TP4 : La carte Armadeus APF28

Blottière Paul <blottiere.paul@gmail.com>

Table of Contents

1. Généralités

1.1. embsys

L'ensemble des cours, exemples, PDF et TP est disponible sur le dépôt github https://github.com/pblottiere/embsys.

Si vous voulez cloner entièrement le dépôt :

```
$ git clone https://github.com/pblottiere/embsys
```

Si vous voulez cloner le dépôt mais avoir simplement les labs dans votre répertoire de travail :

```
$ git clone -n https://github.com/pblottiere/embsys --depth 1
$ cd embsys
$ git checkout HEAD labs
```

Si vous souhaitez mettre à jour un dépôt que vous avez préalablement cloné :

```
$ cd embsys
$ git pull
```

Si vous n'avez pas **git**, téléchargez le ZIP sur la page d'acceuil de **embsys**.

Le TP d'aujourd'hui se trouve ici : https://github.com/pblottiere/embsys/labs/4_armadeus.

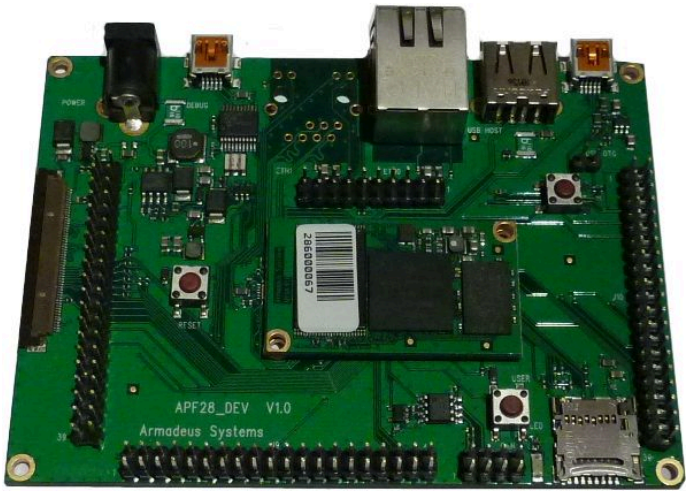
1.2. Attention particulière

Lors de ce TP, vous aurez les droits **root**. Vous devez donc être particulièrement **VIGILANT** à ce que vous faites ainsi qu'aux consignes des exercices afin de ne pas faire d'erreurs malencontreuses…

Il existe une règle simple pour éviter de telles erreurs : ne jamais être identifié root lorsque ce n'est pas nécessaire! Il ne s'agit donc pas ici d'être en root tout le long du TP…

1.3. Objectifs du TP

Dans le cadre de ce TP, nous allons travailler sur la carte Armadeus APF28 à travers sa plateforme de développement :



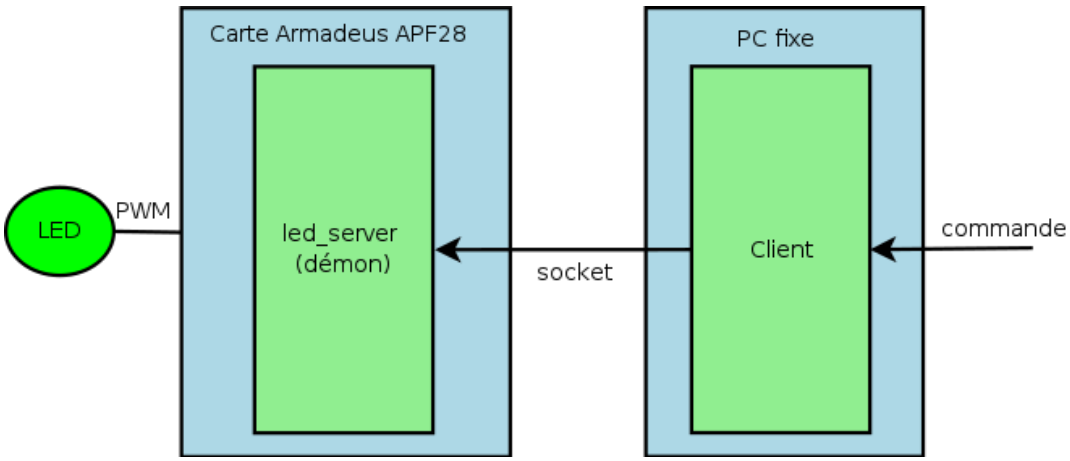
Le but est ici d'appréhender les points suivants :

- la lecture de datasheet
- la connexion à une carte par liaison série
- l'upload et le flashage d'images (kernel et rootfs)
- la cross-compilation d'un logiciel utilisateur
- l'utilisation des GPIO dans l'espace utilisateur
- une révision de syslog et des sockets

Pour aborder chacun de ces points, nous allons réaliser un serveur embarqué sur la carte APF28 chargé de modifier l'intensité lumineuse d'une LED. Cette intensité pourra être modifiée grâce à des messages envoyés sur le réseau.

