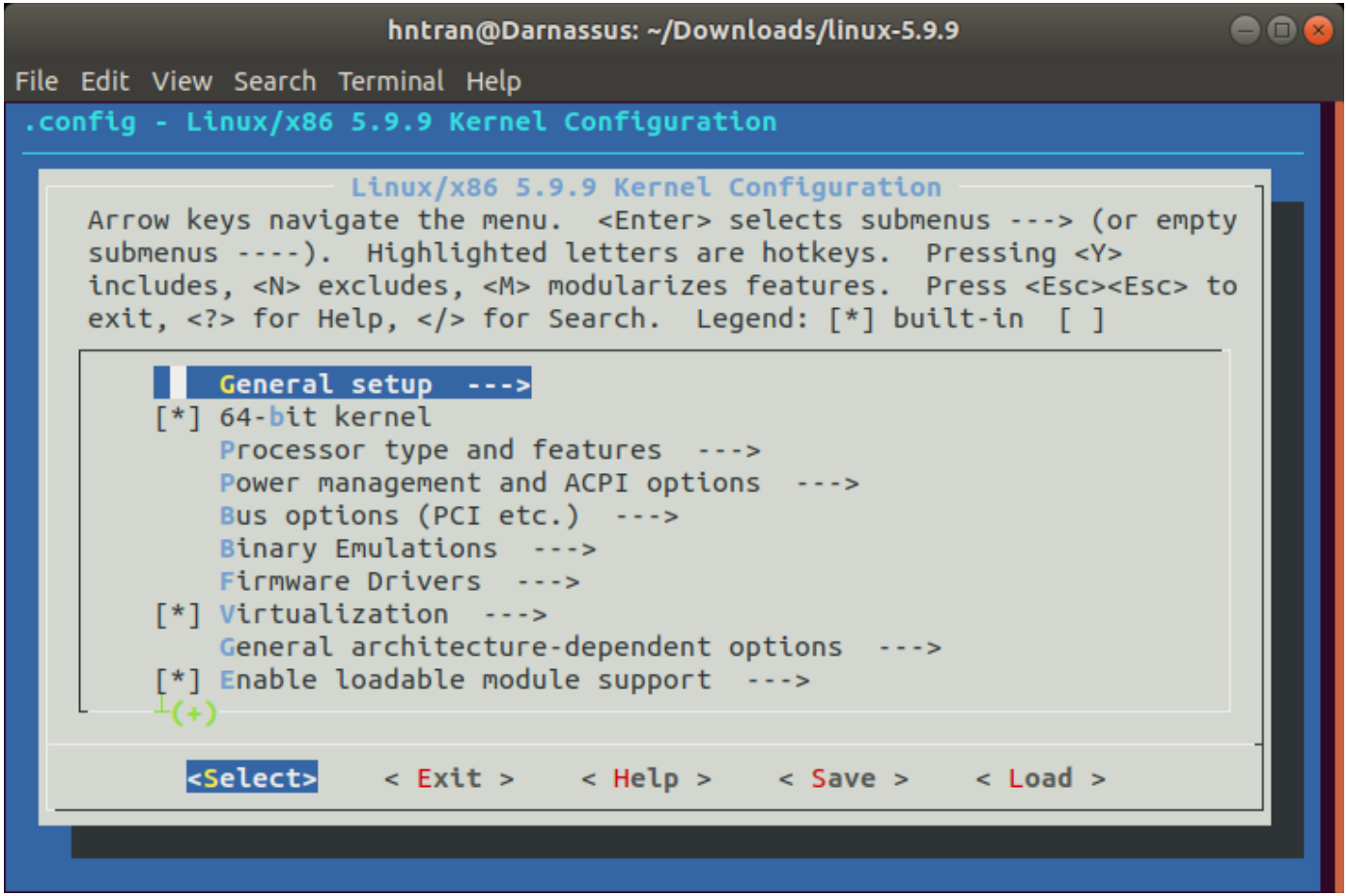
- **Configuration du noyau**
  - **Permet de sélectionner les options que l'on veut inclure dans le noyau (ou dans les modules)**
    - Dépend de l'architecture
  - **À la fin de la configuration, sont générés:**
    - Un fichier .config
    - Un certain nombre de liens symboliques
    - Un certain nombre de fichiers entêtes
  - **Exemple d'options paramétrables :**
    - Options de réseau
    - Support de modules chargeables
    - Dispositif de technologie mémoire
    - ...

1. Construction du noyau

- **Configuration du noyau**
  - **Méthode de configuration**
    - **`make config`**: interface en ligne de commande avec demande pour chaque option (valeur par défaut si fichier .config existant)
    - **`make oldconfig`**: prend en compte un *.config* existant et ne demande que les options non choisies dans l'ancien fichier.
    - **`make menuconfig`**: affiche un menu basé sur un curseur (lib curse) en utilisant comme valeur par défaut un *.config* existant.
    - **`make xconfig`**: affiche un menu (Xwindows) en utilisant comme valeur par défaut un *.config* existant.
    - **`make gconfig`**: autre menu (pareil que *xconfig* mais utilisant d'autres librairies graphiques)
    - Il est possible de sauvegarder plusieurs configurations
    - Attention: certaines libraires (QT, GTK+, etc.) sont obligatoires pour lancer certaines des configuration ci-dessus

