

Configuração de Rede sem Fio e Segurança no Sistema Operacional Android

Daniel Fernando Scota, Gil Eduardo de Andrade, Rafael da Costa Xavier

Especialização em Redes e Segurança de Sistemas - 2008/2
Pontifícia Universidade Católica do Paraná

Curitiba, Abril de 2010

Resumo

Ainda considerado uma novidade no mundo da tecnologia e comunicação, o Android, sistema operacional assinado pela gigante Google Inc., vem ganhando a cada dia mais espaço no mercado e conquistando novos adeptos. Este artigo descreverá a configuração dos serviços de rede necessários para que o sistema Android, executado em uma placa BeagleBoard, conecte-se à rede Wi-Fi e navegue na internet, como interface de rede, DHCP e DNS. Feita a configuração, serão simulados alguns códigos maliciosos que, inseridos no Kernel do sistema antes de sua compilação, demonstrarão possíveis instabilidades do S.O. Neste mesmo artigo, será apresentado uma maneira de utilizar o sistema Android em uma máquina virtual.

1. O Sistema Operacional Android

O Android é um sistema operacional *open-source*, baseado em Linux, destinado a equipamentos móveis. Desenvolvido inicialmente pela Google e posteriormente pela Open Handset Alliance, o sistema possui uma rica e atraente interface gráfica, que apresenta uma grande diversidade de aplicações, navegador de internet, banco de dados integrado, jogos, integração com outros sistemas, como GPS, conectividades diversas, como Bluetooth, EDGE, 3G e Wi-Fi e entre outras características. A plataforma de desenvolvimento do Android permite o desenvolvimento e integração de aplicações na linguagem de programação Java, controlando os dispositivos através de bibliotecas desenvolvidas pela Google.

2. Falhas de Segurança

Em se tratando de sistemas digitais, pode-se dizer que todos os softwares podem apresentar falhas de segurança. Estas falhas, presentes na programação do sistema, podem permitir que vírus de computador ou indivíduos mal-intencionados, invadam o sistema e comprometam sua integridade. Estas vulnerabilidades são descobertas pelo próprio desenvolvedor do sistema, por usuários, ou, em um pior cenário, por criminosos. Em qualquer destes casos, o desenvolvedor precisa correr contra o tempo para corrigir estas falhas e disponibilizar uma atualização com a vulnerabilidade eliminada.

Um sistema seguro é aquele que fornece informações íntegras somente a usuários autenticados e autorizados, no momento em que elas são pedidas, através de requisições válidas e identificadas, não permitindo que estas informações sejam recebidas, observadas ou alteradas por terceiros não autorizados.

2.1 Conceitos básicos para segurança de dados

Alguns conceitos básicos devem ser levados em considerações para identificar e tratar possíveis falhas de segurança, como:

- Confidenciabilidade: Garante que as informações armazenadas em um sistema de computação ou transmitidas através de uma rede de computadores, sejam acessadas ou manipuladas somente pelos usuários devidamente autorizados.
- Integridade: Garante que a informação processada ou transmitida, chegue ao seu destino exatamente da mesma forma em que partiu da origem.
- Disponibilidade: Garante que o sistema de computação continue operando sem degradação de acesso e provê recursos aos usuários autorizados quando necessário.
- Legitimidade: Garante que os recursos não sejam utilizados por pessoas não autorizadas ou de forma não autorizada.

2.2 Meios de Ataque

Em se tratando de sistemas computacionais, existem diversos meios conhecidos de tornar um sistema vulnerável a ataques. Entre eles, destacam-se:

- Vírus: Código infiltrado em programas hospedeiros que se espalham infectando todo o sistema, sempre buscando alguma atividade maliciosa. A contaminação pode ocorrer por arquivos infectados em *pen drives*, e-mail, rede local ou também por utilizar um sistema operacional desatualizado, com falhas de segurança não corrigidas.
- Verme (Worm): É um processo que possui um mecanismo através do qual pode se multiplicar, infectando outros sistemas e consumindo os recursos daquele que reside. Diferente de um vírus, que infecta um programa e necessita deste programa hospedeiro para se propagar, o worm é um programa completo e não precisa de outro para se propagar.
- Cavalo de Troia (Trojan Horse): É um vírus de computador que, geralmente, modifica um programa para que ele realize atividades maliciosas em adição ao seu comportamento normal e que induz o usuário a executar esta versão modificada do programa, pensando estar executando a versão original. É o vírus mais encontrado em computadores domésticos.
- Portas do Fundo (Backdoor / Trapdoor): É considerado como uma falha de segurança existente em um programa de computador ou sistema operacional, esta ocasionada por falha na programação ou até mesmo deixada pelo projetista do software. Uma possibilidade complicada de detectar esta falha é quando esta é incluída pelo compilador.
- Ataque de negação de serviço (DoS - Denial of Service): É um tipo de ataque que procura tornar os recursos de um sistema indisponíveis para seus usuários, negando seus serviços. Neste ataque, são usadas técnicas para sobrecarregar uma rede, tornando-a inacessível, como, derrubar conexões entre computadores ou outros equipamentos de rede e realizar inúmeras requisições a um sistema até que este deixe de aceitar novas conexões ou torne-se, praticamente, inutilizável (sistema lento).

2.3 Falhas de segurança identificadas no Android

Um grupo de pesquisadores da *Universidade de Ben-Gurion*, em Israel, divulgou os principais grupos de vulnerabilidades e riscos do Android, que comprometem a disponibilidade do dispositivo, sua confidencialidade e sua integridade, sendo estes, pontos fundamentais para garantir a segurança de qualquer sistema operacional.

Alguns destes danos referem-se às permissões concedidas a uma aplicação instalada ou, através de uma aplicação que explore as vulnerabilidades do Linux e das bibliotecas de

sistemas. Outro grupo de risco, está nos arquivos contidos no cartão de memória *SD*, que não são protegidas por nenhum mecanismo de controle de acesso e, nas comunicações sem fio, que podem ser pirateadas (*eavesdropped*) à distância.

Outro risco é o comprometimento via rede, pois os dispositivos que rodam Android podem ser atacados por outros dispositivos, computadores ou redes, efetuando um escaneamento das portas, ou através de caminhos abertos em serviços de SMS/MMS ou de e-mail. Um deles se encontra na manipulação de mensagens de texto, onde foi possível enviar requisições *WAP* mal-formadas que causam um erro *Java (Array Index Out Of Bounds Exception)*, fazendo com que o sistema reinicie sem avisos, causando perda temporária de conectividade e, portanto, negação de serviço.

Outra falha conhecida, esta presente no *firmware RC29* e anteriores do G1, o Google Phone e iPhone Killer, caracteriza-se por interpretar tudo que você digita no telefone como um “comando”, em qualquer área onde possibilite a entrada de texto. Por exemplo, ao abrir o navegador, ou uma aplicação de *SMS*, e digitar o termo “*reboot*” e teclar “*enter*”, o sistema era reiniciado. Para este problema o Google liberou o *RC30*, corrigindo o problema.

E, por fim, outro problema identificada estava relacionado a diversas anormalidades na *API Dalvik*, que é uma máquina virtual presente no Android. Uma aplicação maliciosa pode ser desenvolvida de tal forma que, caso seja executada pelo usuário, uma função vulnerável da API pode ser chamada, provocando, por exemplo, a reinicialização do sistema.

3. O Sistema Operacional Google e a Plataforma Android

Conforme já descrito no início deste artigo, O Google Android, plataforma *open-source* baseada no Linux, é utilizado, principalmente, em dispositivos móveis como *Palms*, *Smartphones*, Celulares e, recentemente, *Netbooks*. Possui nativamente a integração com várias ferramentas Google, como o *GMail* e *Google Maps*. O Android é considerado como a primeira plataforma móvel completa, aberta e livre.

3.1 A Estrutura de Diretórios do Android (Código Fonte)

A figura abaixo apresenta a estrutura de diretórios do código-fonte do Sistema Operacional Android, já portado para plataforma *ARM*, processador *OMAP* da *Texas Instrument*, fabricante da placa *BeagleBoard*, utilizada neste projeto.

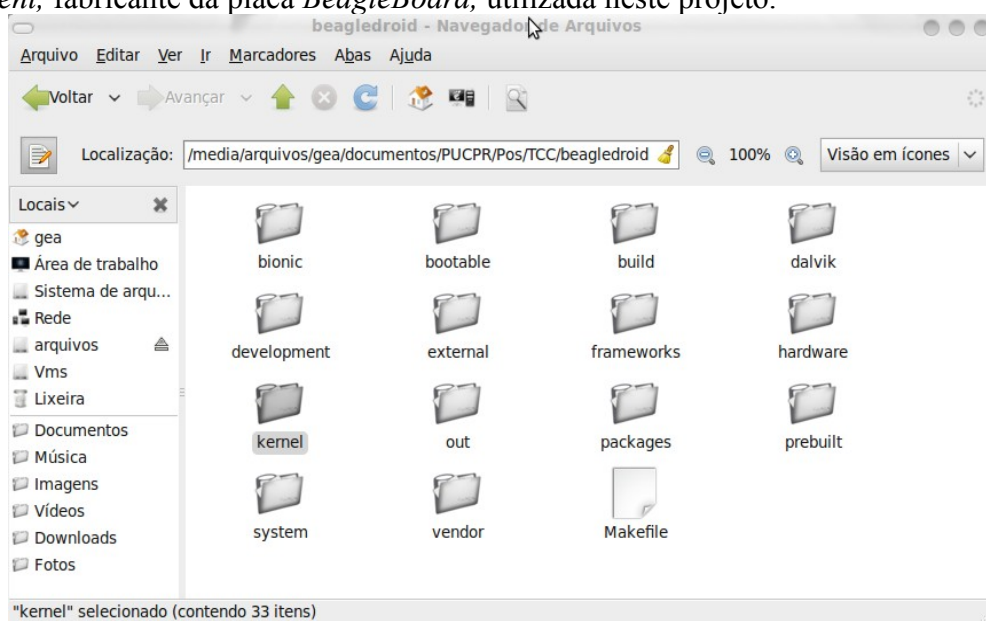


Figura 01: Estrutura de Diretórios do Android

Para configuração do *Wi-Fi* no Android, apenas alguns diretórios serão modificados e comentados com maiores detalhes, porém, uma breve explicação de todos é apresentada logo abaixo:

- Diretório “*bionic*”: contém bibliotecas (*library*) utilizadas pelo sistema operacional.
- Diretório “*bootable*”: contém as ferramentas necessárias para inicializar o sistema operacional (*boot*), através de diversas interfaces como *USB*, memória *Flash*, etc.
- Diretório “*build*”: permite a produção de configurações de compilação do sistema, ou seja a seleção de módulos a serem incluídos ou excluídos do sistema pós-compilado.
- Diretório “*dalvik*”: refere-se a máquina virtual responsável pela execução do sistema operacional Android.
- Diretório “*development*”: contém ferramentas desenvolvidas e integradas ao Android.
- Diretório “*external*”: contém programas externos ao Android, implementados em C, que rodam a nível de kernel (Linux) e que, pós compilados, podem se tornar bibliotecas compartilhadas (*shared library*) ou comandos (executáveis), como por exemplo o comando *ping*.
- Diretório “*frameworks*”: contém a implementação da interface do Android e os códigos-fonte do núcleo do sistema, como sua api e aplicativos.
- Diretório “*hardware*”: contém as bibliotecas (*library*) responsáveis por interagir com dispositivos vitais como *GPS* e o próprio *Wi-Fi* a ser configurado.
- Diretório “*kernel*”: contém o código fonte do sistema operacional Linux, ou seja drivers de dispositivos, bibliotecas do sistema entre outros. Esse diretório será melhor detalhado posteriormente
- Diretório “*out*”: este diretório é criado pós compilação, guardando os arquivos gerados após esse processo, necessários para execução do Android já embarcado.
- Diretório “*packages*”: contém os programas nativos do Android, como os softwares de navegação (*browser*), calculadora (*calculator*), e-mail etc.
- Diretório “*system*”: contém os códigos referentes ao processo de inicialização do sistema. Nele, por exemplo, é possível modificarmos o processo de *boot* do Android, bem como o nome inicial do sistema, carregamento de drivers, dispositivos e etc.
- Diretório “*vendor*”: contém os arquivos (*scripts*) necessários para a compilação do *kernel* do sistema (compilação do Linux) e criação da imagem necessária para inicialização do Android (*boot*).