Il faudra donc aussi un programme côté utilisateur permettant d'envoyer les commandes au serveur et de lire les messages envoyés par le serveur sur le réseau.

L'architecture en image :



2. Exercice 1 (1h30) : Flashage de la carte APF28

2.1. Description de l'exercice

Pendant cet exercice, nous allons suivre le déroulement classique de travail sur une nouvelle carte :

- trouver le site web du constructeur et télécharger le BSP
- analyse rapide du contenu (système d'automatisation utilisé, bootloader, …)
- les différentes règles de compilation disponibles
- compilation de la distribution
- flashage de la carte avec les images compilées
- boot de la carte pour vérifier le bon fonctionnement

En réalité, pendant cet exercice, nous n'allons pas compiler le BSP, simplement pour une question de temps de compilation. Nous allons donc partir d'images déjà compilées.

2.2. Questions

Découverte du BSP

Dans un premier temps, rendez-vous sur le site ci-dessous et récuperez le BSP/SDK (Board Support Package) stable, actuellement la version 6.0 :

http://www.armadeus.com/wiki/index.php?title=Toolchain

Détarrez la tarball. Suite à l'extraction, vous pouvez observer divers répertoires/fichiers.

Question 1 (2 min) : *D'après une analyse rapide du contenu, quel type de système de build automatisé est utilisé?*

Question 2 (5 min) : *Selon vous, que contient le répertoire **patches**? Quels sont les différents sous répertoires et leurs utilités?*

Question 3 (2 min) : *D'après l'étude du répertoire **patches** de la question précédente, que pouvez vous dire sur le bootloader utilisé?*

Maintenant que l'on connait le système de build automatisé ainsi que le bootloader utilisé, nous allons regarder comment compiler notre distribution.

De manière générale, il y a globalement trois étapes lors de la compilation :

1. une règle de configuration générale, indiquant par exemple le type de carte sur laquelle nous allons travailler

2. une phase de configuration plus précise ouverte à l'utilisateur

3. une règle de compilation se basant sur les éléments configurés lors des étapes précédentes.

Question 4 (5 min) : *En étudiant le contenu du fichier **Makefile**, indiquez quelle règle de **configuration** doit être utilisée pour la carte APF28.*

Lancez cette commande de configuration.

Question 5 (15 min) : *Qu'observez vous? Fouillez le contenu de l'interface, discutez et faite le rapprochement avec les notions vues en cours. Trouvez la version du kernel ainsi que le type/la version de libc.*

Lorsque vous êtes dans l'interface, allez sur **exit** et appuyez sur **entrée**. À partir de ce moment, un fichier de configuration **buildroot/.config** est créé et sera utilisé lors de la compilation de la distribution.

Question 6 (5 min) : *En étudiant ce fichier de configuration, déterminez l'ABI utilisée (vue en cours). Par rapport aux discussions eues en cours sur les différentes ABI d'une architecture ARM, que pensez vous de ce choix?*

Question 7 (2 min) : *Toujours par rapport aux notions vues en cours, rappellez l'utilité de **busybox**.*

Question 8 (5 min) : *En étudiant le fichier **Makefile**, déterminez quelle commande faut-il utiliser pour ouvrir le menu de configuration de busybox?*

Lancez cette commande de configuration. Dans une telle interface, l'utilisateur peut utiliser la commande **/** pour faire une recherche sur une chaine de caractères (comme dans vi).

Question 9 (10 min) : *Dans le menu de configuration de busybox, déterminez si busybox fournira **fdisk**, **ping** et **ssh**. Rappellez au passage l'utilité de ces comandes.*

Nous sommes maintenant à un stade où on connaît globalement le BSP de notre carte.

En deux commandes simples, nous pouvons donc configurer la carte et lancer la phase de compilation :

```
> make apf28_defconfig
```

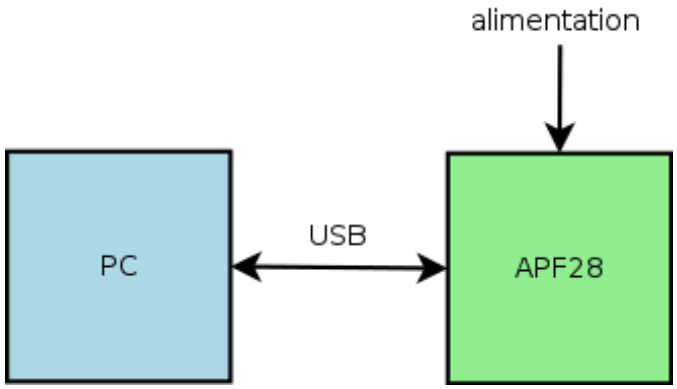
```
> make
```

Cependant, la phase de compilation étant trop longue, nous n'allons pas réaliser cette étape, mais plutôt partir d'images précompilées. Dans le cas d'une compilation via le BSP, ces images seraient disponibles ici : **buildroot/output/images**.

