Universidade de Brasília
Faculdade Gama
Curso – Sistemas Embarcados.
Professor: Renato Coral Sampaio
Data: __/__/____
Aluno: _____**Matrícula:**_____

Laboratório – Instalação do Framework Xenomai 3

1. Objetivo do experimento

Este experimento tem por objetivo demonstrar passo-a-passo os procedimentos básicos necessários para se desenvolver para sistemas embarcados

2. Descrição do experimento

O experimento será realizado em quatro etapas:

a) Preparação da Imagem do SDCard com o Linux (Raspbian);
b) Preparação e compilação da Toolchain para Cross Compilação no ARM;
c) Preparação e instalação do Kernel;
d) Preparando o suporte ao User Space.

A primeira etapa tem por objetivo promover a preparação e instalação de um novo kernel. Em outras palavras, um kernel linux é configurado e modificado para oferecer suporte a funcionalidades de RTLinux. Desse modo, os arquivos de código fonte do Xenomai, e do Kernel do linux devem estar disponíveis. Nesse sentido, o Kernel Space do framework Xenomai deve ser instalado no kernel linux, assim como o micro-kernel através do ADEOS. Também é fundamental escolher o patch (remendo) disponibilizado pelo ADEOS compatível com a versão do kernel que se deseja alterar. Uma escolha errada, em geral, leva a uma falha na instalação. Uma vez que o kernel é alterado e devidamente configurado, procede-se com a compilação e instalação do mesmo.

Após a configuração e instalação do novo kernel linux, deve-se instalar o suporte ao User Space disponibilizado pelo Xenomai. Observa-se que essa etapa não impõe qualquer alteração ao kernel recentemente instalado, mas apenas instala as bibliotecas e aplicativos para se utilizar as funções disponibilizadas pela API do Xenomai. Em geral, executáveis e bibliotecas que encapsulam as funcionalidades do User Space é disponibilizado no diretório /usr/xenomai.

Este tutorial está focado no Kernel 4.1 do Linux com o Xenomai versão 3.0.2 para serem instalados no Raspberry Pi 2 ou 3.

Todas as etapas foram testadas no ambiente Linux Ubuntu 14.04.

A) Preparação da Imagem do SDCard com o Linux (Raspbian Jessie)

1. Baixar a imagem do Linux Raspbian Jessie versão 4.1.x: http://downloads.raspberrypi.org/raspbian/images/raspbian-2016-05-13/2016-05-10-raspbian-jessie.zip
2. Gravar a Imagem no SDCard (https://www.raspberrypi.org/documentation/installation/installing-images/linux.md). Antes de inserir o SDCard no liste os dispositivos de bloco:

```
$ lsblk
```

3. Insira o SDCard e novamente liste os dispositivos de bloco

```
$ lsblk
```

4. Identificar o device (Ex: /dev/sdb ou /dev/sdc) e verificar se existe alguma partição do dispositivo que está montada (Ex: /dev/sdbN onde N é o número da partição: 1, 2, 3…). Caso exista, realize o umount desta partição:

```
$ sudo umount /dev/sdbN
```

5. Após desmontar todas as partições do SDCard, copiar a imagem do SO para o cartão.

```
$ sudo dd bs=4M if=/caminho_da_imagem.img of=/dev/sdX
```

Onde if é o caminho do arquivo de entrada (a imagem do SO baixada no item 1), of é o caminho de destino, e sdX é o device onde X é a letra do device (a, b, c, d, …)

6. Ao final executar um sync para sincronizar os dados antes de ejetar o SDCard

```
$ sync
```

7. Testar o SDCard e verificar o correto funcionamento do Raspberry Pi

B) Preparação e compilação da Toolchain para Cross Compilação no ARM

Ref: http://www.blaess.fr/christophe/2015/12/08/creation-dun-systeme-complet-avec-buildroot-2015-11/

1. Criar as pastas de trabalho

```
$ cd ~
$ mkdir br-tree
$ cd br-tree
$ mkdir -p board/raspberrypi2/
```

2. Fazer download dos códigos fonte da toolchain

```
$ wget http://www.buildroot.org/downloads/buildroot-2016.02.tar.bz2
$ tar xjf buildroot-2016.02.tar.bz2
$ cd buildroot-2016.02/
```