3.2 O Kernel do Android

O *kernel* do sistema operacional Android, portado para arquitetura *OMAP* da *Texas Instrument*, é baseado no Linux *ARM*. A versão utilizada no desenvolvimento deste trabalho é a 2.6.29.

A ARM é fabricante de microprocessadores e microcontroladores usados em eletrônicos embarcados e móveis. Faz parte da Linux Foundation e, com intuito de inovar os sistemas com Linux, contribui com o Kernel Linux há algum tempo.

A figura abaixo apresenta a estrutura de diretórios do Kernel Linux ARM 2.6.29:

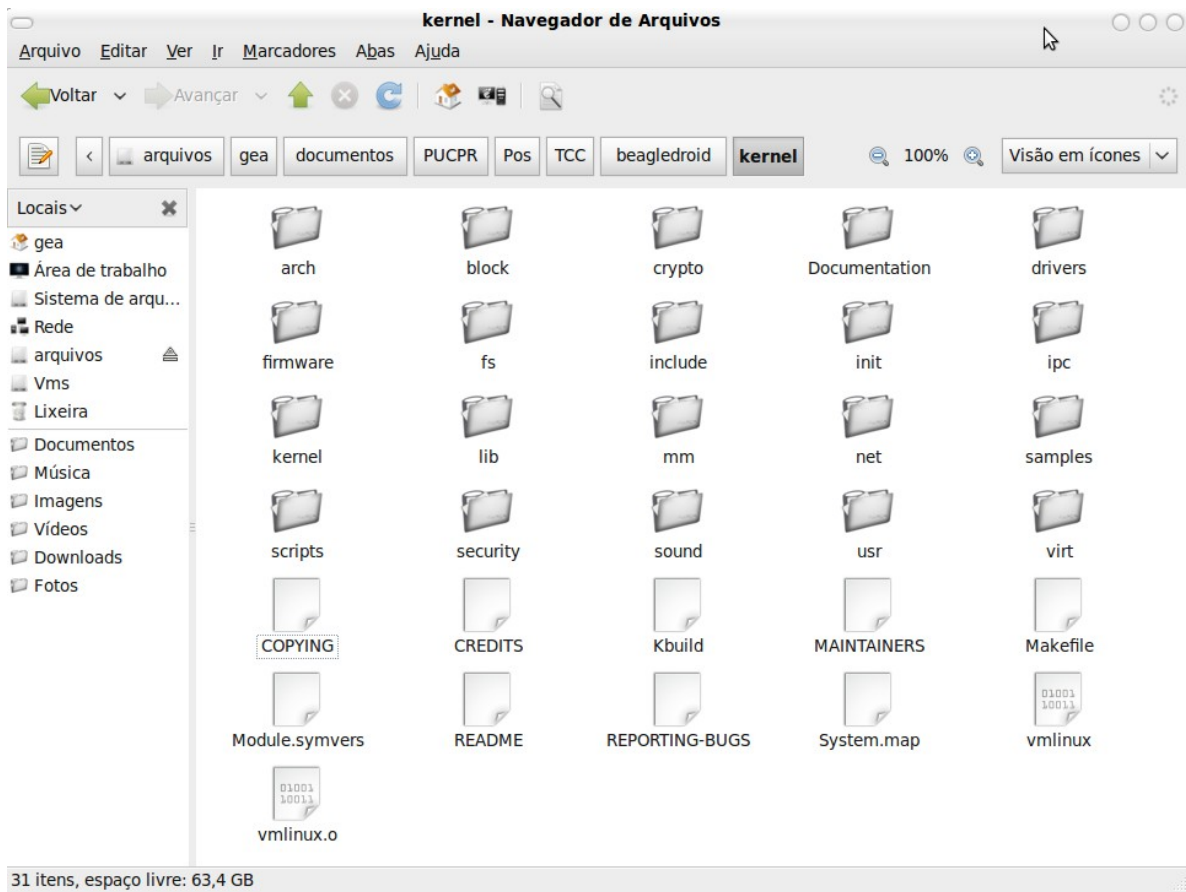


Figura 02: Estrutura de Diretórios do Kernel Linux

Nota: O intuito deste artigo não é descrever em detalhes o Linux ARM, e, por esta razão, descreveremos apenas os diretórios e arquivos necessários para a configuração da rede Wi-Fi no Google Android, portado para a BeagleBoard.

3.2.1 Diretórios de configuração

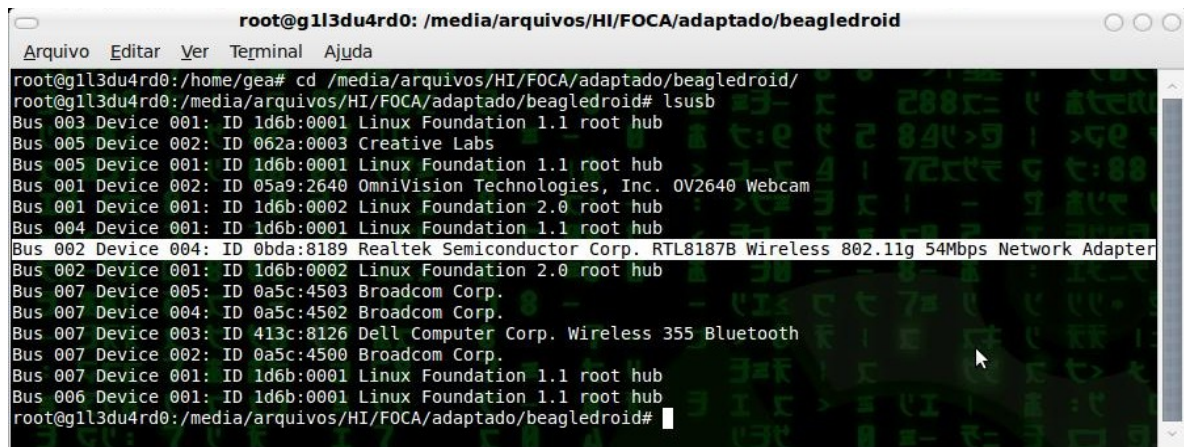
- **Configs:** O diretório *configs*, que encontra-se no caminho */kernel/arch/arm/configs*, possui os arquivos responsáveis pela configuração do Linux ARM, para vários modelos diferentes de placas que utilizam a arquitetura ARM. Em nosso caso, onde utilizamos a placa BeagleBoard, apenas o arquivo “omap3_beagle_android_defconfig” será utilizado para configuração dos drivers e módulos a serem incluídos no kernel durante sua compilação.

- **Drivers:** O diretório *drivers*, que encontra-se no caminho */kernel/drivers*, contém os drivers a serem incluídos no kernel do Linux durante sua compilação. É neste importante diretório que encontraremos o *driver* correto para o adaptador de rede que utilizaremos na conexão e transmissão de dados via rede Wi-Fi.

Dentro do diretório de drivers de dispositivos podemos encontrar o arquivo *Makefile*, que, em seu conteúdo, especifica qual linha devemos adicionar no arquivo *omap3_beagle_android_defconfig* para que seja adicionado o driver requerido durante a compilação. O arquivo *Android.mk* também é importante para definirmos quais módulos deverão ser adicionados ao kernel, visto que os drivers podem possuir alguma dependência.

O adaptador escolhido para contemplar o trabalho foi o *Encore 802.11g Wireless USB Adapter*, que possui como chipset o componente *RTL8187B*. Tal informação foi obtida através do comando “*lsusb*”, no terminal do sistema operacional Linux, rodando num

microcomputador com Sistema Operacional Linux Ubuntu, com adaptador conectado em uma de suas portas USB, como ilustrado na figura abaixo:



```
root@gl13du4rd0: /media/arquivos/HI/FOCA/adaptado/beagledroid
Arquivo  Editar  Ver  Terminal  Ajuda
root@gl13du4rd0: /home/gea# cd /media/arquivos/HI/FOCA/adaptado/beagledroid/
root@gl13du4rd0: /media/arquivos/HI/FOCA/adaptado/beagledroid# lsusb
Bus 003 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 005 Device 002: ID 062a:0003 Creative Labs
Bus 005 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 001 Device 002: ID 05a9:2640 OmniVision Technologies, Inc. OV2640 Webcam
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 004 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 002 Device 004: ID 0bda:8189 Realtek Semiconductor Corp. RTL8187B Wireless 802.11g 54Mbps Network Adapter
Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 007 Device 005: ID 0a5c:4503 Broadcom Corp.
Bus 007 Device 004: ID 0a5c:4502 Broadcom Corp.
Bus 007 Device 003: ID 413c:8126 Dell Computer Corp. Wireless 355 Bluetooth
Bus 007 Device 002: ID 0a5c:4500 Broadcom Corp.
Bus 007 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 006 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
root@gl13du4rd0: /media/arquivos/HI/FOCA/adaptado/beagledroid#
```

Figura 03: Comando “lsusb”, no terminal do Linux.

Identificado o *chipset* de nosso adaptador *USB Wi-Fi*, ilustraremos como adicionar seu driver a compilação do Kernel.

O diretório *drivers*, localizado, a partir do diretório principal, em */kernel/drivers/net*, contém os drivers já disponíveis no Linux ARM para dispositivos de rede. Dentro deste, encontraremos o diretório *.../net/wireless*, que contém todos os drivers disponíveis para adaptadores de rede sem fio. O nosso driver (*RTL8187B*) já está presente no diretório *.../net/wireless/rtl8180*.