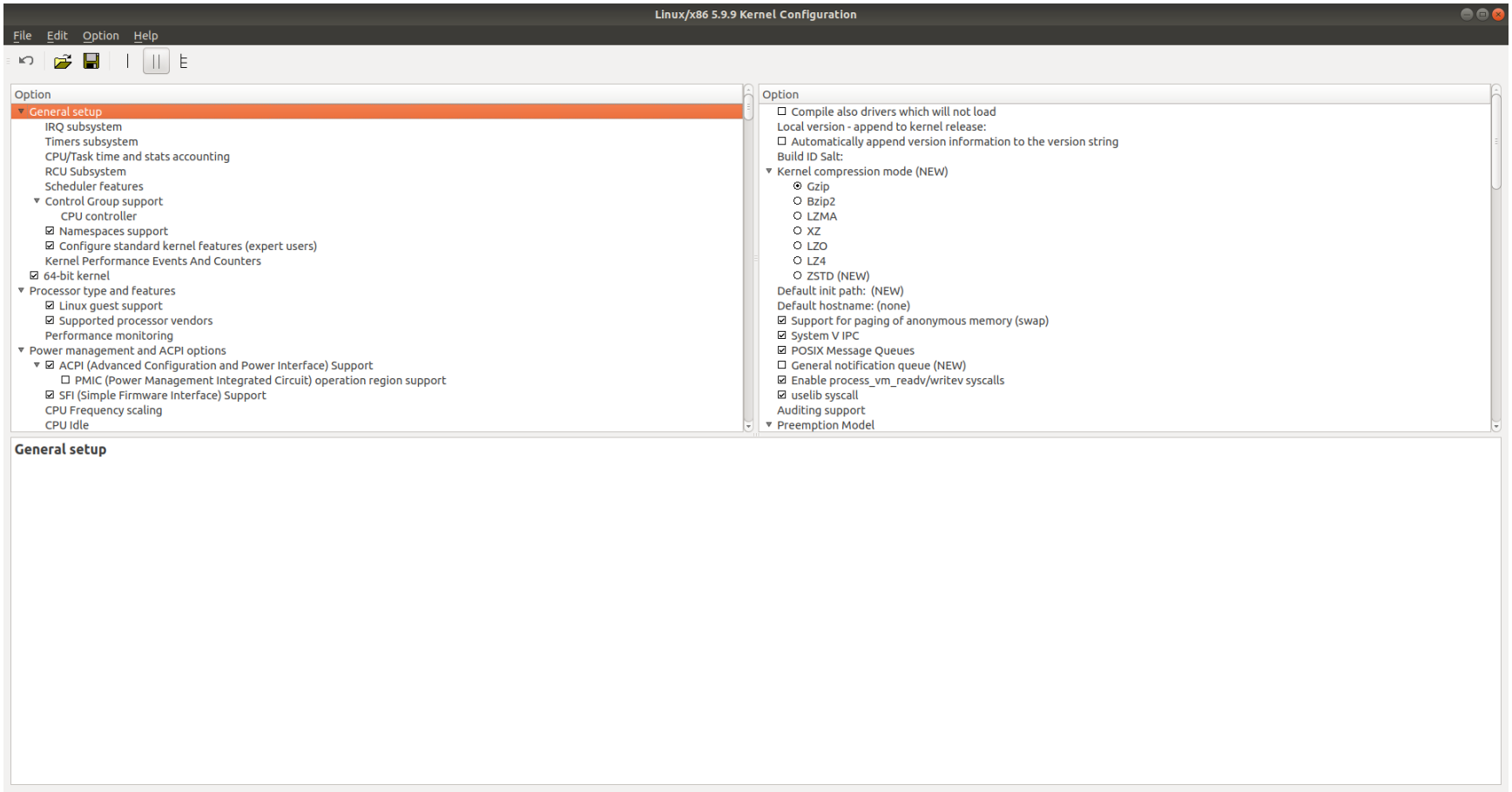
1. Construction du noyau

- **Configuration du noyau**
  - **Méthode de configuration**



1. Construction du noyau

- **Configuration du noyau**
  - **Méthode de configuration**



1. Construction du noyau

- **Compilation du noyau**
  - **Compilation du noyau avec :**

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gcc zImage
```

    - **ARCH=** : architecture cible sur laquelle tournera le noyau (dans l'exemple: arm).
    - **CROSS_COMPILE=**: besoin pour compiler le source. Utilisé pour la construction de nom d'outil utilisé pour la construction des noyaux (ex: nom du compilateur c est arm-linux-gcc)
    - zImage : compressée avec gzip
    - Si vmlinux : image non compressée.

1. Construction du noyau

- **Construction des modules**
  - **Une fois l'image du noyau proprement créée, on peut construire les modules :**

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gcc modules
```

  - **… Il faut ensuite les installer**

```
$ make modules_install
```

Etapes clés

1. Construction du noyau
2. Construction du système de fichiers racine
3. Construction des applications/commandes
4. Configuration du démarrage

2. Construction du système de fichiers racine

- **Quels répertoires y mettre ?**
  - **Plusieurs ne servent que dans le cas d'une utilisation multi-utilisateurs**
  - **Rappel (voir *File Hierarchy Standard* du TP1)**

Répertoires	Contenu
bin	Binaires de commandes **essentielles**
boot	Fichiers statiques utilisés par le chargeur d'amorçage
dev	Périphériques et fichiers spéciaux
etc	Fichiers de configuration système incluant les fichiers de démarrage (initialisation)
home	Répertoire de travail des utilisateurs en plus d'entrées pour des services de type FTP.
lib	Librairies essentielles et modules
mnt	Point de montage pour les systèmes de fichiers temporaires
opt	Autre logiciels
proc	Système de fichiers virtuel pour les information du noyau et des processus
root	Répertoire de travail du root
sbin	Binaires des commandes **essentielles** pour l'administration système
tmp	Fichiers temporaires
usr	Hiérarchie secondaire contenant la plupart des applications et documents partagés par tous les utilisateurs. (serveur X)
var	Données dynamiques stockées par les applications

2. Construction du système de fichiers racine

- **Les répertoires implémentant des fonctionnalités multiutilisateurs peuvent être omis:**
  - **`/home`, `/mnt`, `/opt` et (`/root` → peut être inclus comme unique compte)**
- **Quant au `/boot` …**
  - **Le chargeur d'amorçage peut-il extraire les images du noyau avant que le noyau ne soit démarré ?**
- **Créer certaines sous hiérarchies, ex: `/usr`, `/var`**
  - **Bien réfléchir à ce qui est utile pour votre application embarquée et à ce qu'il l'est moins**

2. Construction du système de fichiers racine

- **Les librairies**
  - **glibc : à l'origine très gourmande en mémoire … mais beaucoup d'efforts a été fait pour y remédier**
  - **uClibc : http://www.uclibc.org/ → uClibc-ng https://uclibc-ng.org/**
    - vient du projet uClinux (linux sur des processeurs sans MMU).
    - Devenu un projet en soit: supporte les architectures avec ou sans MMU + support de plusieurs architectures
    - N'est pas basé sur GNU libc, ne suit pas les mêmes normes mais si ça compile sous Glibc, ça compilera généralement sous uClibC → C89, C99 et SUSv3 (standards)
  - **diet libc : http://www.fefe.de/dietlibc/**
    - Développé de 0 (ne découle pas d'un projet antérieur)
    - Ne supporte pas beaucoup d'architectures
    - Prévue pour être utilisée comme une librairie statique (contrairement à uClibc)

2. Construction du système de fichiers racine

- **Les librairies**
  - **Installation**
  - **Configuration, exemple uClibc:**
    - Options relatives à l'architecture cible
    - Configuration de librairie, options générales
    - Support réseau
    - Support chaînes de caractères
    - Options d'installation
    - ….
  - **Modification de l'environnement pour la prise en compte de la nouvelle librairie.**
    - **Aide**: Lire la doc et faire un essai indépendant du reste de la construction de votre Linux embarqué.