3. Preparando para a compilação da Toolchain

```
$ make raspberrypi2_defconfig
$ make menuconfig
```

Na interface de configurações faça as seguintes alterações:

Menu **`Build options`**:

`Download dir`: modificar o valor para **`$(TOPDIR)/../dl`**

`Host dir`: modificar o valor para **`$(TOPDIR)/../board/raspberrypi2/cross`**

Menu **`Toolchain`** :

`C library` : definir como **`glibc`**

Menu **`System configuration`** :

`Init system`: Este valor está em BusyBox e deve ser alterado para **`none`**.

Menu **`Kernel`** :

`Linux Kernel`: desabilite esta opção. Defina o valor para **`[].`**

Menu **`Target Packages`** :

`BusyBox`: é o único pacote presente. Desabilite esta opção. Defina o valor como **`[]`**.

Menu **`Filesystem image`** :

`tar the root filesystem` : Desabilite esta opção. Defina o valor como **`[]`**.

4. Salve a configuração atual e em seguida copie o rquivo de configurações .config para outra pasta por segurança e inicie a compilação:

```
$ cp .config ../board/raspberrypi2/buildroot-01.cfg
$ make
```

O resultado deve ser parecido com os itens abaixo:

```
arm-buildroot-linux-gnueabihf-addr2line          arm-linux-c++
arm-buildroot-linux-gnueabihf-ar                 arm-linux-c++.br_real
arm-buildroot-linux-gnueabihf-as                 arm-linux-cc
arm-buildroot-linux-gnueabihf-c++                arm-linux-cc.br_real
arm-buildroot-linux-gnueabihf-c++.br_real        arm-linux-c++filt
arm-buildroot-linux-gnueabihf-cc                 arm-linux-cpp
arm-buildroot-linux-gnueabihf-cc.br_real         arm-linux-cpp.br_real
arm-buildroot-linux-gnueabihf-c++filt            arm-linux-elfedit
arm-buildroot-linux-gnueabihf-cpp                arm-linux-g++
arm-buildroot-linux-gnueabihf-cpp.br_real        arm-linux-g++.br_real
arm-buildroot-linux-gnueabihf-elfedit            arm-linux-gcc
arm-buildroot-linux-gnueabihf-g++                arm-linux-gcc-4.9.3
arm-buildroot-linux-gnueabihf-g++.br_real        arm-linux-gcc-4.9.3.br_real
arm-buildroot-linux-gnueabihf-gcc                arm-linux-gcc-ar
arm-buildroot-linux-gnueabihf-gcc-4.9.3          arm-linux-gcc.br_real
arm-buildroot-linux-gnueabihf-gcc-4.9.3.br_real  arm-linux-gcc-nm
arm-buildroot-linux-gnueabihf-gcc-ar             arm-linux-gcc-ranlib
arm-buildroot-linux-gnueabihf-gcc.br_real        arm-linux-gcov
arm-buildroot-linux-gnueabihf-gcc-nm             arm-linux-gprof
arm-buildroot-linux-gnueabihf-gcc-ranlib         arm-linux-ld
arm-buildroot-linux-gnueabihf-gcov               arm-linux-ld.bfd
arm-buildroot-linux-gnueabihf-gprof              arm-linux-nm
arm-buildroot-linux-gnueabihf-ld                 arm-linux-objcopy
arm-buildroot-linux-gnueabihf-ld.bfd             arm-linux-objdump
arm-buildroot-linux-gnueabihf-nm                 arm-linux-ranlib
arm-buildroot-linux-gnueabihf-objcopy            arm-linux-readelf
arm-buildroot-linux-gnueabihf-objdump            arm-linux-size
arm-buildroot-linux-gnueabihf-ranlib             arm-linux-strings
arm-buildroot-linux-gnueabihf-readelf            arm-linux-strip
arm-buildroot-linux-gnueabihf-size               gawk
arm-buildroot-linux-gnueabihf-strings            igawk
arm-buildroot-linux-gnueabihf-strip              m4
arm-linux-addr2line                              mkknlimg
arm-linux-ar                                     toolchain-wrapper
arm-linux-as
```