Como comentado acima, dentro do arquivo *Makefile* identificaremos qual “linha” deverá ser adicionada ao arquivo *omap3_beagle_android_defconfig*. Nele encontraremos o conteúdo apresentado abaixo:



```
Makefile (arquivos) - gedit
Arquivo  Editar  Ver  Pesquisar  Ferramentas  Documentos  Ajuda
Abrir  Salvar  Desfazer
Makefile
1 rtl8180-objs := rtl8180_dev.o rtl8180_rtl8225.o rtl8180_sa2400.o rtl8180_max2820.o
  rtl8180_grf5101.o
2 rtl8187-objs := rtl8187_dev.o rtl8187_rtl8225.o
3
4 obj-$(CONFIG_RTL8180) += rtl8180.o
5 obj-$(CONFIG_RTL8187) += rtl8187.o
6
7
Makefile  Largura das tabulações: 8  Lin 4, Col 37  INS
```

Figura 04: Arquivo Makefile: Driver do chipset RTL8187B - Kernel Linux.

A linha *obj-\$(CONFIG_RTL8187) += rtl8187.o*, indica-nos que o trecho *CONFIG_RTL8187=y* deve ser adicionado ao arquivo *omap3_beagle_android_defconfig*, que é lido durante a compilação para identificar os módulos que devem ser adicionados ao kernel. Essa adição garante que o Android, pós-compilação, contera o driver necessário para controlar a comunicação entre o Sistema Operacional e o Adaptador de rede Wi-Fi da Encore. O arquivo de configuração será apresentado com maiores detalhes no decorrer deste artigo.

- **External:** O diretório *external*, que encontra-se no diretório raiz */external*, possibilita a configuração dos módulos necessários ao Sistema Operacional para execução da conexão e troca de dados da rede. Através dele é possível adicionar programas não disponíveis como padrão no Android, mas que sejam necessários para execução de tarefas indispensáveis.

Um exemplo para a placa BeagleBoard é o driver de som. O padrão usado no Linux é

o *ALSA* (*Advanced Linux Sound Architecture*), que não vem como padrão no Android, portado para arquitetura ARM. Para que seja utilizado tal driver, é necessário adicionar seu código fonte e bibliotecas necessárias ao diretório *external* e configurado para compilar de maneira correta. Para tanto, devemos observar a pasta *wpa_supplicant*, localizada em */external/wpa_supplicant*, apresentada abaixo:

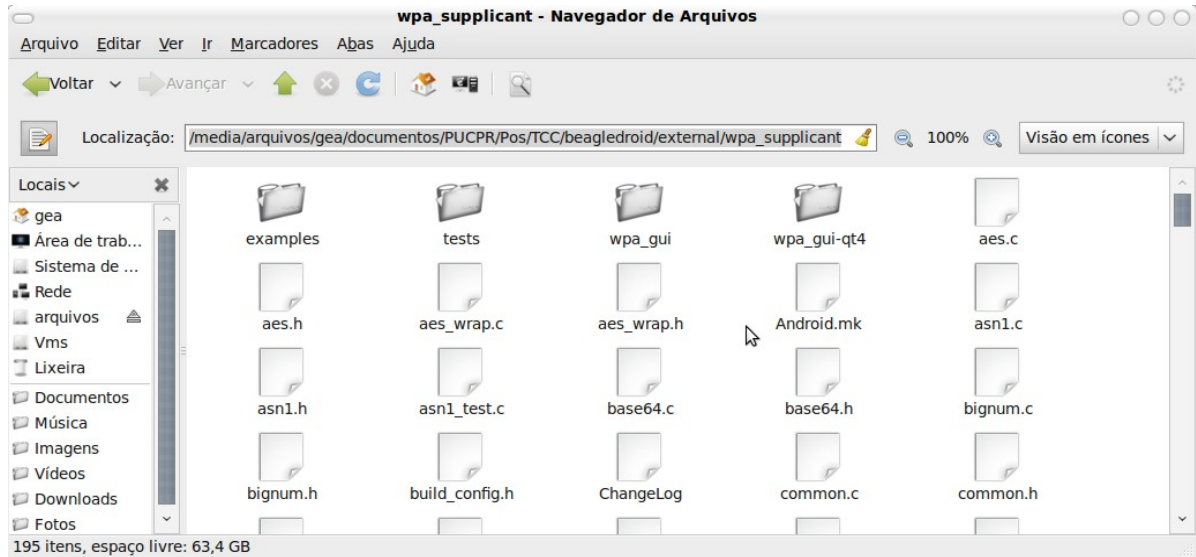


Figura 05: Módulo *wpa_supplicant* – configuração rede Wi-Fi no Android.

Dentro deste diretório será necessário efetuar a configuração de alguns módulos, descritos com maiores detalhes no decorrer deste artigo. A configuração é efetuada dentro do arquivo oculto *“.config”*, localizado em */external/wpa_supplicant/.config*. Observe o conteúdo do arquivo *.config* logo abaixo:

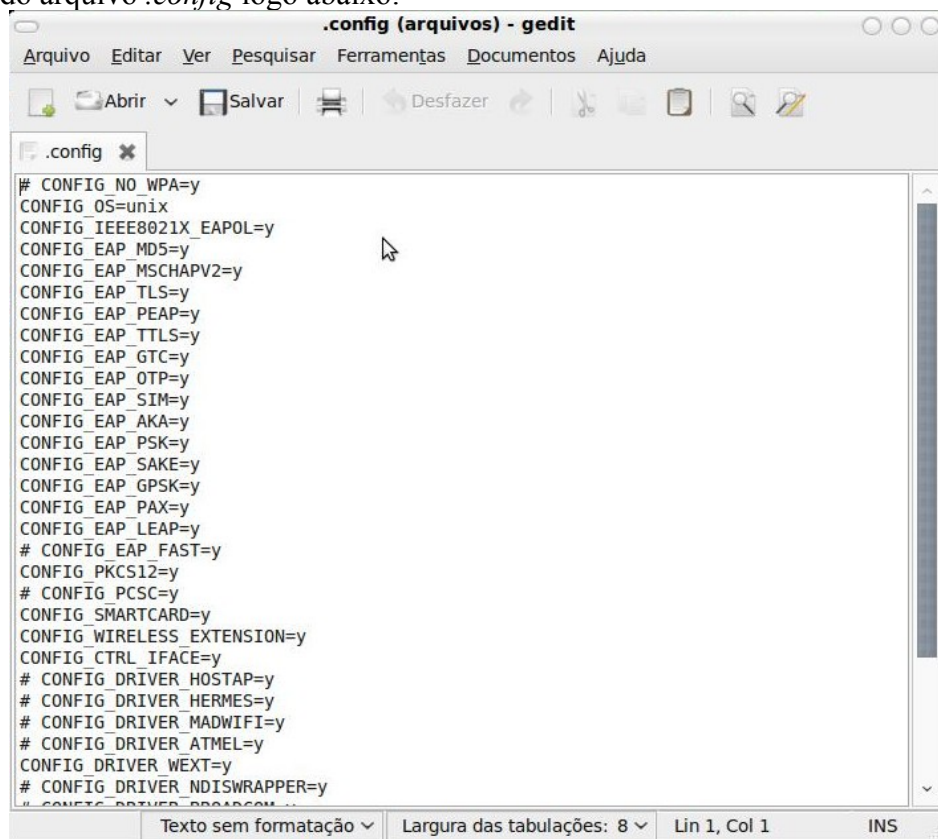


Figura 06: Arquivo de configuração do módulo *wpa_supplicant*

- **Build:** No diretório *build* deverá ser feita a configuração dos dispositivos a serem utilizados pela placa BeagleBoard. Dentro dele, no diretório *generic*, encontrado em */build/target/board/generic/*, configuraremos o arquivo *BoardConfig.mk*. É este arquivo que permite indicar a compilação a utilização do hardware Wi-Fi em conjunto com a *mainboard*, visto que a versão do Android em questão, não vem com suporte a dispositivos wireless ativado por padrão. Estas configurações serão apresentadas com maiores detalhes na seção “As Configurações de Rede Wi-Fi Pré-Compilação”. Veja abaixo o conteúdo do arquivo:



Figura 07: Arquivo configuração do hardware Wifi.

3.3 As Configurações de Rede Wi-Fi Pré-Compilação

A configuração Wi-Fi no Android começa a partir de aplicativos *Java* e vai até *driver/firmware*, com a inserção da interface *JNI* e Wi-Fi nativa do componente. As configurações necessárias dentro do Android e kernel Linux são apresentadas abaixo:

- **Arquivo:** */build/target/board/generic/BoardConfig.mk*

*Comandos (linhas) adicionados:

```
HAVE_CUSTOM_WIFI_DRIVER_2 := true
```

- **Arquivo:** */external/wpa_supplicant/.config*

*Comandos (linhas) adicionados:

```
CONFIG_WIRELESS_EXTENSION=y
CONFIG_CTRL_IFACE=y
CONFIG_DRIVER_WEXT=y
```

- **Arquivo:** */external/wpa_supplicant/Android.mk*

*Comandos (linhas) adicionados:

```
WPA_BUILD_SUPPLICANT := true
```

- **Arquivo:** */kernel/arch/arm/configs/omap3_beagle_android_defconfig*

*Comandos (linhas) adicionados:

```
CONFIG_WLAN_PRE80211=y
CONFIG_WLAN_80211=y
CONFIG_IWLWIFI_LEDS=y
CONFIG_RTL8187=y
CONFIG_LIB80211=y
CONFIG_MAC80211=y
CONFIG_EEPROM_93CX6=y
```


Das etapas apresentadas é importante observar uma configuração em específico, o serviço *wpa_supplicant*, que é quem dá o suporte para WPA e WPA2. Independentemente de configurar uma rede wireless usando uma ferramenta específica, como *NetworkManager*, do Mandriva, ou qualquer outra ferramenta de configuração, é o *wpa_supplicant* que faz a configuração na realidade. Para usá-lo é necessário que a placa wireless tenha sido detectada pelo sistema, e, por isso, deve-se habilitar o driver do adaptador USB-Wi-Fi escolhido, (*CONFIG_RTL8187=y*) para que este seja carregado com o Kernel final, durante a compilação.