Flashage de la carte

Maintenant que nous avons les images, nous allons commencer à travailler avec la carte en tant que telle. Tout d'abord, nous allons essayer de communiquer par liaison série (via cable USB).

Réalisez donc le montage suivant :



Suite à cela, une entrée est créée dans le répertoire **/dev** grâce au mécanisme de hotplug du kernel et à udev.

Question 10 (2 min) : *Rappellez l'utilité de la commande **dmesg**. Quel est l'entrée dans le /dev permettant de comuniquer avec l'APF28?*

Comme vue en cours, une configuration classique de communication série est du 8N1 (bits de données / bits de parité / bit d'arrêt). La carte APF28 n'échappe pas à la règle. Ici, le débit est 115200 bauds.

Question 11 (10 min) : *Utilisez minicom pour vous connecter à la carte et appuyez sur le bouton **reset** de la carte. Que voyez vous? Que se passe t-il?*

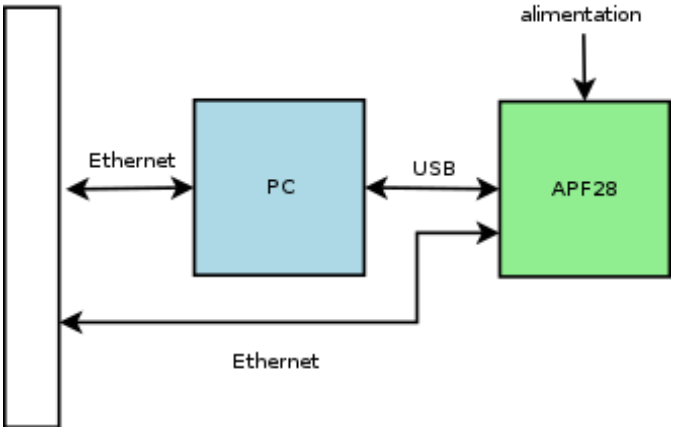
Refaite un reset de la carte mais cette fois-ci interrompez la phase de boot (en appuyant sur une touche pendant le compte à rebour) afin d'arriver dans le prompt de U-boot. La commande **printenv** permet d'afficher les variables d'environnement/macros actuellement enregistrés sur la flash de la carte.

Question 12 (10 min) : *Grâce à la commande **printenv**, déterminez les macros permettant de télécharger et d'enregistrer de nouvelles versions du bootloader, du kernel et du RFS.*

Question 13 (10 min) : *En étudiant de plus prêt les commandes permettant de flasher le kernel et le RFS, déterminez les adresses flash où doivent être enregistrées ces images.*

Pour télécharger les images sur la carte, il faut que celles-ci soient sur un serveur TFTP (voir cours). Dans notre cas, il existe un serveur TFTP distant ayant l'adresse IP : 172.20.5.2. Les images sont dans le répertoire /tftpboot.

Maintenant, connectez la carte APF28 au réseau de l'école par Ethernet.



Nous allons maintenant récupérer une adresse IP par DHCP afin que la carte APF28 soit sur le réseau.

Question 14 (2 min) : *Lancez la commande **dhcp** dans le prompt uboot. Quelle est l'IP de la carte à l'issue de cette commande?*

Dans le prompt U-boot, la commande **setenv** permet de fixer une variable d'environnement et la commande **saveenv** de l'enregistrer sur la flash :

```
> setenv myvar coucou
> saveenv
```

Question 15 (5 min) : *En analysant dans U-boot les commandes de téléchargement des images, déterminez dans quelle variable d'environnement l'adresse IP du serveur TFTP est enregistrée.*

Grâce aux commandes **setenv** et **saveenv**, sauvegardez en flash dans la variable d'environnement correspondante (question précédente) l'adresse IP du serveur TFTP.

Nous pouvons utiliser la commande **ping** afin de voir si la liaison avec le serveur TFTP est opérationnelle :

```
> ping ${LA_VARIABLE_STOCKANT_L_IP_DU_SERVEUR_TFTP}
```

Dans le prompt U-boot, lancez les macros d'update du kernel et du RFS (on ne mettera pas à jour le bootloader pendant ce TP) :

```
> run update_kernel
> run update_rootfs
```

Ensuite, rebootez la carte et laissez la phase de boot se dérouler jusqu'au bout. Vous arrivez alors à un prompt de login de votre distribution! La phase de flashage est alors terminée.