5. Por fim, adicione a toolchain ao PATH

```
$ PATH=$PATH:~/br-tree/board/raspberrypi2/cross/usr/bin/
```

C) Preparação e instalação do Kernel

http://www.blaess.fr/christophe/2016/05/22/xenomai-3-sur-raspberry-pi-2/#comment-172188

1. Baixar a imagem Xenomai versão 3.0.2

```
$ cd ~
$ mkdir xenomai
$ wget http://xenomai.org/downloads/xenomai/stable/xenomai-3.0.2.tar.bz2
$ tar xjf xenomai-3.0.2.tar.bz2
```

2. Verificar a existência dos patches

```
$ ls /xenomai-3.0.2/kernel/cobalt/arch/arm/patches
```

3. Fazer download do código fonte do Linux para Raspberry Pi:

```
$ git clone --depth=1 -b rpi-4.1.y git://github.com/raspberrypi/linux.git linux
$ cd linux
```

4. Verificar a aplicação do Patch (usando o dry-run)

```
$ patch—dry-run -p1 < ../xenomai-3.0.2/kernel/cobalt/arch/arm/patches/ipipe-core-4.1.18-arm-4.patch
```

5. Aplicar o patch

```
$ cd ..
$ xenomai-3.0.2/scripts/prepare-kernel.sh --linux=linux/ --arch=arm
--ipipe=xenomai-3.0.2/kernel/cobalt/arch/arm/patches/ipipe-core-4.1.18-arm-4.patch
```

6. Fazer download do Patch adicional e testar sua aplicação

```
$ wget http://www.blaess.fr/christophe/files/article-2016-05-22/patch-xenomai-3-on-bcm-2709.patch
$ cd linux
$ patch -p1 < ../patch-xenomai-3-on-bcm-2709.patch --dry-run
```

7. Aplicar o patch adicional

```
$ patch -p1 < ../patch-xenomai-3-on-bcm-2709.patch
```

8. Configure o Kernel do Linux

```
$ make ARCH=arm bcm2709_defconfig
$ make ARCH=arm menuconfig
```

Na interface de configurações faça as seguintes alterações:

```
CPU Power Management --->
  CPU Frequency scaling --->
     [ ] CPU Frequency Scaling
Kernel Freatures --->
  [ ] Contiguous Memory Allocator
Kernel Freatures --->
  [ ] Allow for memory compaction
Kernel Hacking --->
  [ ] KGDB: kernel debugger --->
Boot options --->
  Kernel command line like (use bootloader kernel arguments if available) —>
    (X) Extend bootloader kernel arguments
```

9. Salve as configurações e inicie a compilação

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

10. Após o término da compilação do Kernel, copie os arquivos para a a partição BOOT do SDCard

```
$ cp  arch/arm/boot/zImage  /media/$USER/BOOT/
$ cp  arch/arm/boot/dts/bcm2709-rpi-2-b.dtb  /media/$USER/BOOT/
```

11. Editar o arquivo config.txt na partição /boot do SDCard e adicionar a linha abaixo:

```
kernel=zImage
```

D) Preparando o suporte ao User Space

http://www.blaess.fr/christophe/2016/05/22/xenomai-3-sur-raspberry-pi-2/#comment-172188

1. Entrar na pasta do Xenomai, configurar a compilar o Xenomai.

```
$ cd ~
$ cd xenomai/xenomai-3.0.2
$ ./scripts/bootstrap
$ ./configure --host=arm-linux --enable-smp
$ make
```

2. Após a compilação, configurar os arquivos e copiá-los para a partição /root do SDCard

```
$ make  DESTDIR=$(pwd)/target  install
$ sudo cp -a target/* /media/$USER/root
$ umount /media/$USER/*
```

3. Inserir o SDCard no Raspberry Pi e iniciar a placa

4. Para verificar o funcionamento do Xenomai execute o teste de latência

```
$ cd /usr/xenomai/bin
$ sudo ./latency -p0 -t0
```

Observações: Esta versão do Xenomai foi testada nas placas Raspberry Pi 2 e 3 e parece estar funcionando corretamente, exceto pela falta de suporte às portas USB.