4. Compilando o Sistema Operacional

Para compilação do sistema foi utilizado um microcomputador com Linux Ubuntu 9.10 instalado, a partir disso, os passos necessários para configuração do Ubuntu antes de iniciarmos a compilação são descritos abaixo:

4.1 Instalação dos pacotes necessários:

A versão da plataforma utilizada para produção deste documento não suporta a última versão do *Java*, a “6”, por isso, é preciso trabalhar com versão “5” para fazer a compilação do Android. Para tanto:

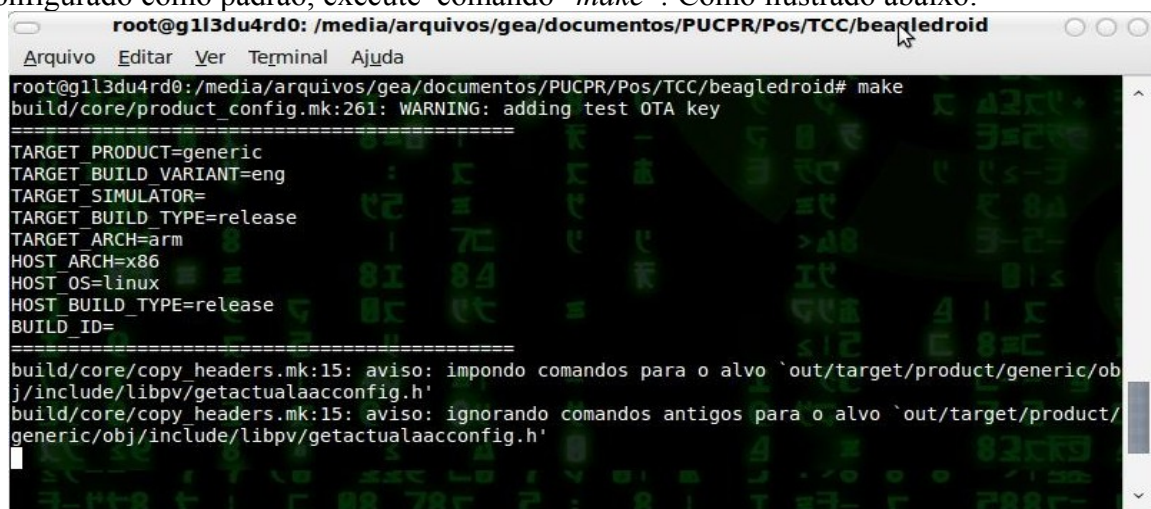
```
#apt-get update
#apt-get dist-upgrade
#apt-get install git-core bison sun-java5-jdk flex g++ zlib1g-dev
#apt-get install libx11-dev libncurses5-dev gperf uboot-mkimage
```

Também é necessário obter uma *toolchain* (caixa de ferramentas) para a compilação do kernel do Android. A etapa de compilação divide-se em duas, pois é necessário compilar o Android (Interface Gráfica do Sistema) e o Kernel (Linux) do Sistema Operacional.

Nota: O toolchain utilizado é o 2007q3, disponível para download em: <http://free-electrons.com/pub/demos/beagleboard/android>, assim como o código fonte inteiro do Android utilizado neste projeto.

4.2 Compilando o Android (Interface)

Acesse o diretório raiz do sistema, onde estão todos os seus diretórios e o arquivo *Makefile*, através do terminal e, com privilégios de *root* (*sudo bash*) e com o *Java 5* configurado como padrão, execute comando “*make*”. Como ilustrado abaixo:



```
root@g113du4rd0: /media/arquivos/gea/documentos/PUCPR/Pos/TCC/beagledroid
Arquivo Editar Ver Terminal Ajuda
root@g113du4rd0: /media/arquivos/gea/documentos/PUCPR/Pos/TCC/beagledroid# make
build/core/product_config.mk:261: WARNING: adding test OTA key
=====
TARGET_PRODUCT=generic
TARGET_BUILD_VARIANT=eng
TARGET_SIMULATOR=
TARGET_BUILD_TYPE=release
TARGET_ARCH=arm
HOST_ARCH=x86
HOST_OS=linux
HOST_BUILD_TYPE=release
BUILD_ID=
=====
build/core/copy_headers.mk:15: aviso: impondo comandos para o alvo `out/target/product/generic/obj
j/include/libpv/getactulaacconfig.h'
build/core/copy_headers.mk:15: aviso: ignorando comandos antigos para o alvo `out/target/product/
generic/obj/include/libpv/getactulaacconfig.h'
```

Figura 08: Compilando Android (Interface)

4.3 Compilando o Kernel do Android (Linux)

Para compilação do Kernel é necessário configurar a variável de ambiente `CC_PATH` utilizada pelo script de compilação. Para tanto, deve-se acessar o caminho do toolchain instalado. Em nosso caso o *toolchain* encontra-se no mesmo diretório dos arquivos do *beagleboard*, que contém todo o código-fonte da plataforma Android, dentro do diretório “*opt*”. Os comandos para configurar a variável e compilar o kernel são:

```
/beagleboard# export
CC_PATH=/media/arquivos/gea/documentos/PUCPR/Pos/TCC/opt/arm-
2007q3/bin/arm-none-linux-gnueabi-

/beagleboard# cd kernel

/beagleboard/kernel# ../vendor/emlinux/support-tools/beagle_build_kernel.sh
```

A figura abaixo mostra o procedimento sendo executado no terminal (root):

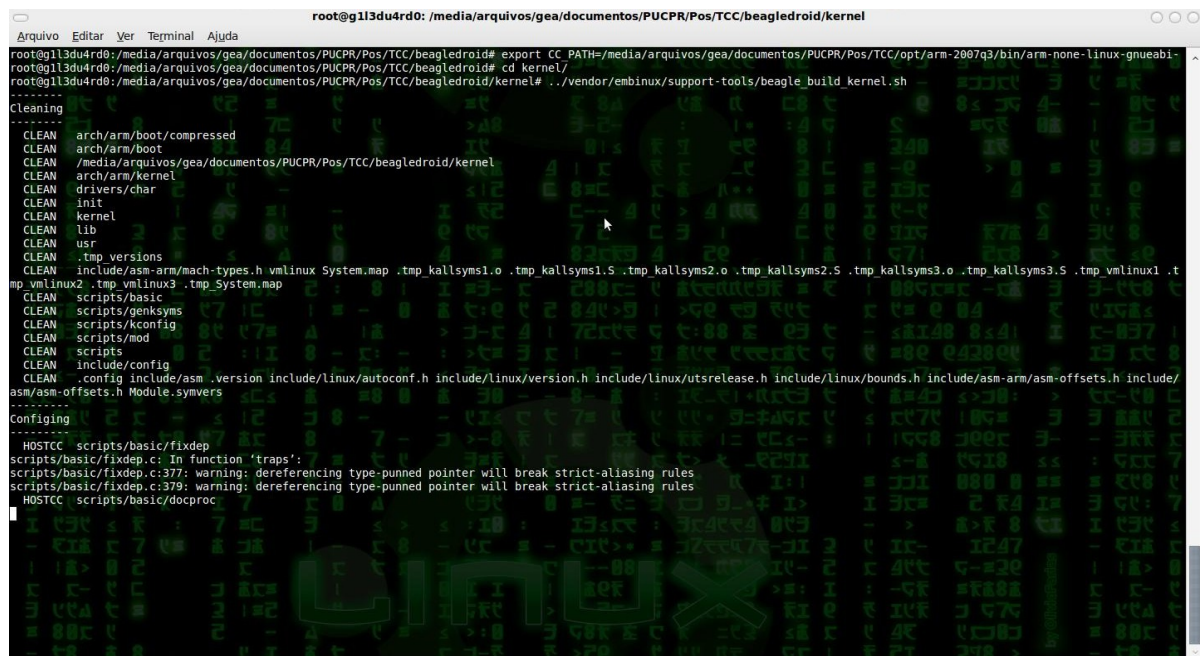


Figura 09: Compilando Kernel do Android (Linux)

4.4 Embarcando o Android no SDCard

Na BeagleBoard, existe a necessidade de utilizar um cartão *SD*, simulando um HD, que armazenará o sistema Android (Linux) e seus bootloaders. Este cartão deve conter o Sistema de Arquivos (*root filesystem*) gerado pós compilação.

Para tanto é preciso criar duas partições no cartão *SD*. A primeira é uma partição do tipo *FAT*, que servirá como *host* (hospedeiro) dos *bootloaders* e da imagem do kernel. Será usada como partição de inicialização, diretamente entendida, por padrão, pela BeagleBoard, não exigindo inteligência do bootloader ou do sistema operacional. O espaço restante do cartão é dedicado à partição do sistema de arquivos *ext3* (*Third Extended*).

O sistema de arquivos *root* do Android (partição *ext3*), pode estar em qualquer formato de sistema de arquivos entendido pelo kernel Linux, como o *JFFS2* (*Journaling Flash File System version 2*) e o *SquashFS*, que também são boas opções para sistemas de armazenamento baseados em flash.

4.5 Particionando o Cartão SD

Existem várias ferramentas que podem ser utilizadas para executar o processo de particionamento do cartão, porém escolhemos a ferramenta *GParted*. Em um terminal, com privilégios de *root*, digite o comando *GParted*, e aguarde a abertura do aplicativo:

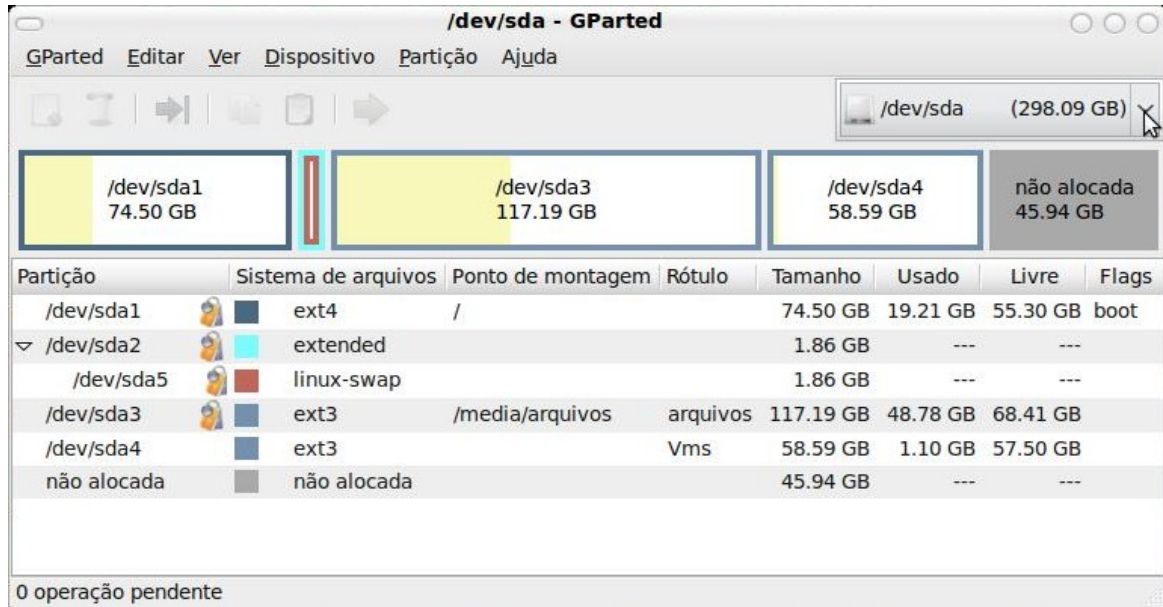


Figura 10: Particionado o cartão SD com o GParted

Nota: Caso não tenha o GParted instalado em seu computador, execute a instalação através do comando “apt-get install gparted”, no terminal com privilégios de root.

Inicialmente, a partição selecionada é a */dev/sda*, correspondente HD do microcomputador.

Com o cartão previamente conectado ao leitor de cartões do microcomputador, selecione a opção */dev/”nome_do_cartão”* (em nosso caso: *mmcblk0*) e com o botão direito do mouse, exclua qualquer partição existente no cartão.

