Bordeaux INP – ENSEIRB-MATMECA
Filière Systèmes Electroniques Embarqués

MI208 - Systèmes d'exploitation temps réel
Compte-Rendu TP3,4,5 et 6

Délivré le : 4 avril 2022

Mathieu SANCHEZ
Jean-Baptiste DIRIS

Table des matières

Première partie

Partie 1 : TP3 - Familiarisation avec la Beaglebone

1 Contexte

Durant ce TP nous avons pu utiliser une carte BeagleBone Black, grande soeur de la Raspberry Pi.
Afin de nous familiariser avec son fonctionnement, nous découvrirons grace à ce TP plusieurs manière d'interagir avec elle.

2 Familiarisation avec le fonctionnement de la carte

Plusieurs solutions existent pour faire fonctionner un noyau sur une carte embarquée.
Nous décrirons ici le téléchargement d'une image prête à l'emploi.
Sur le site de beagleboard, nous pouvons télécharger une image Debian afin de l'installer sur la carte MicroSD de la beagleboard.
Afin de programmer notre carte SD, nous utilisons le logiciel "balenaEtcher".
Une fois que la Beagleboard démarre sur le noyau de la carte SD, celle-ci se voit réserver l'adresse IP 192.168.7.1. Il est alors possible d'explorer le contenu du noyau avec une navigateur internet portant sur cette adresse IP.

3 Accès à la carte à partir d'une VM

Nous avons d'abord procédé à l'installation de gtkterm sur notre machine Dual-Boot Linux.
GTKTerm est un émulateur de terminal écrit, il est conçu pour l'environnement Gnome et est un clone du logiciel Hyperterminal. Il permet donc de communiquer avec tout périphérique connecté sur un port série.
Nous pouvons désormais communiquer avec notre carte à l'aide du port ttyACMO. C'est un port de communication USB.
Il nous est maintenant possible de naviguer au sein du noyau embarqué dans notre carte et ainsi explorer le système de fichiers.
Afin de transférer des programmes en langage C sur notre carte, nous utilisons le protocole SCP.
SCP est un protocole de transfert de fichiers de poste-à-poste basé sur SSH permettant de sécuriser les échanges. A l'aide de celui-ci, et du compilateur intégré sur la carte, nous pouvons alors lancer notre code issu du TP1 afin d'afficher le contenu du répertoire /proc.

Deuxième partie

Partie 2 : TP4 - Buildroot

1 Contexte

A travers ce TP, nous allons aller un peu plus loin dans notre familiarisation avec le fonctionnement de la Beaglebone. Celui-ci sera dédié à l'installation d'un nouvel environnement de développement croisé, afin d'explorer l'ensemble des possibilités de customisation de la carte.

2 Installation de Buildroot

2.1 Packages

L'installation de Buildroot nécessite plusieurs packages.
Un package est une archive comprenant les fichiers informatiques, les informations et procédures nécessaires à l'installation d'un logiciel sur un système d'exploitation, en s'assurant de la cohérence fonctionnelle du système modifié.
Voici la liste des packages nécessaires à Buildroot :

- *sed* : Sed est un éditeur non interactif. Cette commande permet d'appliquer un certain nombre de commandes sur un fichie. En lignes de commande, sed permet de modifier ou de supprimer une partie d'une chaîne de caractères, par exemple pour remplacer un caractère par un autre dans un fichier, ou encore supprimer des chaînes de caractères inutiles.

- *make* : Make est une commande qui fait appel à un fichier Makefile contenant les spécifications de compilation pour un projet.

- *binutils* : Binutils est un paquet qui contient les programmes utiles pour assembler, éditer les liens et manipuler des fichiers binaires et objets.
  Ils peuvent être utilisés avec un compilateur et diverses bibliothèques pour construire des programmes.

- *gcc* : (GNU Compiler Collection) est une suite de logiciels libres de compilation. Il est utilisé dans le monde Linux dès que l'on veut transcrire du code source en langage machine. GCC est le compilateur le plus répandu, il gère le C et ses dérivés, mais aussi le Java ou encore le Fortran.

- *g++* : Compilateur, comme gcc, mais pour le langage C++ (programmation orientée objet).