Jouez un peu avec votre distribution fraichement installée :

- changez le mot de passe root avec la commande **passwd**
- connectez vous en SSH sur la carte APF28
- copiez un fichier quelconque avec SCP
- …

3. Exercice 2 (1h30) : Cross-compilation, GPIO et syslog

3.1. Description de l'exercice

À partir du moment où on possède une carte fonctionnelle, la question de cross-compilation se pose. Dans le cas d'une compilation from scratch des images à flasher, le cross-compilateur est disponible à travers le BSP. Le but de l'exercice est donc :

- d'étudier une chaine de cross-compilation précompilée
- de compiler/cross-compiler un binaire hello-world
- d'apprendre à utiliser Syslog
- de jouer avec une LED pour découvrir l'utilisation des GPIO dans l'espace utilisateur

3.2. Questions

Cross-compilation

Dans un premier temps, écrivez un hello-world et utilisez le gcc natif du PC fixe. Utilisez SCP pour copier le binaire résultant sur la carte APF28.

Question 16 (10 min) : *Exécutez le binaire sur la carte. Qu'observez-vous? Lancez la commande **file** sur le binaire. À quoi sert cette commande selon vous?*

Un cross-compilateur est accessible sur la machine **saltp7-l** dans le répertoire /appli/arma/armadeus/buildroot/output/host/usr/bin. Connectez vous en SSH sur cette machine avec votre login personnel.

Question 17 (5 min) : *Dans le répertoire cité ci-dessus, déterminez le path du cross-compilateur pour du code C.*

d Cross-compilez le hello-world pour l'APF28, copiez le sur la carte et exécutez le.

Syslog

Sous GNU-Linux, les logs enregistrés dans le répertoire /var/log sont gérés par le démon **syslog**. Vous pouvez trouver un exemple d'utilisation ici : **labs/4_armadeus/src/syslog**.

Question 18 (5 min) : *Compilez le code de syslog avec gcc et testez le.*

Question 19 (5 min) : *Utilisez **grep** dans /var/log pour déterminer dans quel fichier les messages sont écris.*

Question 20 (15 min) : *En étudiant le fichier syslog.c, décrivez l'utilisation des fonctions **setlogmask**, **openlog**, **syslog** et **closelog**.*

Cross-compilez le fichier syslog.c et testez sur la carte APF28.

GPIO

Nous allons utiliser un GPIO de l'APF28 pour allumer/éteindre la led présente sur la carte à côté du lecteur de mini SD.

Sous Linux, un GPIO est défini par un numéro. En revanche, côté matériel, un GPIO est défini par un numéro de banque et un numéro de pin. Dans le cas de l'APF28dev, le numéro sous Linux est calculé comme suit :

```
gpio_linux = (32 * bank_number) + pin_number
```

Question 21 (15 min) : *Trouvez sur internet la datasheet de la plateforme de développement de l'APF28 et fouillez dans la documentation pour trouver les numéros caractérisant le GPIO qui gère la LED utilisateur.*

Question 22 (2 min) : *En déduire le numéro du GPIO qui doit être utilisé sous Linux.*

Question 23 (15 min) : *Utilisez les notions vues en cours pour allumer et éteindre la LED dans le shell.*

Question 24 (20 min) : *Écrivez un code C permettant de gérer la LED. Ce programme doit agir comme un simple toggle.*

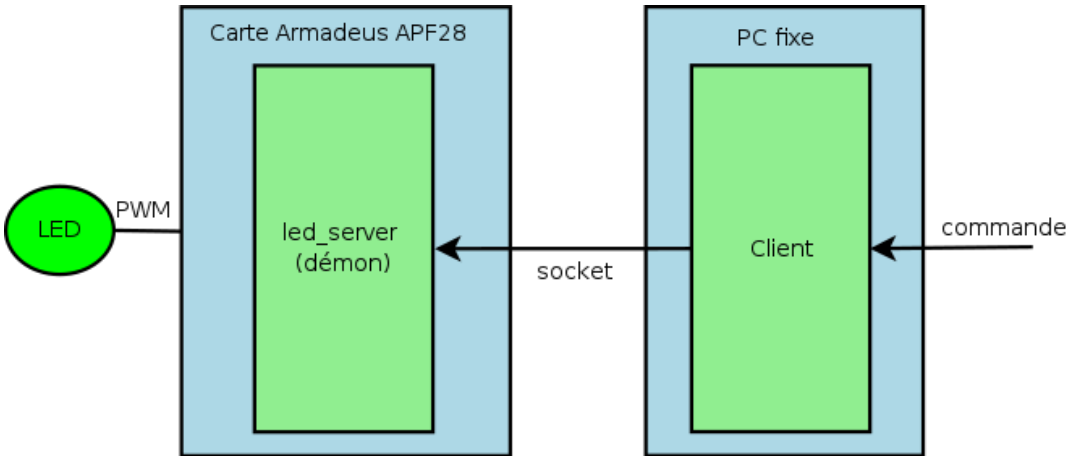
4. Exercice 3 (3h45) : Serveur de LED

4.1. Description de l'exercice

Dans cet exercice, nous allons développer une petite application serveur permettant de contrôller une LED reliée à la board APF28. Cette LED devra être connectée à une pin PWM pour faire varier la tension, et donc l'intensité lumineuse. Une connexion réseau par socket permettra d'envoyer des messages au serveur indiquant l'intensité lumineuse souhaitée.