Agora, sem partição alguma no SD, devemos criar as duas partições necessárias, a *FAT32* com tamanho de 128 MB (suficiente) e o restante do cartão, com uma partição *EXT3*.

Para isso, clique com botão direito do mouse sobre o dispositivo e selecione a opção novo. Utilize o tamanho (128Mb) e o tipo *FAT32* para primeira partição. Para a segunda partição é preciso informar apenas o tipo de partição *EXT3*, e aceitar a sugestão do aplicativo para o uso do restante do espaço disponível no cartão. Após estes procedimentos é necessário selecionar o botão Aplicar todas as operações para que as configurações sejam gravadas no cartão.

Após a criação das duas partições no cartão de memória, retire o mesmo do leitor de cartões (*slot*) de sua máquina e recoloque-o. Automaticamente o sistema operacional Ubuntu fará o reconhecimento e montagem das partições *FAT* e *EXT3*.

Veja as ilustrações abaixo para um melhor entendimento das configurações:

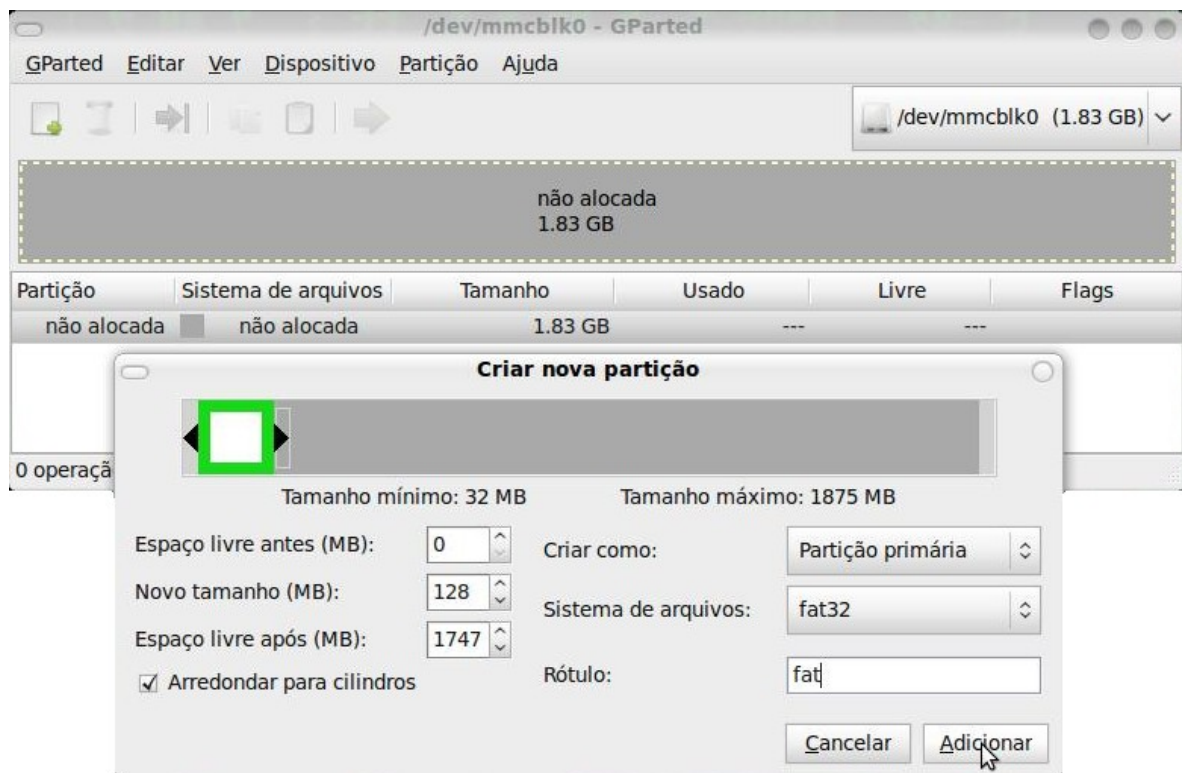


Figura 11: Criando partição FAT no SD Card com GParted

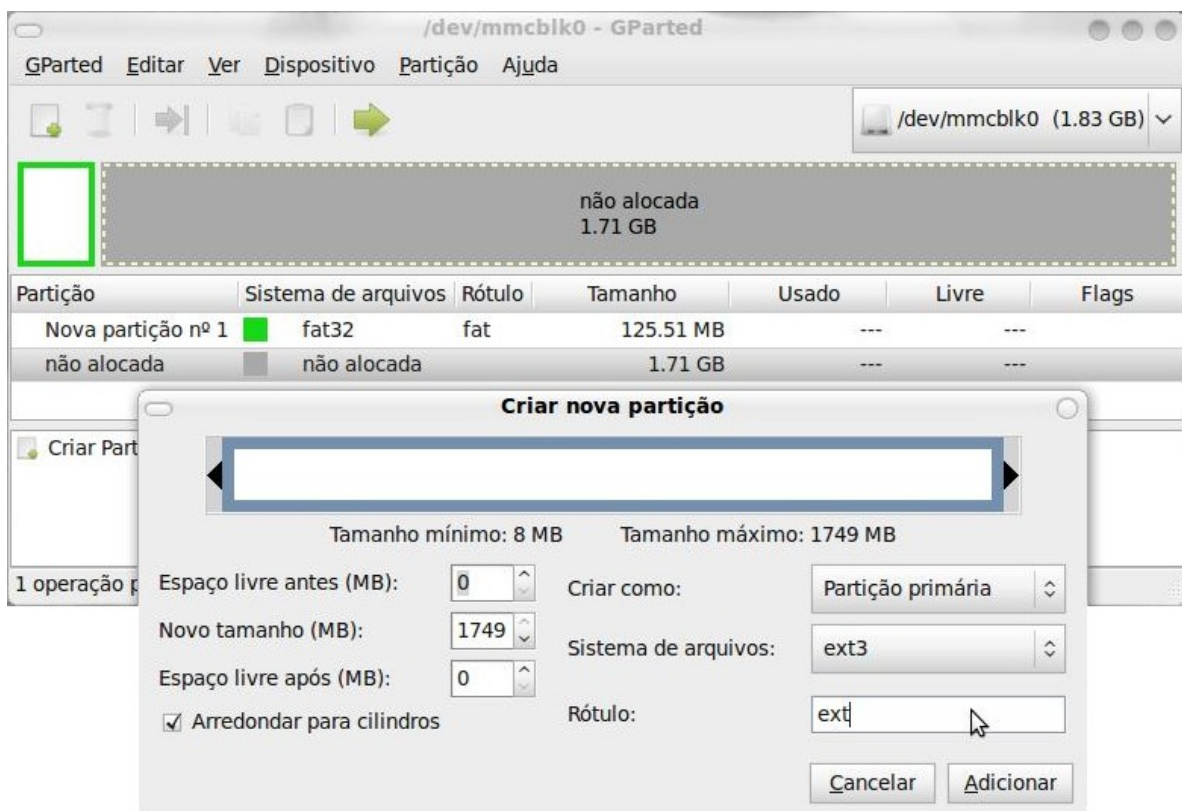


Figura 12: Criando partição EXT3 no SD Card com GParted

4.6 Copiando os arquivos para o Cartão SD e definindo as permissões

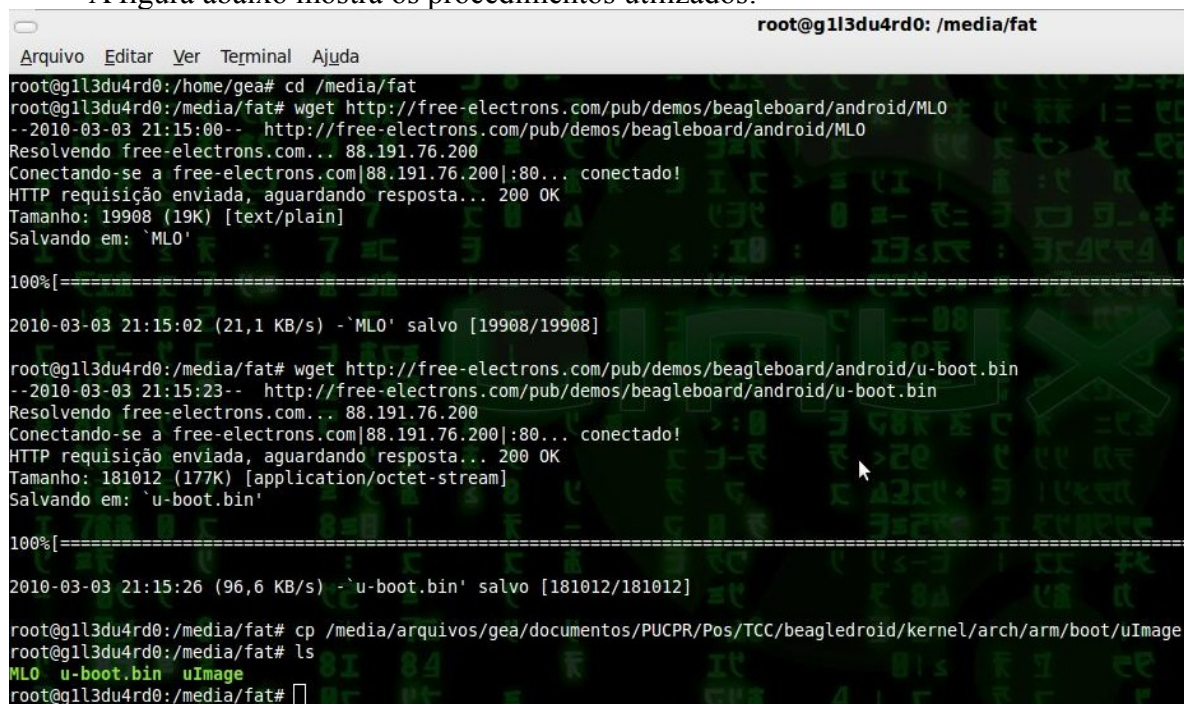
Considerando o Android e o Kernel Linux já compilados, conforme mostrado anteriormente, deve-se copiar os arquivos gerados para o SD e efetuar a configuração das permissões. Para tanto, no terminal com privilégios de root, digite o comando “`cd /media/fat`”, considerando que, durante a utilização do GParted para criação das partições, especificamos o rótulo de cada partição como “*fat*” para *FAT32* e “*ext*” para *EXT3*. Agora, dentro da partição FAT, proceda com a cópia dos arquivos de *boot* necessários, utilizando os comandos:

```
# wget http://free-electrons.com/pub/demos/beagleboard/android/MLO
# wget http://free-electrons.com/pub/demos/beagleboard/android/u-boot.bin
# cp /local_do_android_na_sua_maquina/kernel/arch/arm/boot/uImage .
```

Nota: No comando acima, assim como em outros utilizados neste artigo, deve-se atentar ao caminho correto dos arquivos do Android em seu microcomputador. No caso acima, troque o “local_do_android_na_sua_maquina” pelo caminho correto. Em nosso caso ficou assim:

```
# cp /media/arquivos/gea/documentos/PUCPR/Pos/TCC/beagledroid/kernel/arch/arm/boot/uImage .
```