2. Construction du système de fichiers racine

- **Les librairies**
  - **Comment installer ces librairies dans le système de fichiers racine pour être utilisées à la volée ?**
  - **On doit sélectionner quelles librairies inclure, exemple pour glibc :**

librairie	Contenu	Commentaires
ld	Éditeur de liens dynamique	Obligatoire
libc	Librairie des fonctions C principales	Obligatoires.
libcrypt	Fonctions de cryptographie	Nécessaire pour les applications d'authentification
libm	Fonctions mathématiques	Nécessaire pour les fonctions mathématiques
libmemusage	Fonctions pour profiler les piles et le tas	Rarement utilisées
libpthread	Fonctions de threads Posix 1003.1c	Programmation multithreadée
librt	Fonctions d'E/S asynchrones.	Peu utilisées
:	:	:

  - **Déterminer les librairies dont on a besoin: commande ldd / application**
  - **Copier celles dont on a besoin comme librairies + leurs liens symboliques**

2. Construction du système de fichiers racine

- **Les modules du noyau**
  - **Ne pas oublier de copier les modules résultants de l'étape de compilation du noyau dans le système de fichiers racine … au bon endroit … (doc FHS!)**
    - `/lib/modules` (pour une distribution classique)

2. Construction du système de fichiers racine

- **L'image du noyau**
  - **Si le noyau est démarré à partir du système de fichiers racine :**
    - On copie l'image du noyau dans le répertoire ***/boot*** du système de fichiers racine.
    - On peut aussi y copier la configuration *.config*

Etapes clés

1. Construction du noyau
2. Construction du système de fichiers racine
3. Construction des applications/commandes
4. Configuration du démarrage

3. Construction des applications/commandes

- **Beaucoup (trop) de commandes dans les systèmes actuels**
  - **Solution 1 : ne prendre que les commandes dont on a besoin**
  - **Solution 2 : dégrader/configurer un ensemble de commandes spécifiques à l'embarqué**
    - BusyBox : http://www.busybox.net/
      - **Plusieurs commandes parmi lesquelles**: ar, cat, chgrp, chmod, chown, chroot, cp, cpio, date, dd, df, dmesg, dos2unix, du, echo, env, expr, find, grep, gunzip, gzip, halt, id, ifconfig, init, insmod, kill, killall, ln, ls, lsmod, md5sum, mkdir, mknod, modprobe, more, mount, mv, ping, ps, pwd, reboot, renice, rm, rmdir, rmmod, route, rpm2cpio, sed, stty, swapon, sync, syslogd, tail, tar, telnet, tftp, touch, traceroute, umount, uname, uuencode, vi, wc, which, et whoami
      - **TP6** : `make busybox-menuconfig`
    - TinyLogin http://tinylogin.busybox.net/
    - Embutils http://www.fefe.de/embutils/

Etapes clés

1. Construction du noyau
2. Construction du système de fichiers racine
3. Construction des applications/commandes
4. Configuration du démarrage

4. Initialisation du système

- **Dernière action de l'initialisation du noyau: init**
  - **Finalise l'initialisation en lançant des applications clés**
- **Peut être même remplacée par une application en configurant l'amorçage**
  - **L'application sera la seule à être exécutée.**
- **http://www.linux.it/~rubini/docs/init/**
- **Plusieurs possibilités:**
  - **Init System V standard**
  - **Init de la BusyBox**
  - **Minit (miniature init) de Dietlibc http://www.fefe.de/minit**

Démarrage

- **TP3 :**
  - "Copiez les fichiers suivants se trouvant dans le répertoire "./output/images" de votre répertoire buildroot dans la partition "boot"
    - **MLO, u-boot.img, zImage, uEnv.txt, et am335x-boneblack.dtb**
- **uEnv.txt** : fichier de configuration de U-Boot
- **MLO :** Minimal bootloader
  - **First-stage bootloader : au démarrage ou à la réinitialisation du système, le bootloader ROM (RBL) démarre.**
    - Faire une initialisation minimale, puis de trouver et démarrer à partir d'un périphérique tel qu'une carte MMC ou un flash
    - Le RBL a la capacité de démarrer à partir d'une variété de périphériques tels que flash NOR, flash NAND, MMC, USB ou UART ou d'un fichier nommé **MLO**
  - **Second-stage bootloader : Le MLO lit la première partition de la carte SD, et charger un fichier appelé "u-boot"**
    - Init MMU + SDRAM
  - **Third-stage bootloader : U-boot**

Démarrage

- **u-boot.img**
  - **Chargeur de démarrage** - le programme qui va s'exécuter après le MLO qui va charger le système d'exploitation (zImage) et lui faire passer des structures comme le dtb
  - **https://www.denx.de/wiki/U-Boot**
- **zImage**
  - **Noyau de l'OS**
- **am335x-boneblack.dtb**
  - **dtb : Device Tree Blob**
  - Les détails du matériel : la taille et l'emplacement de la RAM physique, la vitesse d'horloge du processeur, ...
  - Plus d'information
    - https://elinux.org/images/f/f9/Petazzoni-device-tree-dummies_0.pdf