En plus des aspects techniques fondamentaux, nous allons profiter de cet exercice pour étudier le système de build GNU (les autotools) ainsi que l'outil de gestion de versions GIT.

Voici l'architecture logicielle globale attendue :



4.2. Étape 1 (15 min) : GIT

Par définition, un outil de gestion de versions permet basiquement de garder l'historique de travail des fichiers d'un projet. Durant la vie du projet, les fichiers (par exemple le code source) vont évoluer et grâce à l'outil de gestion de versions, une trace de chaque état intermédiaire est accessible. Cela permet donc de contrôler le cycle de vie d'un logiciel. On peut d'ailleurs faire un snapshot d'un moment particulier pour fixer une version précise d'un logiciel : c'est ce qu'on appelle "tagger".

Il existe de nombreux outils de gestion de versions :

- CVS
- Mercurial
- SVN
- GIT
- …

Celui que nous allons utiliser aujourd'hui est très à la mode : GIT. Pour la petite histoire, GIT a été initialement entrepris par Linus Torvald pour gérer les versions du kernel Linux car il voulait combler certains manques des outils existants.

Il faut savoir qu'il existe la notion de branches permettant par exemple de travailler sur une fonctionnalité particulière sans affecter le reste du projet. Dans le cadre de cet exercice, nous allons simplement travailler dans la branche par défaut : la branche *master*.

Dans un premier temps, créez un répertoire de travail nommé **led_server**. Allez dans ce répertoire et initilisez la gestion de version de votre logiciel avec la commande suivante :

```
> cd led_server
> git init
Initialized empty Git repository in /YOUR/PATH/led_server/.git/
```

Un répertoire **.git** est alors créé. À partir de ce moment, les commandes git peuvent être appellées dans ce répertoire. Il existe par exemple une commande permettant de voir le statut actuel de travail.

```
> ls -a
.  ..  .git
> git status
On branch master

Initial commit

nothing to commit (create/copy files and use "git add" to track)
```

On appercoit alors que nous travaillons actuellement sur la branche par défaut nommée *master* et qu'il n'y a rien à **commiter**. La notion de commit est relativement simple : une modification a été réalisée et on veut garder dans l'historique de travail la version actuelle. Par exemple, créez un fichier README (généralement indispensable à un projet) avec pour titre "LED Server" et committez. Notez qu'il faut d'abord "tracker" un fichier avant de pouvoir lancer un commit sur celui-çi :

```
> touch README
```

```
> git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    README

nothing added to commit but untracked files present (use "git add" to track)
> git add README
> git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   README
> git commit -m "My commit message to describe my work"
[master (root-commit) 4d62e36] My commit message to describe my work
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 README
> git status
On branch master
nothing to commit, working directory clean
```

Une commande permet aussi de visualiser les différents commits réalisés :

```
> git log
Author: Blottiere Paul <blottiere.paul@gmail.com>
Date:   Sun Jan 17 16:55:12 2016 +0100

    My commit message to describe my work
```

On peut par exemple faire un tag de notre version actuelle pour indiquer qu'il s'agit d'une version 0.1 de notre logiciel :

```
> git tag 0.1
> git tag
0.1
```

On pourra alors retourner à des tags précédents (par exemple pour naviguer dans le code source et analyser les modifications) grâce à la commande *git checkout*.

Nous avons ici les prémices de GIT. Cela devrait déjà vous permettre de travailler basiquement avec cet outil de gestion de version pendant le reste de l'exercice.

4.3. Étape 2 (1h00) : autotools et cross-compilation

Un outil de build permet de gérer les différentes phases de travail de déploiement d'un logiciel :

- configuration
- compilation

- installation / désinstallation

Il existe de nombreux systèmes de build, en fonction des langages utilisés, des plateformes et plus généralement des besoins et exigences de chacun. Pendant cet exercice, nous allons utiliser un système de build ultra-classique : les autotools.