- *bash* : Bash (acronyme de Bourne-Again shell) est un interpréteur en ligne de commande de type script. C'est le shell Unix du projet GNU.

- patch : Patch est une commande pour mettre à jour un paquet ou fichier (application d'un patch).

- *gzip* : Gzip (acronyme de GNU zip) est un logiciel libre de compression de données.

- *bzip2* : Bzip2 est un logiciel libre de compression utilisant l'algorithme de compression du même nom.

- *perl* : Perl est un langage de programmation créé en 1987 pour traiter facilement de l'information de type textuel. Ce langage, interprété, s'inspire des structures de contrôle et d'impression du langage C, mais aussi de langages de scripts sed, awk et shell (sh).

- *tar* : Tar (tape archiver) est un outil pour la manipulation d'archives sous les systèmes Unix. Il ne compresse pas les fichiers, mais les concatène au sein d'une seule et même archive.

- *cpio* : Cpio est un utilitaire d'archivage ainsi qu'un format de fichier utilisé sur UNIX. Il était conçu à l'origine comme un moyen de sauvegarder des données sur bande magnétique sur les premières versions d'UNIX System III et UNIX System V. Il est maintenant dépassé en popularité par tar.

- *python* : Python est un langage de programmation en script.

- *unzip* : Unzip : extrait le contenu d'une archive zip.

- *rsync* : Rsync (pour remote synchronization ou synchronisation à distance), est un logiciel de synchronisation de fichiers. Il est fréquemment utilisé pour mettre en place des systèmes de sauvegarde distante.

- *wget* : Wget est un programme en ligne de commande non interactif de téléchargement de fichiers depuis le Web.

- *libncurses-dev* : libncurses-dev est un paquet de développement pour Linux embarqué.

Afin de télécharger buildroot, nous avons utilisé le lien directement donné sur le texte du TP. Une fois celui-ci installé, il est possible de paramétrer le noyau que nous souhaitons utiliser au travers de la commande "make menuconfig".

Les commandes Linux s'utilise pour les opérations sur les répertoires, elles permettent ainsi de créer, supprimer et gérer les répertoires du système via le terminal mais aussi de naviguer dans l'arborescence des répertoires.

Buildroot définit des configurations par défaut suivant le type de carte employé.
Dans notre cas la commande "make beaglebonedefconfig" nous permet de sélectionner les paramètres par défaut pour notre carte.
En plus de ces valeurs par défaut, nous sélectionnons les options suivantes :

- Dernière version du Kernel (5.3)
- Version des headers identiques à celles du Kernel

Un appel de la commande "make" nous permet de compiler l'ensemble de l'environnement Linux avec la configuration sélectionnée.

3 Installation de l'environnement embarqué

De la même manière que pour le TP3, nous installons notre environnement sur une carte micro SD.
Cette fois-ci, nous devons partitionner notre carte SD en deux partitions :

- Une partition "Bootloader" devant être conforme aux exigences de la plateforme AM335x. Cette plateforme contient le programme de démarrage ainsi que les fichiers suivants :

  — *MLO* : Il s'agit d'un micrologiciel ayant pour tâche le chargement du noyau de système d'exploitation. Il est exécuté après le boot ROM (en charge de la première étape du démarrage et du lancement de l'exécution du SPL) et avant l'exécution du noyau (en charge de l'initialisation du noyau)

  — *Zimage* : Zimage est l'ancien format pour les petits kernels (compressé, en dessous de 512 Ko). Au démarrage, cette image est chargée en mémoire (les premiers 640 Ko de la RAM). Elle est compressée, auto-extractible, et contient le noyau Linux.

  — *U-boot.img* : bootloader permettant de charger le système d'exploitation.

  — *am335x-boneblack.dtb* : device tree (contient la description du matériel)

  — *uEnv.txt* Le fichier uEnv.txt contient les variables d'environnement d'U-BOOT.

- Une partition contenant les systèmes de fichiers : Comme son nom l'indique, cette partition contient l'ensemble du système de fichiers. Nous formatons celle-ci en ext4 : ext est le système de fichiers par défaut de Linux qui supporte beaucoup de type de fichiers comme JFS, ReiserFS etc. Ce file system est actuellement en version 4 et porte donc le nom de ext4.