Université de Bretagne Occidentale

Noyau temps réel

Cours 3 : Construction d'un Linux embarqué

https://tinyurl.com/m2lsesee

Hai Nam TRAN
UBO – M2 LSE

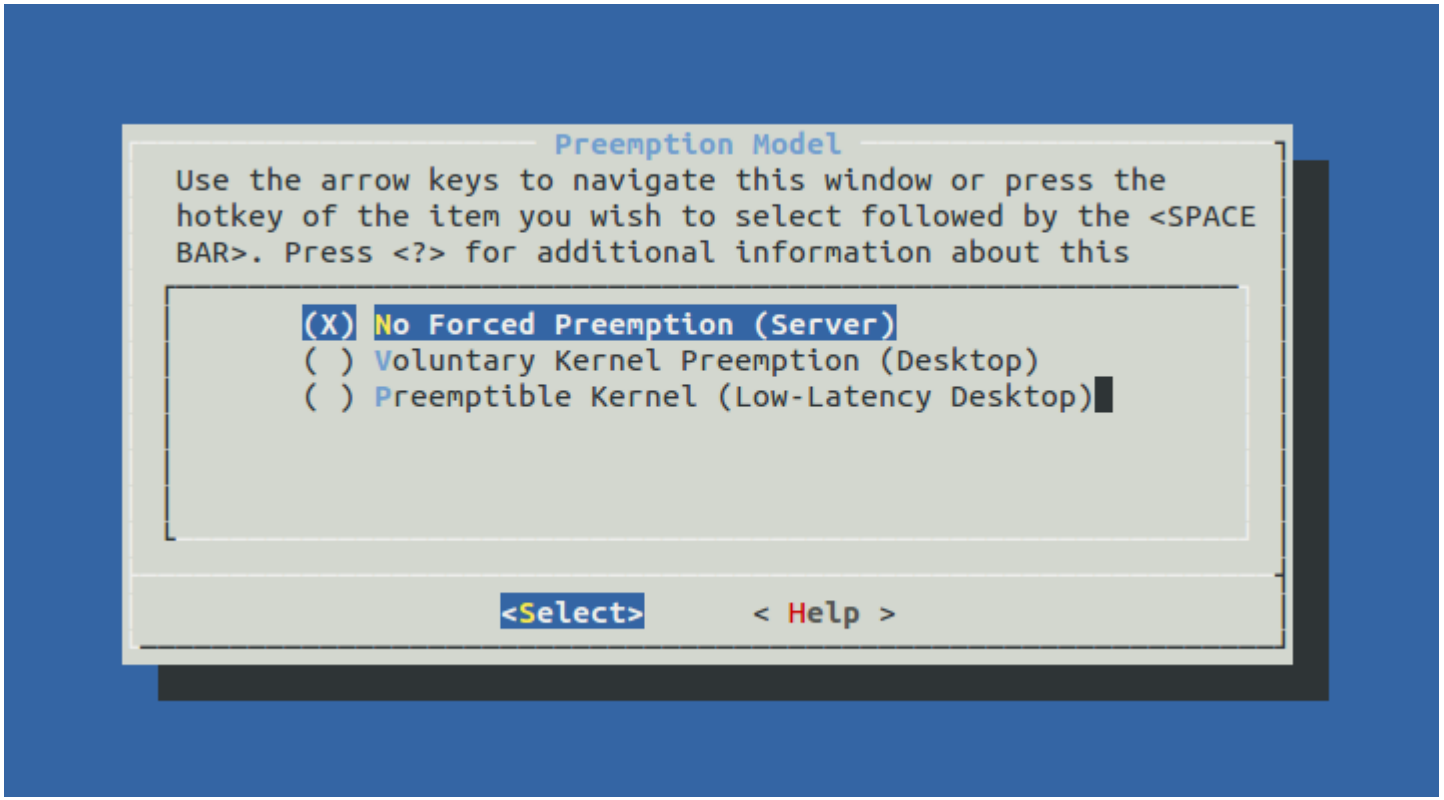
Le CM est inspiré du cours précédemment dispensé par Jalil Boukhobza

Linux et le temps réel

- **Implémentation temps réel souple/mou de Linux**
  - **Résolution du timer → quelques dizaine de µs, variabilité de quelques µs (dépend du matériel)**
  - **Commutation de processus: quelques µs**
  - **Préemptibilité du noyau : plus fine depuis la version 2.6**
  - **Gestion des interruptions : des délais importants**
- **Patch Linux-rt (Preempt-rt)**
  - **Linux-rt (aussi appelé Preempt-rt): patch permettant d'améliorer les aspects temps réel**
    - Meilleure préemptibilité du noyau
    - Gestion des interruptions par threads → avoir une priorité moins importante que les tâches temps réel (mais plus de latence sur le traitement d'interruptions)
    - https://www.kernel.org/pub/linux/kernel/projects/rt/

Linux et le temps réel

- **Preemption Model**
  - **~/buildroot$ make linux-menuconfig**
    - Kernel Features -->
    - Preemption Model



Linux et le temps réel

- **Preemption Model** *https://devarea.developpez.com/linux-comprendre-preemption-noyau/*
  - **No Forced Preemption**
    - La commutation de contexte (context switch) est faite seulement lors du retour du noyau
      - nous avons deux threads, l'un avec une priorité temps réel haute(50), et l'autre avec une priorité temps réel basse (30)
      - le thread haute priorité est mis en veille pour 3 secondes ;
      - le thread basse priorité appelle un code noyau durant 5 secondes ;
      - après 5 secondes, le thread basse priorité sort de l'espace noyau ;
      - le thread de haute priorité va se réveiller (avec 2 secondes de retard)
  - **Preemptible Kernel**
    - La commutation de contexte se fait dans les temps dans le noyau également
      - le thread haute priorité se réveiller après 3 secondes
  - **Voluntary Kernel Preemption**
    - Le système travaille comme une "No Forced Preemption". Mais si le développeur noyau écrit un code complexe, il est chargé de vérifier de temps en temps si une nouvelle planification est nécessaire avec la fonction might_resched()