Tout d'abord rendez-vous dans le répertoire **labs/4_armadeus/src/autotools**. Dans ce mini-projet, les autotools sont utilisés pour gérer l'étape de compilation du client de notre serveur de LED. Il y a aussi un mini serveur permettant de tester le bon fonctionnement de notre application.

```
> cd labs/4_armadeus/src/autotools
> ls
autogen.sh  configure.ac  Makefile.am
```

Les fichiers **configure.ac** et **Makefile.am** sont les marques de fabriques des autotools. On peut alors lancer la phase de configuration :

```
> ./autogen.sh
Running aclocal...
aclocal: warning: couldn't open directory 'm4': No such file or directory
Running libtoolize...
libtoolize: putting auxiliary files in AC_CONFIG_AUX_DIR, `config'.
libtoolize: linking file `config/ltmain.sh'
libtoolize: Consider adding `AC_CONFIG_MACRO_DIR([m4])' to configure.ac and
libtoolize: rerunning libtoolize, to keep the correct libtool macros in-tree.
libtoolize: Consider adding `-I m4' to ACLOCAL_AMFLAGS in Makefile.am.
Running autoheader...
Running autoconf...
Running automake...
configure.ac:7: installing 'config/ar-lib'
configure.ac:7: installing 'config/compile'
configure.ac:11: installing 'config/config.guess'
configure.ac:11: installing 'config/config.sub'
configure.ac:6: installing 'config/install-sh'
configure.ac:6: installing 'config/missing'
> ./configure
...
config.status: creating config.h
config.status: executing depfiles commands
config.status: executing libtool commands
configure:

  autotools 0.1 - Configuration Report

 prefix:            /usr/local
 exec-prefix:       ${prefix}
 bin path:          ${exec_prefix}/bin
 lib path:          ${exec_prefix}/lib
 include path:      ${prefix}/include
 etc path:          ${prefix}/etc
 data path:         ${prefix}/share
```

Cette phase de configuration, lancée dans un premier temps sans options, vérifie si tout le nécessaire (librairies, headers, …) est présent pour que la phase de compilation se passe bien.

Question 24 (5 min) : *En analysant le contenu du fichier configure.ac, déterminez quel header est vérifié pendant la phase de configuration.*

Question 25 (5 min) : *Modifiez le fichier configure.ac pour forcer l'étape de configuration à vérifier la présence d'un header inexistant sur le système. Relancez la phase de configuration et observez les plaintes! Remettez les choses dans l'ordre lorsque vous avez terminé.*

Question 26 (5 min) : *En cherchant sur le net, déterminez à quoi sert la macro AC_ENABLE_SHARED utilisée dans le fichier configure.ac.*

Question 27 (5 min) : *À quoi sert le fichier Makefile.am?*

Lorsque la phase de configuration se passe bien, on peut lancer la phase de compilation :

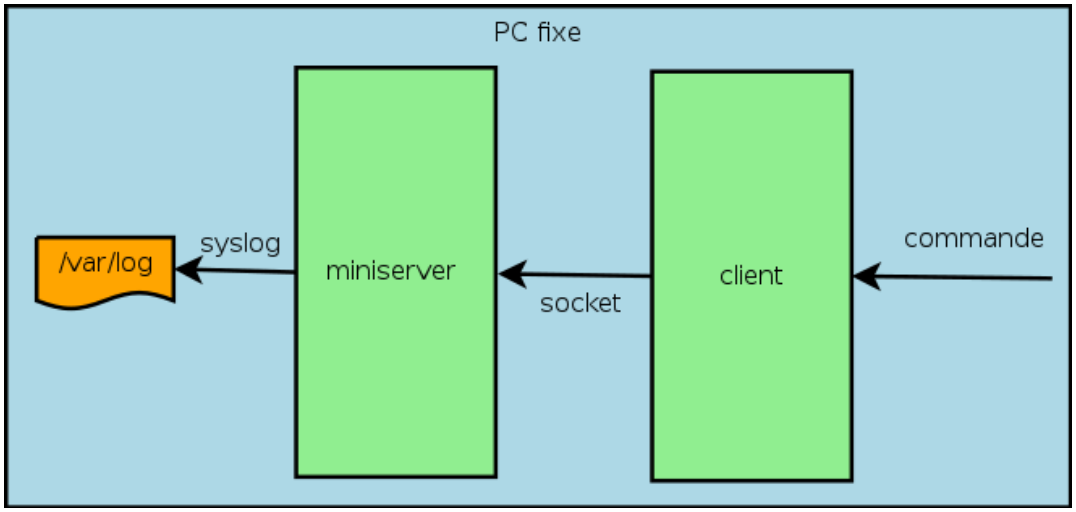
```
> make
```

Suite à cette étape, un binaire **src/client/client** est généré.

Question 28 (10 min) : *Modifiez le fichier **Makefile.am** pour qu'il compile le fichier **src/miniserver/miniserver.c** en un binaire nommé **miniserver**.*

Le miniserveur écoute sur un port réseau et log les messages reçus grâce à syslog. Le client, quant à lui, envoie des messages sur le réseau à une adresse IP particulière et sur un port particulier (par défaut le port 5000 codé en dur).