A figura abaixo mostra os procedimentos utilizados:



```
root@g113du4rd0: /media/fat
Arquivo  Editar  Ver  Terminal  Ajuda
root@g113du4rd0:/home/gea# cd /media/fat
root@g113du4rd0:/media/fat# wget http://free-electrons.com/pub/demos/beagleboard/android/MLO
--2010-03-03 21:15:00--  http://free-electrons.com/pub/demos/beagleboard/android/MLO
Resolvendo free-electrons.com... 88.191.76.200
Conectando-se a free-electrons.com[88.191.76.200]:80... conectado!
HTTP requisição enviada, aguardando resposta... 200 OK
Tamanho: 19908 (19K) [text/plain]
Salvando em: `MLO'
100%[=====]
2010-03-03 21:15:02 (21,1 KB/s) - `MLO' salvo [19908/19908]

root@g113du4rd0:/media/fat# wget http://free-electrons.com/pub/demos/beagleboard/android/u-boot.bin
--2010-03-03 21:15:23--  http://free-electrons.com/pub/demos/beagleboard/android/u-boot.bin
Resolvendo free-electrons.com... 88.191.76.200
Conectando-se a free-electrons.com[88.191.76.200]:80... conectado!
HTTP requisição enviada, aguardando resposta... 200 OK
Tamanho: 181012 (177K) [application/octet-stream]
Salvando em: `u-boot.bin'
100%[=====]
2010-03-03 21:15:26 (96,6 KB/s) - `u-boot.bin' salvo [181012/181012]

root@g113du4rd0:/media/fat# cp /media/arquivos/gea/documentos/PUCPR/Pos/TCC/beagledroid/kernel/arch/arm/boot/uImage
root@g113du4rd0:/media/fat# ls
MLO  u-boot.bin  uImage
root@g113du4rd0:/media/fat#
```

Figura 13: Copiando os arquivos para partição FAT.

Feita a cópia de arquivos para a partição FAT32 do cartão de memória, devemos efetuar a cópia dos arquivos necessários e permissões para a partição EXT3. Para tanto, utilize os seguintes comandos:

```
# cp -a /local_do_android_na_sua_maquina/out/target/product/generic/root/* .
# cp -a /local_do_android_na_sua_maquina/out/target/product/generic/system/*
./system/
# chown -R root.root .
# chmod -R a+rwX data system
```


Nota: No comando acima, assim como em outros utilizados neste artigo, deve-se atentar ao caminho correto dos arquivos do Android em seu microcomputador. No caso acima, troque o “local_do_android_na_sua_maquina” pelo caminho correto. Em nosso caso ficou assim:

```
# cp -a /media/arquivos/gea/documentos/PUCPR/Pos/TCC/beagledroid/out/target/  
product/generic/root/* .  
# cp -a /media/arquivos/gea/documentos/PUCPR/Pos/TCC/beagledroid/out/target/product/  
generic/system/* ./system/  
# chown -R root.root .  
# chmod -R a+rwX data system
```

A figura abaixo mostra os procedimentos utilizados:

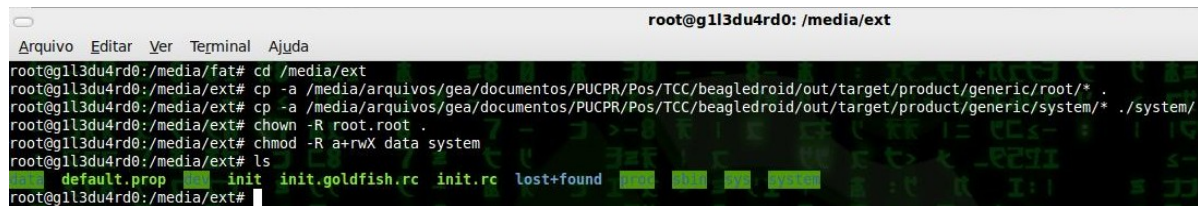
A screenshot of a terminal window titled "root@gl13du4rd0: /media/ext". The terminal shows a series of commands being executed: "cd /media/ext", "cp -a /media/arquivos/gea/documentos/PUCPR/Pos/TCC/beagledroid/out/target/product/generic/root/* .", "cp -a /media/arquivos/gea/documentos/PUCPR/Pos/TCC/beagledroid/out/target/product/generic/system/* ./system/", "chown -R root.root .", and "chmod -R a+rwX data system". The prompt "root@gl13du4rd0: /media/ext#" is visible at the end of each line. The background of the terminal has a green and black pattern.

Figura 14: Copiando os arquivos para partição EXT3.

Feita a cópia dos arquivos para as partições, estamos com *SD Card* praticamente pronto. Faltam apenas algumas configurações pós-compilação, que serão apresentadas a seguir, para a inicialização do Android e assim, efetuar a conexão via rede Wi-Fi.

4.7 Configurando a BeagleBoard via porta Serial (HyperTerminal)

Para que esta conexão seja realizada, é necessário um cabo serial, qual deve ser conectado a placa BeagleBoard e a um microcomputador com porta RS-232. Para a construção deste cabo, utilizamos a documentação disponibilizada no site da fabricante da placa.

Com o cabo previamente conectado, inicializamos o aplicativo *HyperTerminal*, este presente no sistema Windows XP (é possível efetuar esse mesmo processo com programas similares dentro do Linux). Na conexão, utilize os seguintes parâmetros:

- BAUD RATE:	115200
- DATA:	8 bit
- PARITY:	none
- STOP:	1bit
- FLOW CONTROL:	none

Criada a conexão, e, com cabo conectado a porta serial RS-232 (COM1) de seu computador e na porta serial da BeagleBoard, pode-se ligar a placa e observar no HyperTerminal os dados apresentados, como mostra a figura abaixo:

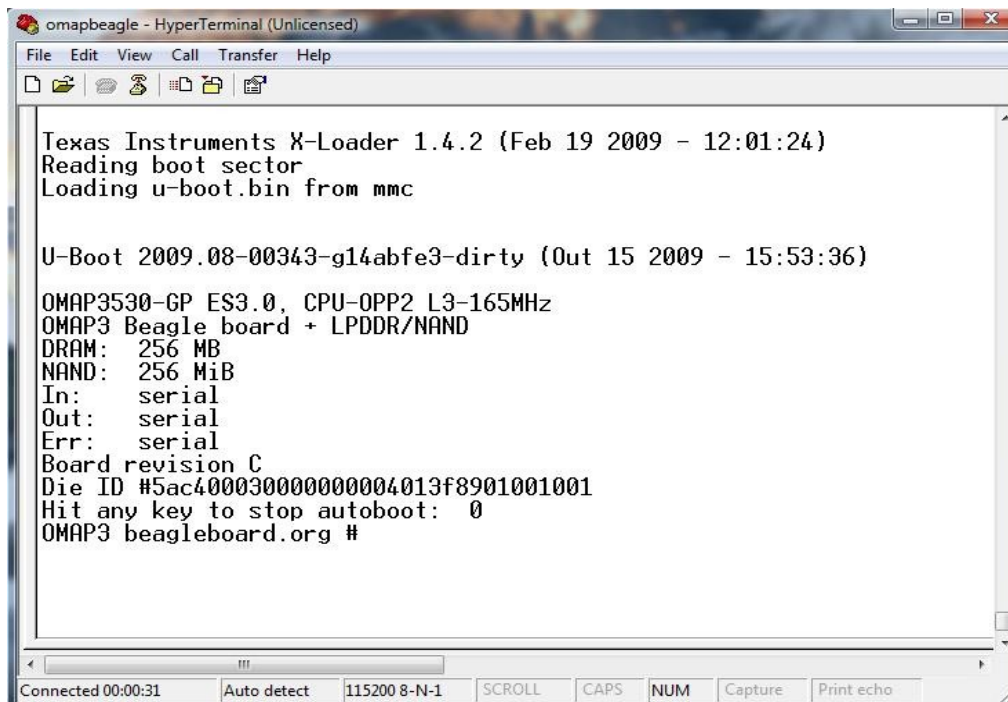


Figura 15: Inicializando a BeagleBoard via HyperTerminal

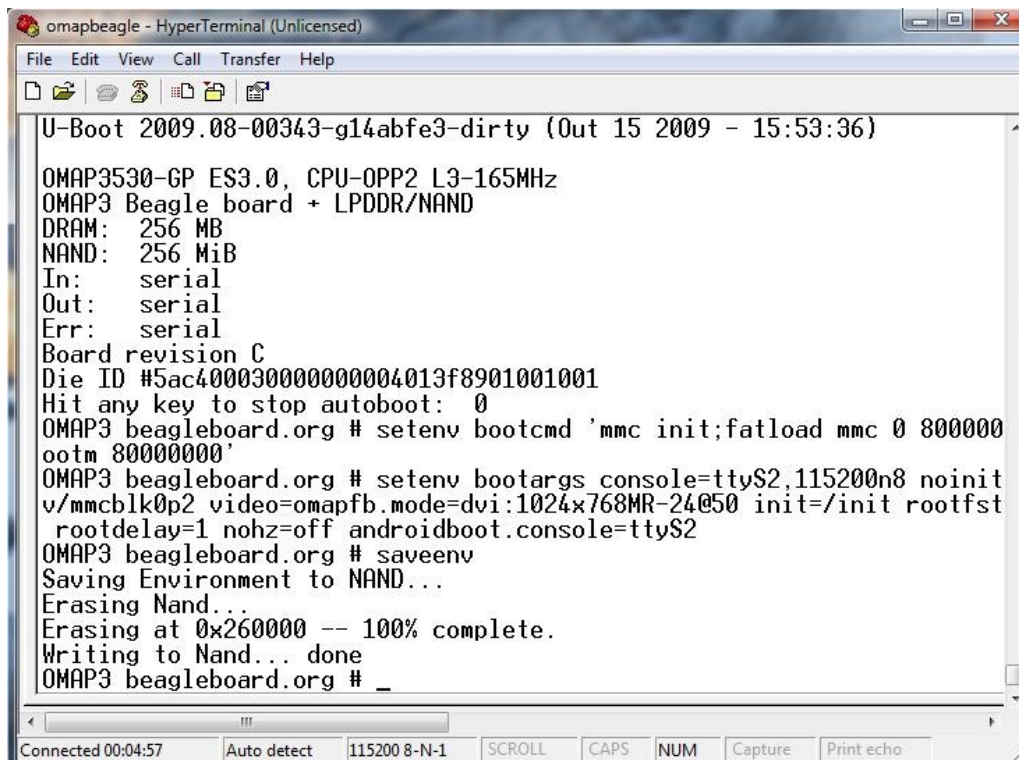
Para que o *boot* no Android seja efetuado pelo cartão de memória (*MMC – SD Card*), é preciso configurar utilizando os seguintes comandos:

```
- setenv bootcmd 'mmc init;fatload mmc 0 80000000 uImage;bootm 80000000'
- saveenv
```