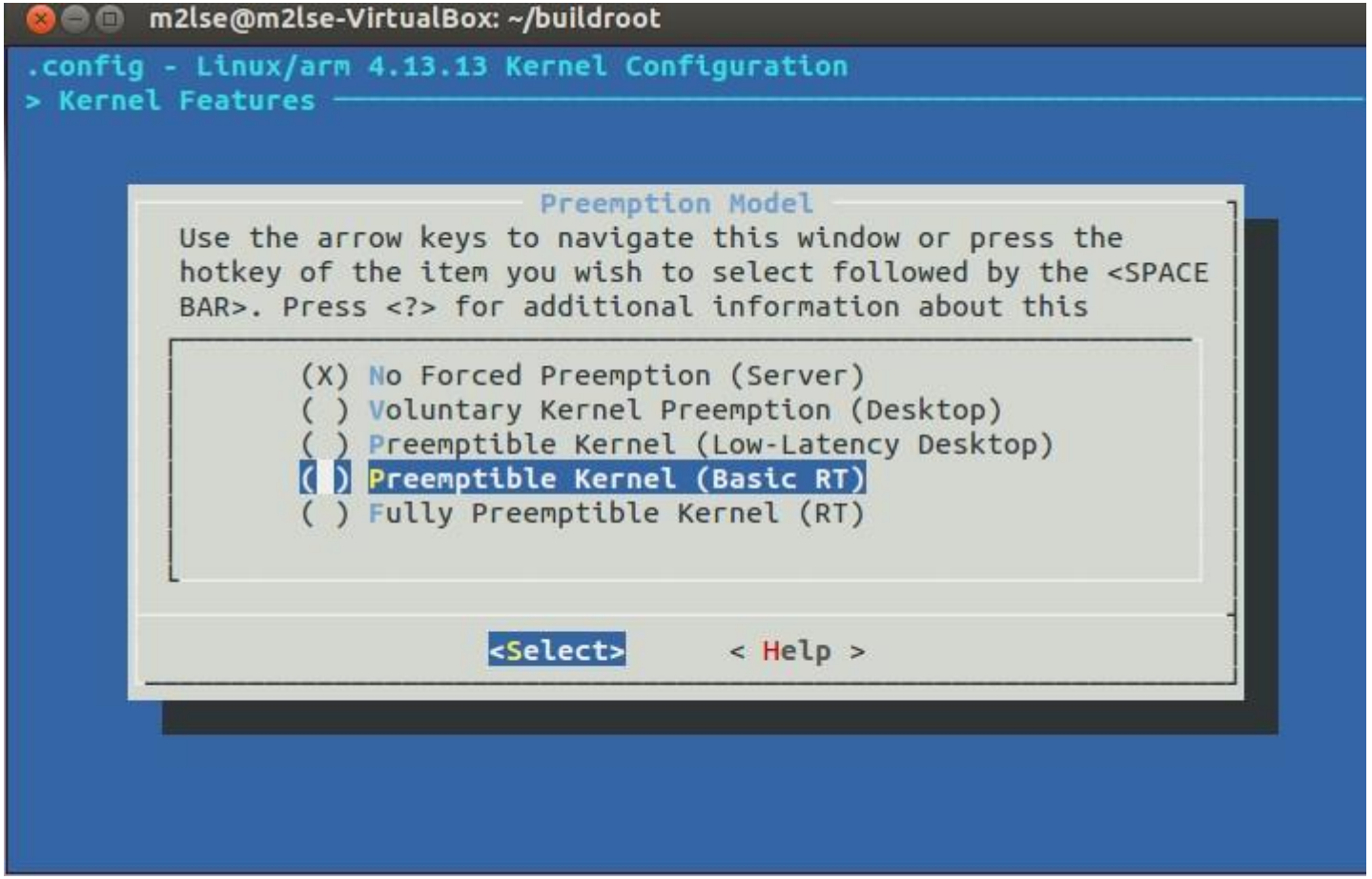
Linux et le temps réel

- **Appliquer le patch Linux-rt**
  - **La page où l'on peut télécharger le patch est la suivante**
    - https://www.kernel.org/pub/linux/kernel/projects/rt/
    - Il faut télécharger le patch correspondant à votre version du noyau
      - Par exemple TP3 : noyau version 4.13.13
      - `$ wget https://www.kernel.org/pub/linux/kernel/projects/rt/4.13/patch4.13.13-rt5.patch.gz`
  - **Une fois le fichier téléchargé, on va le décompresser.**
  - **Copier le patch dans le répertoire du source Linux**
    - `./buildroot/output/build/linux-4.13.13`
  - **Enter dans `./buildroot/output/build/linux-4.13.13/`**

et executér :

    - `$ patch -p1 < ./ patch-4.13.13-rt5.patch`
  - **Revenir sur le menu de configuration du noyau Linux. Vous devriez voir apparaître 2 options supplémentaires dans la configuration de la préemptibilité**

Linux et le temps réel



Linux et le temps réel

- **Preemption Model**

  https://devarea.developpez.com/linux-comprendre-preemption-noyau/

  - **Preemptible Kernel (Basic RT)**
    - Ce modèle est principalement utilisé pour tester et déboguer les mécanismes de substitution implémentés par le patch PREEMPT_RT.
  - **Fully Preemptible Kernel (RT)**
    - N'importe quel code peut en bloquer un autre.
      - Une tache peut bloquer le code d'une sous-routine d'interruption
    - Le patch change les choses suivantes
      - convertit les interruptions matérielles en threads avec une priorité RT de 50
      - convertit les interruptions logicielles en threads avec une priorité RT de 49
      - convertit tous les spinlocks en mutex
      - configure et utilise les timers haute résolution

- **Tester et évaluer les Preemption Model ?**
  - **Programmes du TP2 sur les 5 noyaux**
  - **Emulation avec QEMU**

Université de Bretagne Occidentale

BeagleBone Black : Getting Started

Cours 3 : Construction d'un Linux embarqué

https://tinyurl.com/m2lsesee

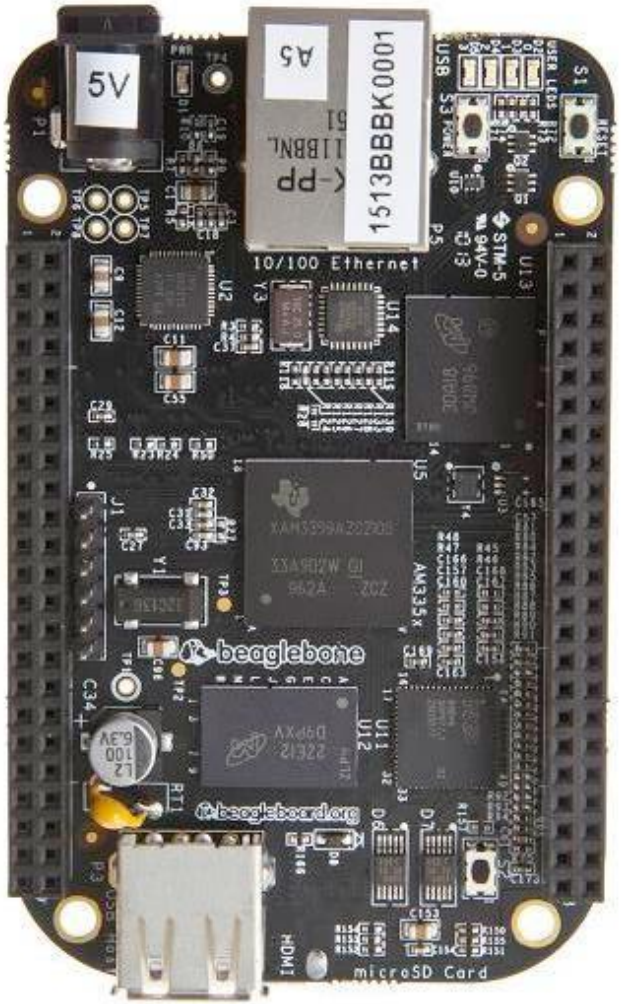
Hai Nam TRAN
UBO – M2 LSE

Le CM est inspiré du cours précédemment dispensé par Jalil Boukhobza

What is BeagleBone Black ?