Question 29 (10 min) : *Grâce à l'aide des binaires **client** et **miniserver** (lancez les binaires sans options), testez leur bon fonctionnement en local (voir schéma ci-dessous).*



Lors de la phase de compilation, le compilateur natif est utilisé. Dans le cadre d'une cross-compilation, on peut utiliser l'option *--host* lors de la configuration pour indiquer l'utilisation d'un compilateur spécifique au moment du make. Par exemple :

```
> ./configure --host=arm-linux
...
checking for arm-linux-gcc... arm-linux-gcc
...
```

Lors de la compilation, le compilateur **arm-linux-gcc** sera alors utilisé. Cependant, il faut pour cela que le path du cross-compilateur soit dans la variable d'environnement PATH. Comme le compilateur pour l'APF28 se trouve sur la machine saltp7-l, il faut se connecter sur celle-çi puis faire :

```
> export PATH=$PATH:/appli/arma/armadeus/buildroot/output/host/usr/bin
```

Suite à cela, le cross-compilateur doit être accessible dans le shell sans avoir besoin d'indiquer un chemin absolu.

Question 30 (2 min) : *Relancez la phase de configuration/compilation pour cross-compiler le binaire miniserver pour l'APF28 sur saltp7-l.*

Question 31 (15 min) : *Utilisez le client natif côté PC fixe et le serveur cross-compilé sur l'APF28 pour vérifier leur bon fonctionnement.*

Placez vous dans le répertoire **labs/4_armadeus/src/autotools** et lancez la commande **make clean-maintainer**.

Question 32 (2 min) : *En observant l'effet, déterminez à quoi sert cette dernière commande.*

Question 33 (5 min) : *Copiez les fichiers originaux (c'est à dire sans les fichiers générés par les autotools) du répertoire **autotools** dans votre répertoire de travail **led_server** en gardant la même arborescence. Commitez vos modifications avec git.*

Question 34 (5 min) : *Dans votre répertoire **led_server**, lancez la phase de configuration des autotools. Puis faite un **git status** et observez le résultat. Qu'en pensez-vous? Que doit-on faire?*

En effet, comme vous l'avez déduit lors de la question précédente, les fichiers générés ne doivent pas être commités car ils sont plateforme dépendants. Cependant, leur présence pollue la sortie de **git status**. Il existe avec git un moyen d'ignorer purement et simplement des fichiers via l'utilisation d'un fichier **.gitignore**.

Créez un fichier .gitignore dans votre répertoire de travail **led_server** et éditez le pour obtenir ceci :

```
Makefile.in
aclocal.m4
autom4te.cache/
config.h.in
config/
configure
Makefile
config.h
config.status
libtool
m4/
stamp-h1
.deps
.dirstamp/
```

Question 35 (5 min) : *Suite à cela, relancez un **git status**. Quel changement observez vous par rapport à la question 34?*

Question 36 (1 min) : *Commitez le fichier .gitignore.*

4.4. Étape 3 (1h00) : init.d et GPIO

Dans le cas de notre serveur, on va vouloir que celui-ci soit lancé dès le démarrage de notre OS. Le serveur va alors tourner en tâche de fond, c'est ce qu'on appelle un **démon**.

On va dans un premier temps créer un serveur nommé **led_server** à partir du miniserveur étudié dans les question précédentes. Ensuite, nous allons le modifier pour que celui-ci allume une LED à son démarrage. Nous pourrons ensuite le lancer en tant que démon sur l'APF28. La LED donnera alors une indication à l'utilisateur concernant son état : online ou offline.

Question 37 (15 min) : *Copiez le répertoire led_server/src/miniserver en led_server/src/led_server et renommez le fichier en led_server.c. Modifiez le fichier Makefile.am pour obtenir en plus du client et du miniserver un binaire appellé* ***led_server*** *lors de la compilation.*

Question 38 (15 min) : *En vous inspirant du travail réalisé lors de l'exercice 2, modifiez le code de led_server.c pour que la LED utilisateur de l'APF28 s'allume dès que le serveur est prêt à recevoir des messages. De même, faite en sorte que la LED s'éteigne lorsque le serveur s'arrête.*

Question 39 (10 min) : *Cross-compilez votre miniserver et testez son fonctionnement sur la carte.*

Question 40 (2 min) : *Suivez les directives de la page http://www.armadeus.com/wiki/index.php?title=Automatically_launch_your_application pour créer un script de lancement automatique nommé S99minisever.*