Para a configuração via linha de comandos do kernel, indicando configurações de vídeo, sistemas de arquivos, dispositivos, entre outros necessários, utilize o comando:

```
setenv bootargs console=ttyS2,115200n8 noinitrd root=/dev/mmcblk0p2
video=omapfb.mode=dvi:1280x720MR-24@50 init=/init rootfstype=ext3 rw
rootdelay=1 nohz=off androidboot
```

Estes procedimentos são apresentados na figura abaixo:



```
omapbeagle - HyperTerminal (Unlicensed)
File Edit View Call Transfer Help

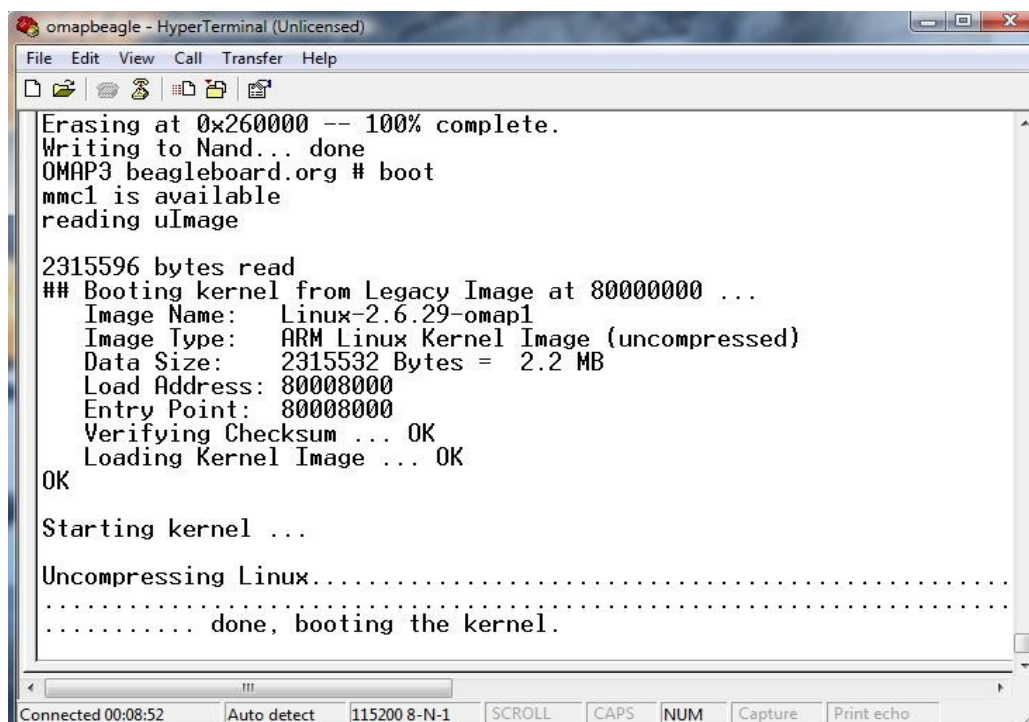
U-Boot 2009.08-00343-g14abfe3-dirty (Out 15 2009 - 15:53:36)

OMAP3530-GP ES3.0, CPU-OPP2 L3-165MHz
OMAP3 Beagle board + LPDDR/NAND
DRAM: 256 MB
NAND: 256 MiB
In: serial
Out: serial
Err: serial
Board revision C
Die ID #5ac400030000000004013f8901001001
Hit any key to stop autoboot: 0
OMAP3 beagleboard.org # setenv bootcmd 'mmc init;fatload mmc 0 800000
ootm 80000000'
OMAP3 beagleboard.org # setenv bootargs console=ttyS2,115200n8 noinit
v/mmcblk0p2 video=omapfb.mode=dvi:1024x768MR-24@50 init=/init rootfst
rootdelay=1 nohz=off androidboot.console=ttyS2
OMAP3 beagleboard.org # saveenv
Saving Environment to NAND...
Erasing Nand...
Erasing at 0x260000 -- 100% complete.
Writing to Nand... done
OMAP3 beagleboard.org # _

Connected 00:04:57 Auto detect 115200 8-N-1 SCROLL CAPS NUM Capture Print echo
```

Figura 16: Configurando a BeagleBoard via HyperTerminal

Por fim, para inicialização do sistema a partir do cartão de memória, utilize o comando *boot*, no HyperTerminal, como mostrado na figura abaixo:



```
Erasing at 0x260000 -- 100% complete.
Writing to Nand... done
OMAP3 beagleboard.org # boot
mmc1 is available
reading uImage

2315596 bytes read
## Booting kernel from Legacy Image at 80000000 ...
Image Name: Linux-2.6.29-omap1
Image Type: ARM Linux Kernel Image (uncompressed)
Data Size: 2315532 Bytes = 2.2 MB
Load Address: 80008000
Entry Point: 80008000
Verifying Checksum ... OK
Loading Kernel Image ... OK
OK

Starting kernel ...

Uncompressing Linux.....
..... done, booting the kernel.

Connected 00:08:52 Auto detect 115200 8-N-1 SCROLL CAPS NUM Capture Print echo
```

Figura 17: Android inicializando no HyperTerminal

5. Configurações de Rede Wi-Fi (pós-compilação)

As configurações efetuadas depois do sistema compilado devem ser efetuadas parte

antes do primeiro boot do sistema e parte pós primeiro boot. Isso porque no primeiro boot serão criados os diretórios do sistema, necessários para a configuração pós primeiro boot. Assim sendo, antes da primeira inicialização do sistema operacional Android, deve-se editar algumas configurações necessárias dentro do arquivo *init.rc*, executado durante o boot do Linux.

No arquivo *init.rc*, encontrado na partição EXT do *SD Card*, adicione em seu final os seguintes comandos:

```
#----- #
# Estrutura de Diretórios Wi-Fi (Gil Eduardo de Andrade) #
#----- #
mkdir /data/misc/wifi 0770 system system
mkdir /data/misc/wifi/sockets 0770 system system
mkdir /data/system/wpa_supplicant 0770 system system
mkdir /data/misc/dhcp 0770 dhcp dhcp
chown dhcp dhcp /data/misc/dhcp
# ----- #

# -----#
# Define interface a wifi = wlan0 para wifi (Gil Eduardo) #
# -----#
setprop wifi.interface wlan0
# ----- #

# ----- #
# Serviços/Daemons (Gil Eduardo de Andrade) #
# ----- #
service ifcfg_ralink /system/bin/ifconfig wlan0 up
    disabled
    oneshot
service wpa_supplicant /system/bin/logwrapper /system/bin/wpa_supplicant -Dwext -iwlan0
-c /system/etc/wifi/wpa_supplicant.conf -dd
    disabled
    group system
service dhcpcd /system/bin/logwrapper /system/bin/dhcpcd -d wlan0 -dd
# ----- #
```

Aplicada esta configuração, pode-se proceder com o primeiro *boot* no Android, assim serão criados os diretórios necessários para o funcionamento do Wi-Fi. Assim que o Android tiver sido inicializado, desligue-o, retire o *SD Card* da placa e recoloque este cartão no computador com Linux.

Abra a partição EXT, que contém os diretórios Wi-Fi, criados no primeiro boot, através dos comandos inseridos no arquivo *init.rc*. Para concluir a configuração, acesse o arquivo */system/etc/dhcpd/dhcpcd.conf* e modifique a linha *interface tiwlan* para *interface wlan*. Salve as alterações.

Nos arquivos */data/misc/wifi/wpa_supplicant* e */system/etc/wifi/wpa_supplicant* é necessário alterar a linha *ctrl_interface=tiwlan0* para *ctrl_interface=DIR=/data/system/wpa_supplicant GROUP=system*. Salve as alterações.

Agora o Android já está preparado para utilizar uma rede Wi-Fi, conectar-se em um ponto de acesso qualquer, autenticar-se e navegar na internet.

Por fim, desmonte as partições EXT e FAT, retire o SD do computador e recoloque-o na BeagleBoard.

5.1 Conectando e Navegando na Internet via rede Wi-Fi

Inicialize o Android na BeagleBoard, agora com todas as configurações necessárias efetuadas, acesse através de sua interface, o menu, localizado a direita e selecione a opção *Settings* → *Wireless Controls* → *Wi-Fi* e ative o dispositivo Wi-Fi. Com isso, o sistema já está apto para efetuar uma conexão com um ponto de acesso e navegar na internet.

6. O Módulo JNI: Gerando uma Falha de Denial of Service (DoS)

O sistema operacional Android é implementado utilizando a linguagem de programação *Java*, que permite portabilidade e a criação de uma interface muito atrativa para o usuário. Contudo a linguagem *Java* não permite uma comunicação direta de baixo nível, ou seja, através dela, não é possível o acesso direto ao hardware.

Por esta razão, a linguagem C / C++ foi definida, pois, o *Java* possui um módulo chamado *JNI*, que permite que funções C / C++ sejam invocadas dentro um programa *Java*, de forma transparente, através do envio e recebimento de parâmetros a estas funções. Isso permite que o *Java*, através do C, efetue a comunicação com dispositivos de hardware.

Porém, pesquisando e analisando o funcionamento do Android, constatou-se que uma falha dentro de uma função nativa C, invocada por um programa *Java*, programado para acessá-la, causa uma falha grave de segurança, gerando um erro de *Denial of Service* (Negação de Serviço).

Então, dentro dos serviços do Android que precisam de comunicação direta com hardware e conseqüentemente utilizam JNI, escolhemos o serviço de bateria (Battery Service) para implementar uma chamada C por um programa *Java*, gerando um erro dentro da função C e demonstrando o comportamento do sistema operacional Android.

Abaixo são mostradas os arquivos e modificações efetuadas:

- /frameworks/base/services/JNI/com_android_server_BatteryService.cpp

```
static jint android_server_BatteryService_getBattery(JNIEnv* env, jobject obj)
{
    int i;
    int retorno[5];
    for(i=0; i < 10; i++) //Erro, acessa posição do vetor que ã existe
        retorno[i] = i * 2;
    return retorno[4];
}
static JNINativeMethod sMethods[] = {
    /* name, signature, funcPtr */
    {"native_update", "()V", (void*)android_server_BatteryService_update},
    {"getBattery", "()I", (void*)android_server_BatteryService_getBattery},
};
```

- /frameworks/base/services/java/com/android/server/BatteryService.java

```
private int Ibattery;

public BatteryService(Context context) {
    mContext = context;
    mBatteryStats = BatteryStatsService.getService();

    mUEventListener.startObserving("SUBSYSTEM=power_supply");
```



```
// set initial status
update();

Ibattery = getBattery();
Log.e("[Dos]: ", "Obtendo Valor Bateria [" + Ibattery + "]);
}
```

Estas modificações causaram ao Android uma grande instabilidade, pois, com elas, não foi mais possível inicializar o sistema. Ao tentar iniciá-lo o mesmo ficou “religando” sem parar, não chegando a apresentar nem sua tela de inicial. Falha esta conhecida como *Denial of Service (DoS)* ou Negação de Serviço.