Les commandes suivantes nous sont données dans le sujet du TP pour créer les partitions :

- *sudo dd if=/dev/zero of=/dev/sdX bs=1M count=16* : permet d'écrire des 0 sur l'ensemble des premiers 16Mo de la carte SD, correspondant à l'espace de démarrage nécessaire.

- *sudo mkfs.vfat –F 16 –n boot /dev/sdc1* : Créer un système de fichier en fat16, portant le nom "boot" sur sdc1.

- *sudo mkfs.ext4 –L rootfs –E nodiscard /dev/sdc2* : Créer un système de fichier en ext4, portant le nom "rootfs" sur sdc2. L'option "nodiscard" permet de réduire le temps de formatage.

Nous copions l'ensemble des fichiers cités plus haut dans la partition boot. Par la suite nous exécutons la commande :

- *sudo tar –C /media/rootfs/ -xf output/images/rootfs.tar* : Permet d'extraire l'archive

"rootfs.tar" dans la partition créée juste avant. L'option -C nous permet de sélectionner une nouvelle directory, car la commande tar prend en compte le chemin courant en tant que chemin par défaut.

Nous pouvons désormais communiquer avec le nouveau noyau installé sur la carte SD. Le port utilisé pour communiquer via gtkterm est désormais "ttyUSB0".

- *ttyUSB* sert a connecter des périphériques type x86 (driver x86). Sur la machine virtuelle si un équipement de type x86 est connecté, il sera « branché » à un port « typé x86 » (ex : Prolific USB/Serial converter).

- *ttyACM* sert a connecter des périphériques ARM (driver Linux). Sur la machine virtuelle si un équipement de type ARM est connecté, il sera « branché » à un port « typé ARM » (ex : Carte Beagle Bone en USB).

4 Communication avec la carte

4.1 Description des protocoles

- *SCP* : SCP est un protocole de transfert de fichiers de poste-à-poste basé sur SSH permettant de sécuriser les échanges.

- *TFTP* : (Trivial File Transfer Protocol) est un protocole simplifié de transfert de fichiers. Il fonctionne en UDP sur le port 69. L'utilisation d'UDP, protocole « non fiable », implique que le client et le serveur doivent gérer eux-mêmes une éventuelle perte de paquets. On réserve généralement l'usage du TFTP à un réseau local.
  Les principales simplifications visibles du TFTP par rapport au FTP sont qu'il ne gère pas le listage de fichiers, et ne dispose pas de mécanismes d'authentification, ni de chiffrement. Il faut connaître à l'avance le nom du fichier que l'on veut récupérer. De même, aucune notion de droits de lecture/écriture n'est disponible en standard. À cause de ces fonctionnalités absentes, FTP lui est généralement préféré. TFTP reste très utilisé pour la mise à jour des logiciels embarqués, ou pour démarrer un PC à partir d'une carte réseau.

4.2 Explication des commandes

- *sudo apt-get install tftp tftpd xinetd*
  — Sudo : se substituer à root (nécessite le mot de passe root)
  — Apt-get : commande de récupération des paquets listés après sur le dépôt
  — Install : installer les paquets après téléchargement
  — Tftp : protocole de communication
  — Tftpd : serveur TFTP.
  — Xinetd : paquet servant à piloter l'accès à un ou plusieurs services réseaux.

- *sudo mkdir /tftpboot*
  — Mkdir : commande de création d'un répertoire.
  — /tftpboot : Le répertoire sera créé a la racine du système (/) et s'appellera tftpboot.

- *sudo chmod 777 /tftpboot*
  — Chmod : commande de changement de droits sur les fichiers et dossiers.
  — 777 : donne les droits en lecture, écriture et exécution a tout le monde.
  — /tftpboot : répertoire impacté par la commande.

- *sudo killall -HUP* xinetd
  — Killall : permet de tuer un processus par son nom.
  — -HUP : signal utilisé.

- *sudo service xinetd start*
  — service xinetd start : commande et nom du service à démarrer.

- *sudo service xinetd restart*
  — service xinetd restart : commande et nom du service à redémarrer.

Troisième partie

Partie 3 : TP5 - Compilation du noyau

1 Contexte

Ce TP nous permet de découvrir plusieurs options disponibles lors de la compilation d'un noyau linux avec les outils Buildroot.