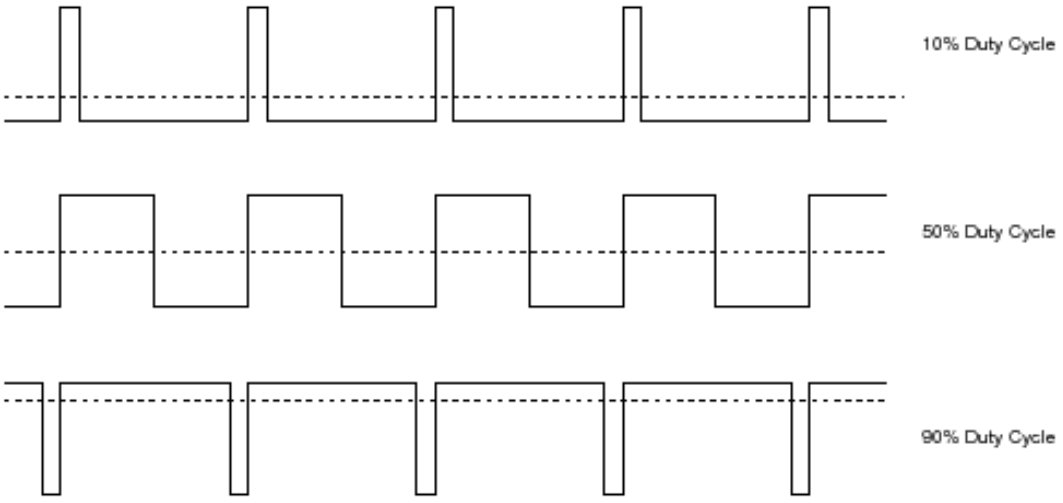
Question 41 (2 min) : *Rebootez l'APF28 et observez la LED s'allumer.*

Question 42 (5 min) : *Utilisez le client pour envoyer des messages au serveur. Observez le bon fonctionnement en regardant si le miniserver log bien les messages recu dans le répertoire /var/log.*

Question 43 (2 min) : *Commitez les modifications apportées (création de led_server, script d'init, …).*

4.5. Étape 4 (1h30) : PWM

Le PWM, ou Pulse Mith Modulation, permet de simuler un signal continu en moyenne grâce à des signaux discrets (de valeur constante fixe). En image, cela donne ceci :



Deux paramètres permettent donc de gérer la tension moyenne souhaitée :

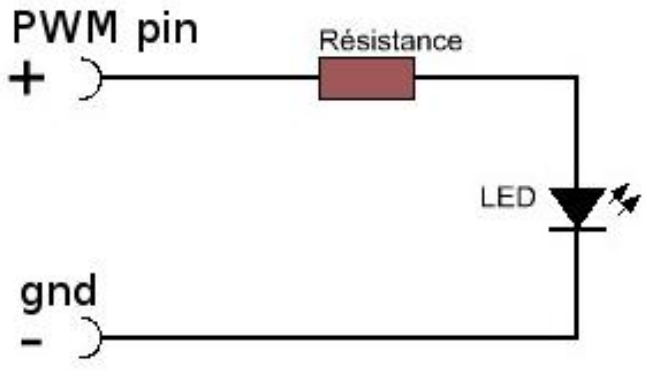
- le **duty_cycle**
- la **period**

Dans l'espace utilisateur, la logique d'utilisation est la même que pour les GPIO : on passe par des fichiers virtuels de /sys/class pour le contrôle.

La carte APF28 possède un connecteur J10 où se trouve une pin PWM.

Question 45 (10 min) : *Fouillez la datasheet de l'APF28dev pour déterminer le numéro de la pin offrant du PWM ainsi qu'une pin de masse sur J10.*

Question 46 (10 min) : *Réalisez le montage ci-dessous. N'oubliez pas que la broche la plus courte de la LED est le -.*



Ensuite, placez vous dans le répertoire /sys/class/pwm et réalisez les commandes suivantes afin d'avoir accès au PWM de la pin :

```
> cd /sys/class/pwm/
> ls
pwmchip0
> export PWM_CHIP=pwmchip0
> export PWM=4
> cd pwmchip0/
> ls
device     export     npwm       power      subsystem  uevent     unexport
> echo $PWM > /sys/class/pwm/$PWM_CHIP/export
> ls
device     npwm       pwm4       uevent
export     power      subsystem  unexport
```

Afin d'activer la pin, réalisez la commande suivante :

```
> echo 1 > /sys/class/pwm/$PWM_CHIP/pwm$PWM/enable
```

Maintenant, vous pouvez changer la valeur de la periode ainsi que la valeur du duty cycle comme ci-dessous :

```
> echo 50000 > /sys/class/pwm/$PWM_CHIP/pwm$PWM/duty_cycle
> echo 100000 > /sys/class/pwm/$PWM_CHIP/pwm$PWM/period
```

Faite différents tests pour faire varier l'intensité de la LED lumineuse.

Question 45 (1h00) : *Modifiez le code de led_server et du client afin que ce dernier puisse envoyer des commandes au serveur par réseau pour modifier l'intensité de la LED de l'APF28.*