- **BeagleBone Black is a low-cost, community-supported development platform for developers and hobbyists**
  - Processor: AM335x 1GHz ARM® Cortex-A8
    - 512MB DDR3 RAM
    - 4GB 8-bit eMMC on-board flash storage
    - 3D graphics accelerator
    - NEON floating-point accelerator
    - 2x PRU 32-bit microcontrollers
  - Software Compatibility
    - Debian, Android, Ubuntu, Cloud9 IDE
  - Connectivity
    - USB client for power & communications
    - USB host
    - Ethernet
    - HDMI
    - 2x 46 pin headers

What is BeagleBone Black ?



What we will do with this BBB ?

- **Develop some simple programs or execute the examples implemented in TP1 and 2**
  - **With a distribution of Debian**
  - **With a OS and Linux kernel that we build ourselves**
  - **Without OS (not possible to execute some if not all examples)**

Getting Started : Update board with latest software

- **Download the latest software image**
  - **beagleboard.org/latest-images**
  - **In this course, we will use the image**
    - **Buster** IoT (without graphical desktop) for BeagleBone and PocketBeagle via microSD card
    - The image you downloaded includes Cloud9, a web-based intergrated development environment (IDE)
- **Install SD card programming utility**
  - **Download and install balenaEtcher (https://www.balena.io/etcher/)**
- **Connect SD card to your computer**
  - **You will need a microSD card**
  - **Use Etcher to write the image to your card**
- **Boot your board off of the SD card**
  - **Insert SD card into your (powered-down) board, hold down the USER/BOOT button and apply power, either by the USB cable or 5V adapter**

Getting Started : Update board with latest software

BeagleBoard.org Latest Firmware Images

Download the latest firmware for your BeagleBoard, BeagleBoard-xM, BeagleBoard-X15, BeagleBone, BeagleBone Black, BeagleBone Black Wireless, BeagleBone AI, BeagleBone Blue, SeeedStudio BeagleBone Green, SeeedStudio BeagleBone Green Wireless, SanCloud BeagleBone Enhanced, element14 BeagleBone Black Industrial, Arrow BeagleBone Black Industrial, Mentorel BeagleBone uSomIQ, Neuromeka BeagleBone Air, or PocketBeagle

See the Getting Started guide and the community wiki page for hints on loading these images. See our Debian page on how the latest images are built.

Recommended Debian Images

Buster IoT TIDL (without graphical desktop and with machine learning acceleration tools) for BeagleBoard-X15 and BeagleBone AI via microSD card

- AM5729 Debian 10.3 2020-04-06 8GB SD IoT TIDL
  image for BeagleBoard-X15, and BeagleBone AI - more info - sha256sum: b9ac77af8be8156144b6192ed5d94404e381f19c0611042b26aadff18f49530e

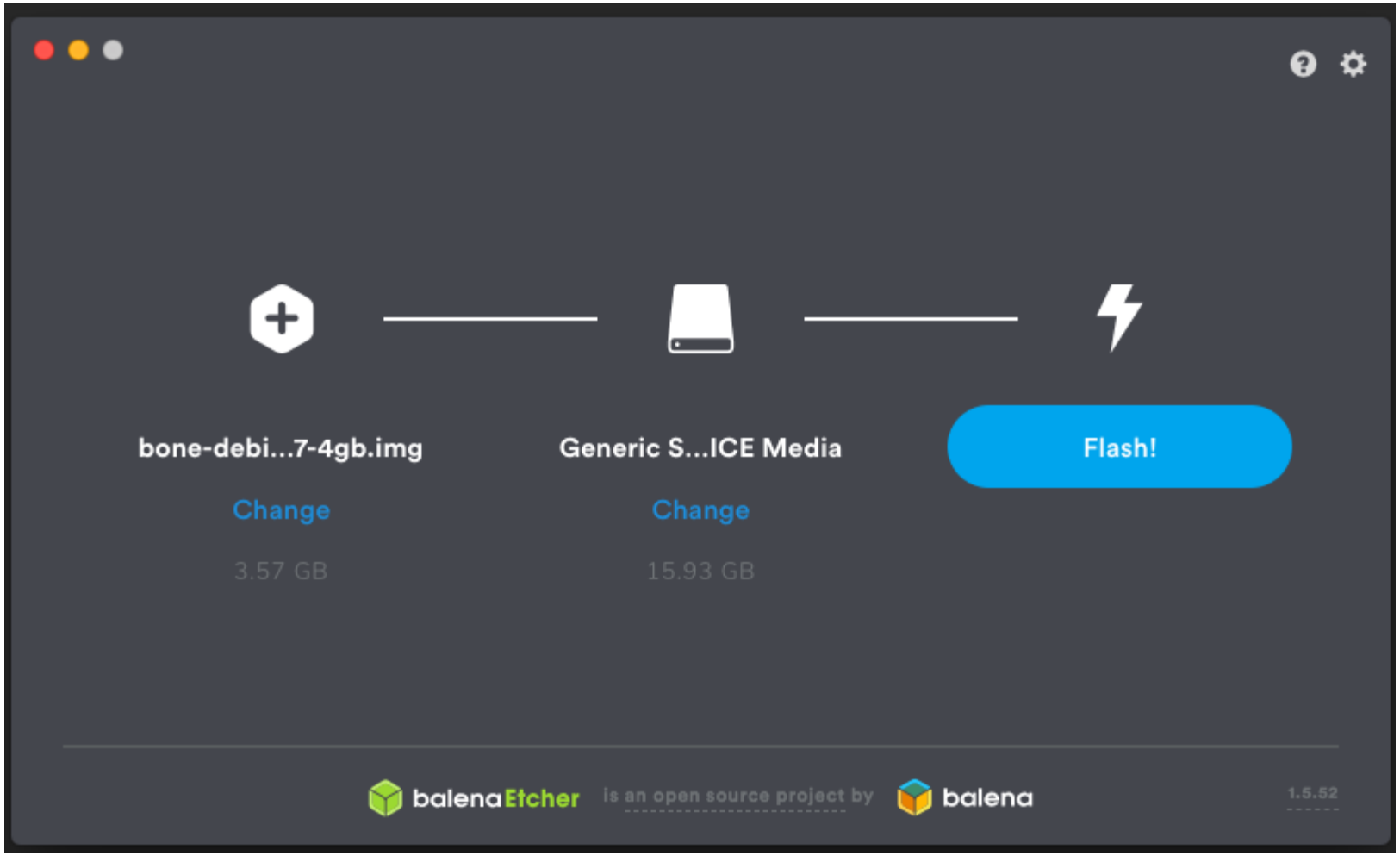
Buster IoT (without graphical desktop) for BeagleBone and PocketBeagle via microSD card