2 Utilisation des leds

Le pilotage des leds se fait au travers de l'interaction avec le fichiers "leds" situé dans le répertoire "/sys/class".
Cependant, avec le noyau que nous utilisons, ce fichier est absent. Ceci est du à l'absence de sélection de certaines options lors de la configuration du noyau.
Pour les ajouter, nous devrons nous rendre dans le menu "linux-menuconfig" de buildroot. Ce menu permet de sélectionner différentes options relatives au noyau dont la gestion des leds.
Après avoir coché l'ensemble des options données dans le sujet du TP, nous pouvons bien remarqué l'apparation de 4 dossiers dans le dossier "/sys/class/leds" :

- *beaglebone :green :heartbeat* correspondant à la led 0
- *beaglebone :green :mmc0* correspondant à la led 1
- *beaglebone :green :usr2* correspondant à la led 2
- *beaglebone :green :usr3* correspondant à la led 3

Dans chacun de ces dossiers se trouvent principalement 2 fichiers permettant d'interagir avec les leds :

- *trigger* : Spécifie ce qui enclenche la led
- *brightness* : Spécifie l'allumage ou l'extinction de la led

Ainsi, à l'aide du fichier "GestionLeds.c", joint à ce compte-rendu, nous avons pu animer l'ensemble des leds à la manière de K2000, comme demandé.
Afin de pouvoir exécuter ce programme sur la beaglebone, nous avons du auparavant compiler à l'aide du compilateur "arm-linux-gcc" avant de transférer les binaires sur la Beaglebone.

Quatrième partie

Partie 4 : TP6 - Noyau temps réel

1 Contexte

Dans ce TP, nous allons installer puis tester différentes versions du noyau Linux afin de voirs les différentes versions de préemption du noyau possibles.

2 Exploration des options du noyau temps réel

Buildroot nous permet d'explorer les possibilités relatives à un noyau classique. Les options sont les suivantes :

- *No Forced Preemption (Server)* C'est le modèle traditionnel. Il offre de faibles temps de latence la plupart du temps, mais ce n'est pas garanti et occasionnellement quelques délais plus longs peuvent apparaître.

- *Preemptible Kernel (Low-Latency Desktop)* : Avec ce scheduler, tous les processus du noyau (qui ne sont pas en section critique) sont préemptibles. Cela permet au système d'être très réactif aux interactions en autorisant un processus à forte priorité d'être préempté, même s'il s'agit d'un processus noyau exécutant un appel système, et ceux sans avoir à atteindre un point depréemption définit (à la différence du scheduler précédent). A la réduction du débit de sortie, il faut ajouter un coût d'avantage élevé pour l'exécution du code du noyau.

- *Voluntary Kernel Preemption* : Cette option insère des points de descheduling supplémentaires. Concrètement, les appels système relancent volontairement le scheduler dans leur déroulement. Ces appels modifient peu le code, car ils redéfinissent simplement la fonction de débogage mightsleep() déjà existante et la font appeler le scheduler.

- *Preemptible Kernel (Basic RT)* : noyau temps reel mou (application multimedia). Il s'accommode de dépassements des contraintes temporelles dans certaines limites au-delà desquelles le système devient inutilisable ou pénible : visioconférence, jeux en réseau, etc. Un système temps réel souple doit respecter ses limites pour une moyenne de ses exécutions. On tolère un dépassement exceptionnel, qui pourra être compensé à court terme.

- *Fully Preemptible Kernel (RT)* : noyau temps réel dur, il ne tolère aucun dépassement de ces contraintes, ce qui est souvent le cas lorsque de tels dépassements peuvent conduire à des situations critiques, voire catastrophiques : pilote automatique d'avion, système de surveillance de centrale nucléaire, etc. On peut considérer qu'un système temps réel strict doit respecter des limites temporelles données même dans la pire des situations d'exécution possibles.

Nous avons pu, à l'aide des programmes générés lors du TP2, mesurer le temps de gigue du premier noyau de la liste, à savoir : Preemptible Kernel (Low-Latency Desktop). Les résultats sont les suivants :

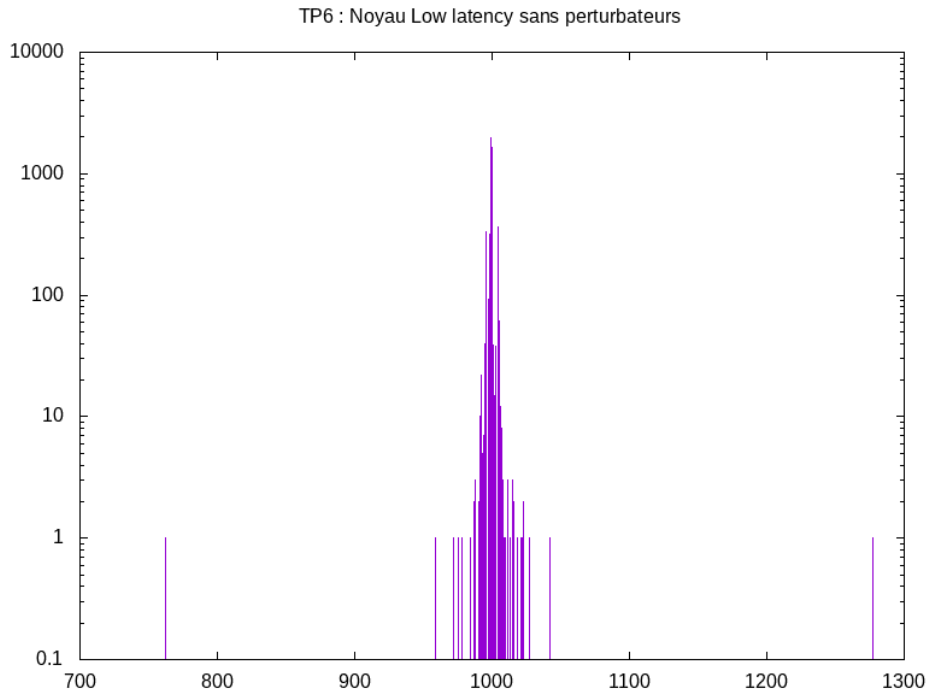

FIGURE 1 – Temps de gigue Noyau Low Latency sans Perturbations

Ce premier histogramme est généré en lançant le programme de mesure seul, c'est à dire sans perturbations. Nous remarquons que les mesures sont principalement centrées sur la période demandée lors de l'exécution du programme : 1000us.
Malgré l'oubli de sauvegarde des résultats de l'analyse en terme d'écart-type et de moyenne nous pouvons, à vu de cet histogramme, assurer que cette version du noyau a de grandes performances en terme de réactivité.

Les nouvelles mesures ci-dessous ont été effectuées en lançant en parallèle du programme, un processus perturbateur. Pour ce faire, nous avons créé le programme "GenePing.c" qui envoie des salves de ping aléatoires vers la carte depuis la VM.
Ainsi la carte reçoit un certain nombres d'interruptions pendant l'exécution des mesures. Nous constatons sur l'histogramme que les performances restent très bonnes et que ce noyau tient bien ses performances en étant préemptible.