- AM3358 Debian 10.3 2020-04-06 4GB SD IoT
  image for PocketBeagle, BeagleBone, BeagleBone Black, BeagleBone Black Wireless, BeagleBone Black Industrial, BeagleBone Blue, SeeedStudio BeagleBone Green, SeeedStudio BeagleBone Green Wireless, SanCloud BeagleBone Enhanced, Arrow BeagleBone Black Industrial and Mentorel BeagleBone uSomIQ - more info - sha256sum: 22448ba28d0d58e25e875aac3b4e91eaef82e2d11c9d2c43d948ed60708f7434

Stretch for BeagleBoard via microSD card

- OMAP3/DM3730 Debian 9.5 2018-10-07 4GB SD LXQT image for BeagleBoard, BeagleBoard-xM - more info - sha256sum: 2a29626ab7c20890109a0eea4ea6e88e4e31d01d8a447b38eaac5953d8eb9ece

Getting Started : Update board with latest software



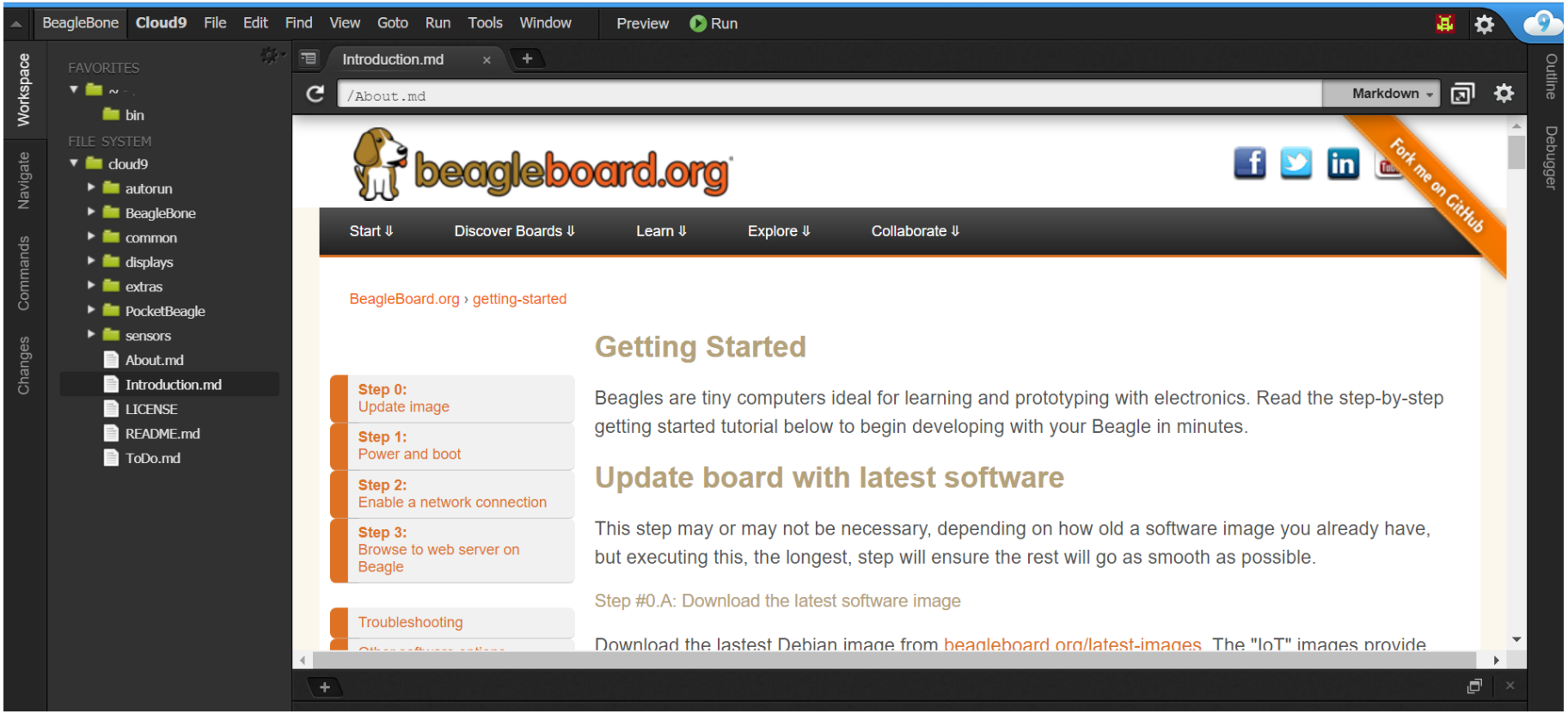
Getting Started : Step 1 - Start your Beagle

- **An USB cable is provided for us to have a convenient way to power the card and connect it to the computer**
  - **The microSD card should be installed before providing power**
  - **When connected, power (PWR or ON) LED lit steadily**
  - **Other LEDs blink in their default configurations**
    - USR0 blink in a heartbeat pattern
    - USR1 light during SD (microSD) card accesses
    - USR2 light during CPU activity
    - USR3 light during eMMC accesses