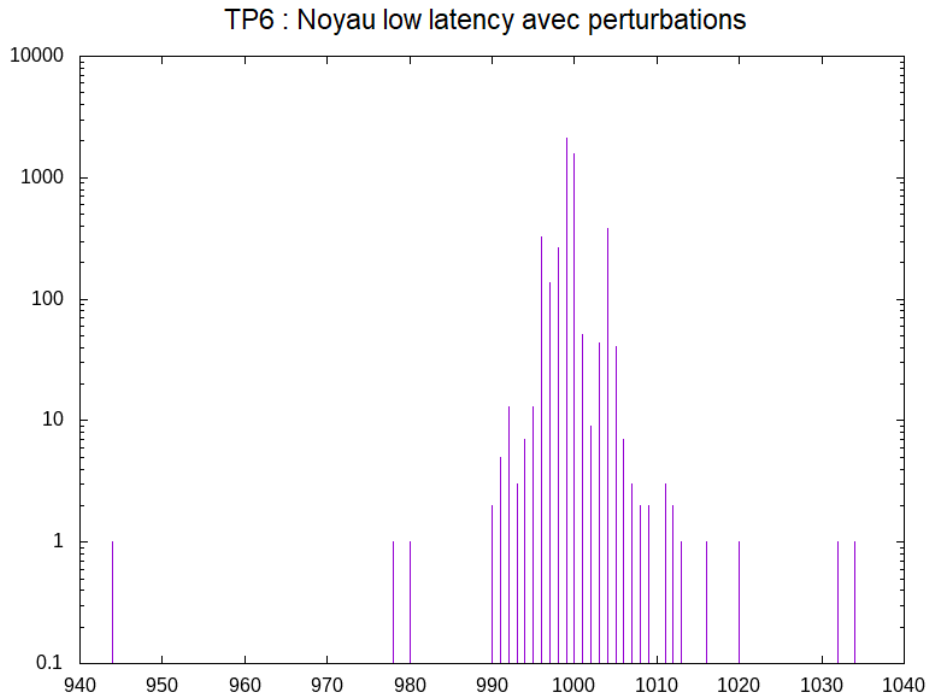

FIGURE 2 – Temps de gigue Noyau Low Latency avec Perturbations

3 Linux RT

Dans cette partie, nous allons utiliser un noyau Linux spécifique aux applications temps réel. Pour ce faire nous travaillons sur la version 5.2.21 accompagné d'un Patch. Cette version de Linux est appelée "Linux-rt".

Après compilation de ce nouveau noyau, nous avons pu effectuer un "dry run" sur le patch de celui-ci.
L'option –dry-run : simule l'installation d'un paquet ou patch. Cela permet de révéler certaines erreurs avant une installation en dur qui pourrait créer des problèmes. Cette opération n'ayant pas générer d'erreurs, nous avons pu patcher réellement le noyau.

Comme nous le voyons ci-dessous, ce patch nous donne accès à deux nouveaux modèles de préemption : "No forced Preemption" et "Fully Preemptible Kernel" :

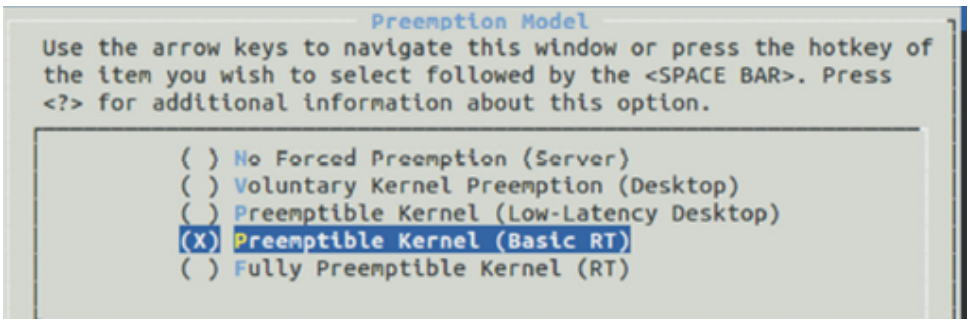

FIGURE 3 – Options de préemption du noyau

Afin d'analyser les performances de ces deux nouvelles options, nous avons effectué les mêmes tests que précédemment :

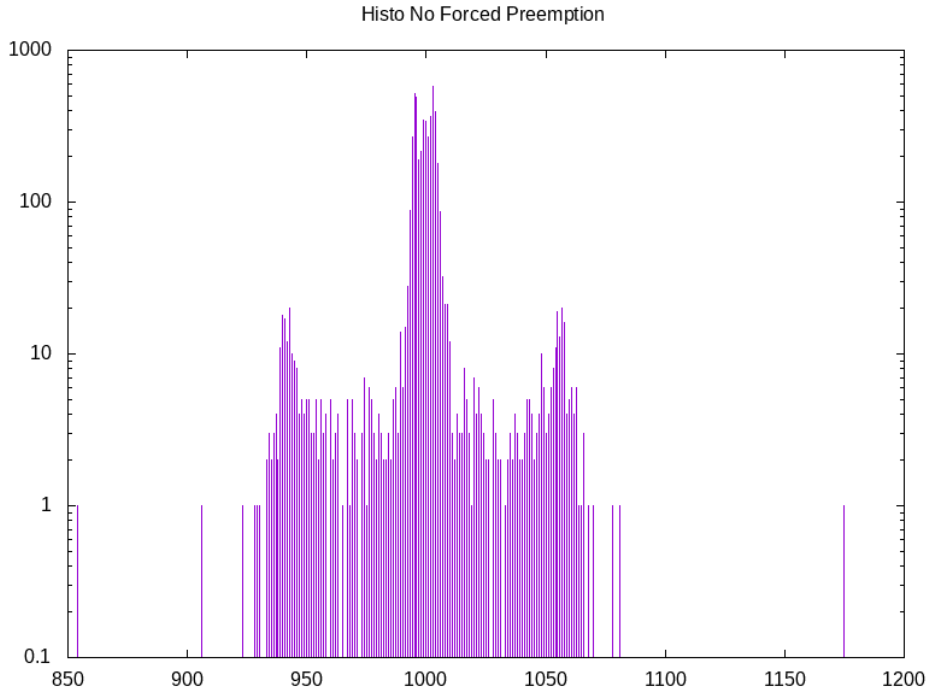

FIGURE 4 – Option No Forced Preemption avec perturbations

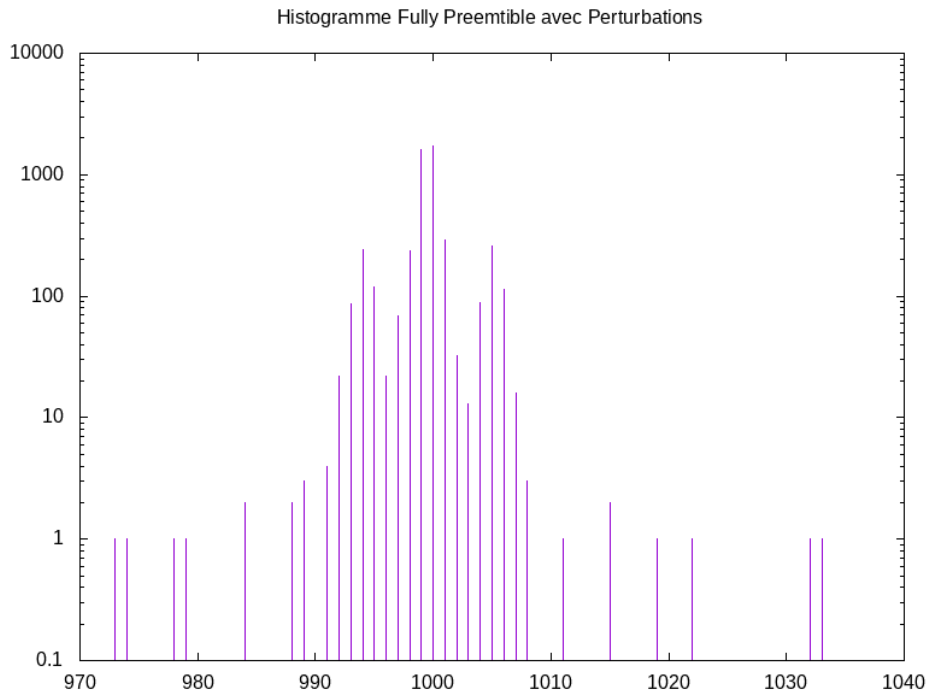

FIGURE 5 – Option Fully Preemptible avec perturbations

```
Nb mesures = 5000
Minimum = 973
Maximum = 1033
Moyenne = 1000
Ecart-type = 3
```

FIGURE 6 – Resultats noyau Fully Preemptible avec perturbations

Malgré l'absence de sauvegarde des résultats d'analyse pour le Kernel avec "No Forced Preemption", nous voyons que l'ensemble des mesures sont bien plus centrées et précises sur la période demandée (1000 us) dans le cadre du noyeu "Fully Preemtible", noyau pouvant être utilisé donc, dans le cadre des applications temps-réel.

4 Conclusion

Au travers de l'ensemble des manipulations réalisées, nous avons pu découvrir le monde des systèmes d'exploitation embarqués.
Ces applications nécessitent d'assurer non seulement un déterminisme au niveau des résultats mais aussi des timings.

Ce TP nous a montré l'importance de la maitrise de ces éléments mais également ce que pouvait être la compilation d'un noyau LINUX pour l'embarqué avec les différentes options et différences que cela pouvait apporter.