Getting Started : Step 2 - Network connection

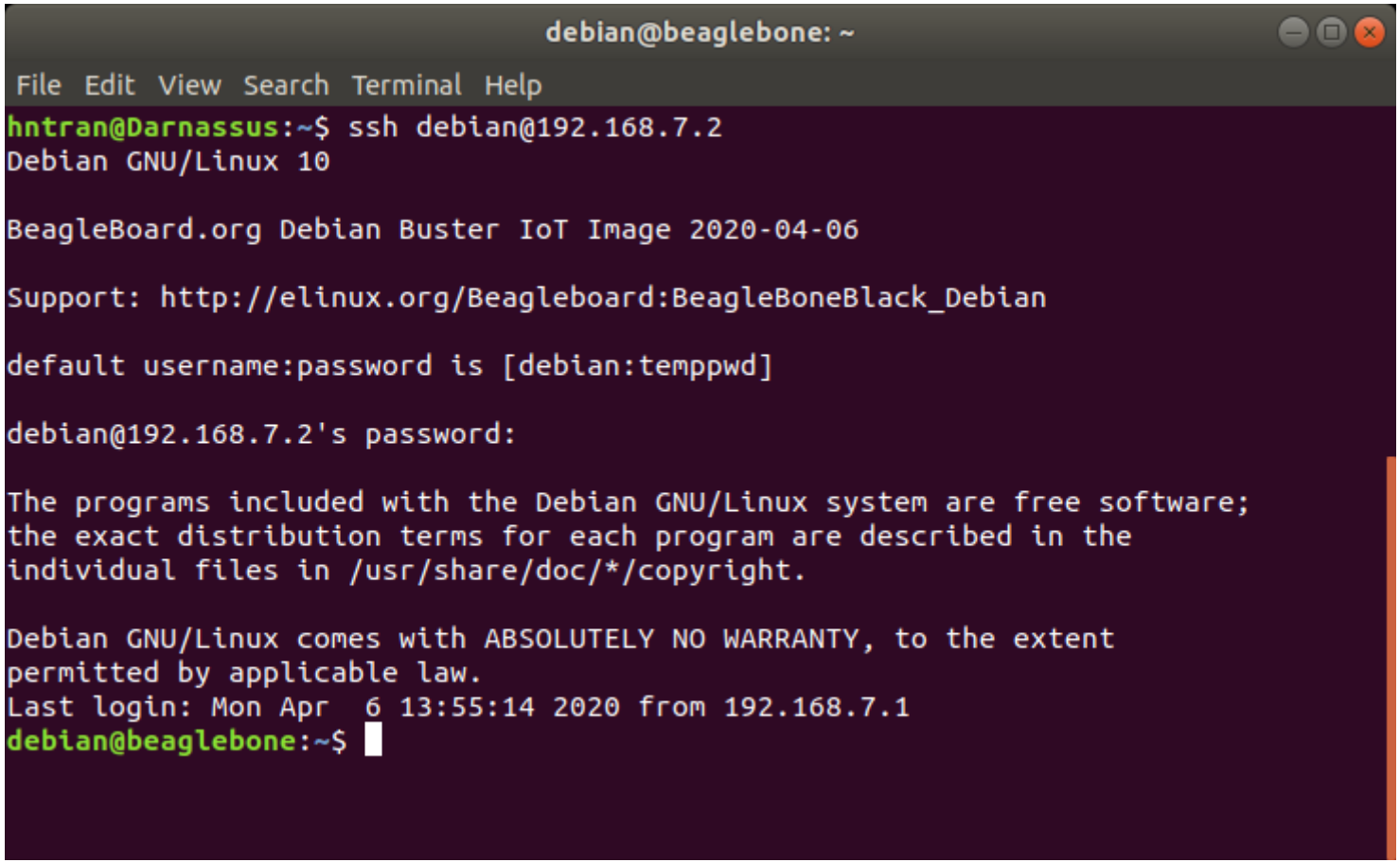
- **If connected via USB, a network adapter should show up on your computer**
  - **Your Beagle should be running a DHCP server that will provide your computer with an IP address of either 192.168.7.1 or 192.168.6.1**
  - **Your Beagle will reserve 192.168.7.2 or 192.168.6.2 for itself**

Getting Started : Step 2 - Network connection



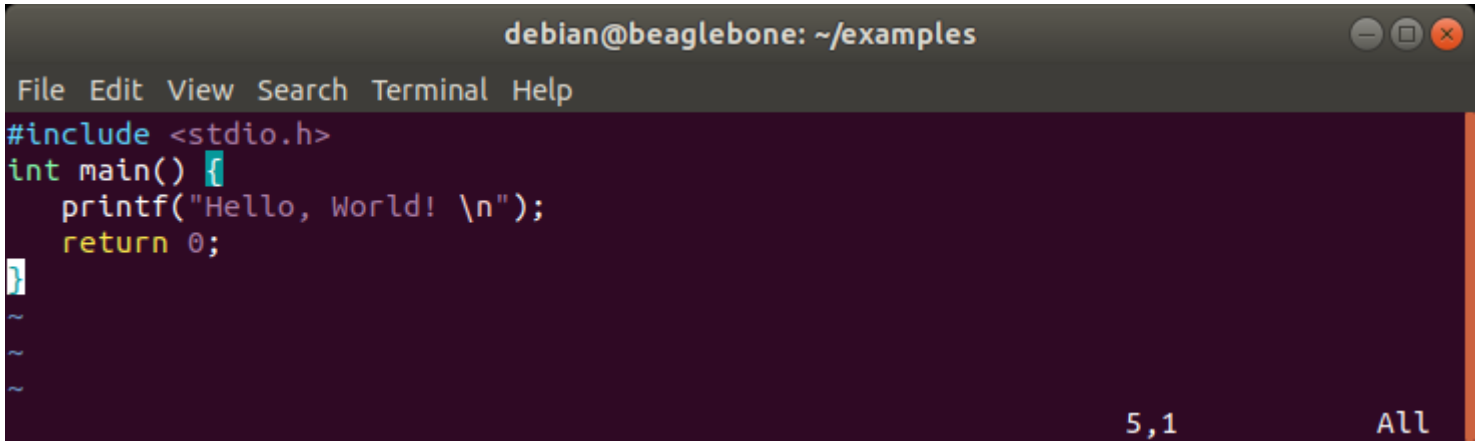
Connect to your card via ssh

- You can connect to your card by using the protocol ssh
  - `ssh debian@192.168.7.2`



Create and execute your first program

- **The image includes an development environment (compiler, text editor)**
  - **We can implement a program and compile it directly without the need of cross-compilation**
- **Use vi to create your first program**
  - **For example : helloWorld.c**



  - **Compile and execute it**
    - $: gcc helloWorld.c -o helloWorld
    - $: ./helloWorld

Create and execute your first program

- **You have executed your first program in an embedded environment**
  - **The program is compiled with the gcc corresponding to the processor of the card !**
- **However, this method of developing an application is quite limited**
  - **For larger projects, an IDE is needed**
    - **Solution 1 :** Develop on your host PC and deploy your code using **`scp`**

      `scp helloWorld.c debian@192.168.7.2:/home/debian`

    - **Solution 2 :** Install then use an IDE on your BeagleBone Black; for example : Cloud9 IDE