7. Instalando o Android no VirtualBox (Máquina Virtual)

O VirtualBox é um aplicativo de virtualização, desenvolvido pela *Sun Microsystems*, que possibilita a criação de máquinas virtuais, ou seja, um ambiente reservado dentro do sistema operacional principal, onde é possível instalar e utilizar paralelamente outros sistemas operacionais. O VirtualBox funciona em Linux, Macintosh e Windows e pode ser baixado e instalado gratuitamente através de seu site oficial, www.virtualbox.org.

Nesta demonstração, instalamos o VirtualBox em um computador operado com o Linux Ubuntu 9.10, através da Central de Programas do Ubuntu (repositório)

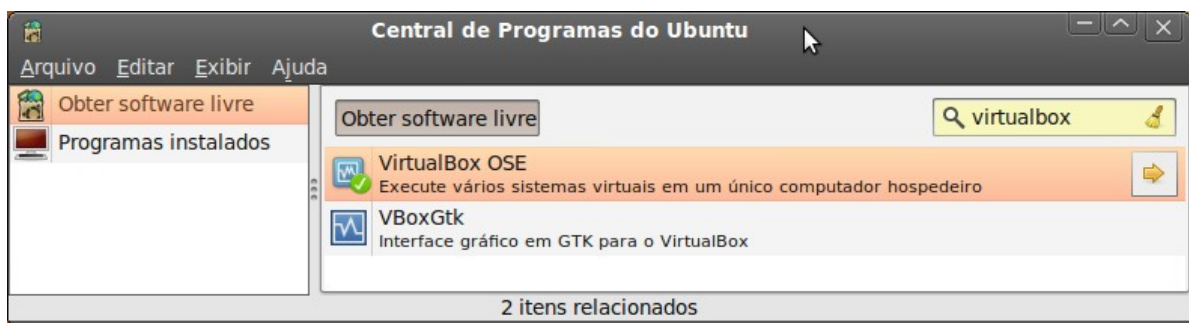


Figura 18: Central de Programas do Ubuntu Linux

No VirtualBox, clique em Novo para iniciar o processo de criação de uma máquina virtual. O *Assistente de Criação de Máquina* será apresentado, seguindo seus passos, informe primeiramente o nome da máquina virtual (em nosso exemplo, utilizamos “Android”). No tipo de sistema, selecione o “Linux”, versão “2.6”. Na próxima tela do assistente defina entre 64 e 128 MB para a memória principal (em nosso exemplo, utilizamos “128”).

Na próxima tela, a de configuração do disco rígido virtual, selecione a opção “Criar novo disco rígido”. Ao clicar em próximo, o VirtualBox abre um novo assistente, o de *Criação de Discos Rígidos Virtuais*. Seguindo seus passos, informe que utilizaremos um armazenamento de tamanho fixo e clique em próximo. A localização padrão deste disco virtual é o diretório `/home/usuario/.VirtualBox/HardDisks`, portanto, insira um nome (ex: “Android.vdi”) para a criação do arquivo. Não é necessários mais do que *1 GB* para o tamanho deste disco virtual.

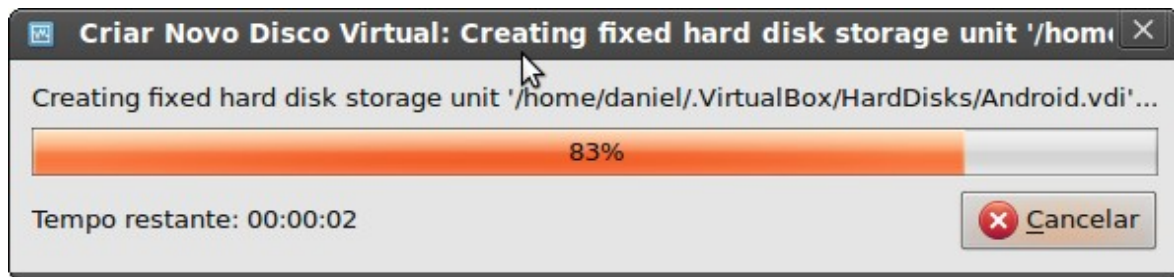


Figura 19: VirtualBox criando um disco rígido de tamanho fixo

A última tela do assistente de criação de disco virtual traz um resumo da configuração. Caso precise alterar algum parâmetro, clique em voltar e faça as alterações necessárias. Ao clicar em finalizar, voltamos a tela principal do VirtualBox, agora, com o sistema *Android* listado na barra lateral esquerda do aplicativo.



Figura 18: VirtualBox com sistema "virtual" Android

Selecione o *Android* e clique no botão Iniciar. O *Assistente de Primeira Execução* será exibido e nele informaremos a origem da mídia de instalação. Selecione a opção arquivo de imagem e procure pelo arquivo "*liveandroidv0.2.iso*". Na próxima tela, clique em finalizar.

Pronto! O sistema operacional Android iniciará sua execução.

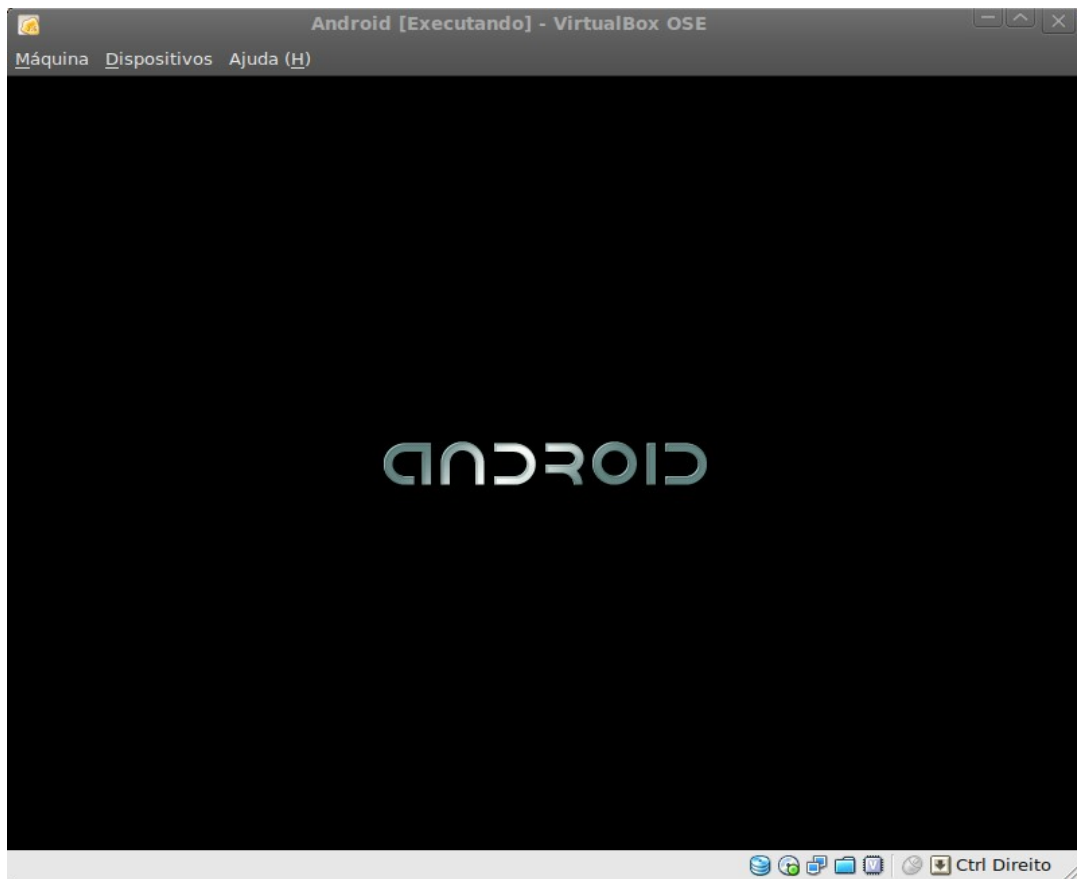


Figura 20: Android em inicialização

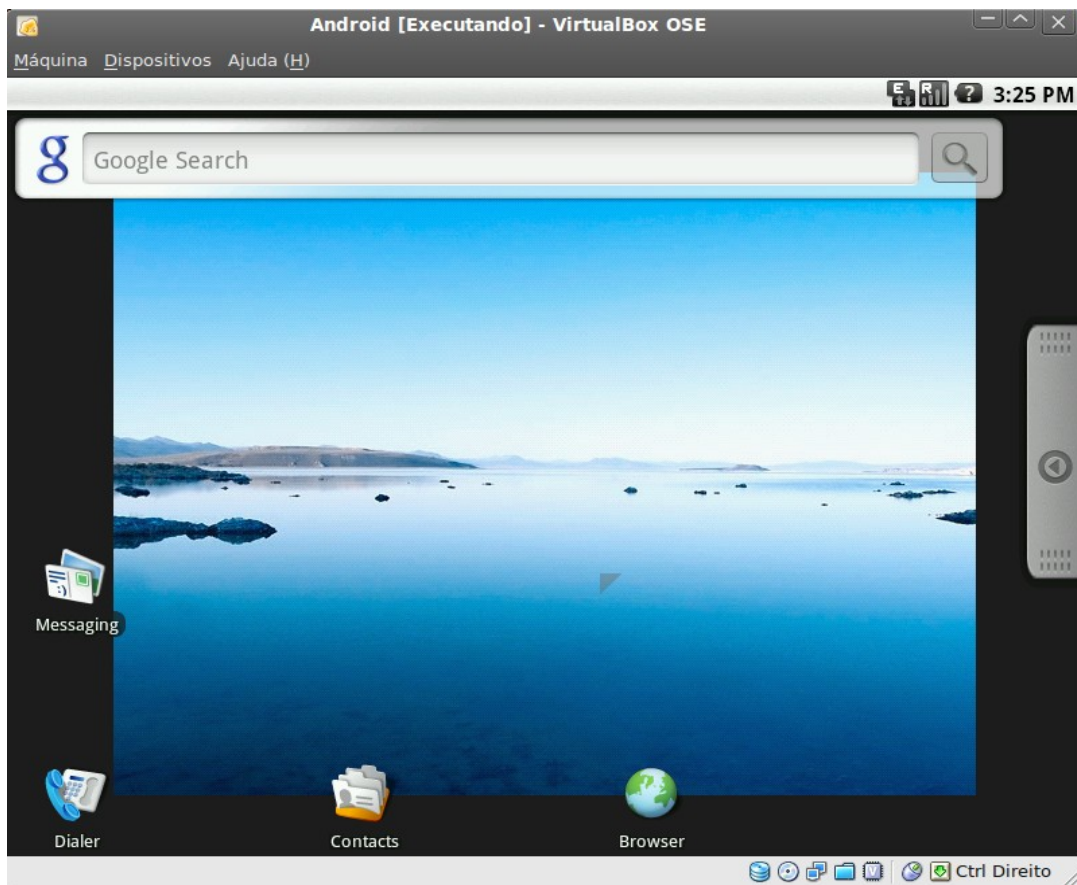


Figura 21: Android já carregado, pronto para uso

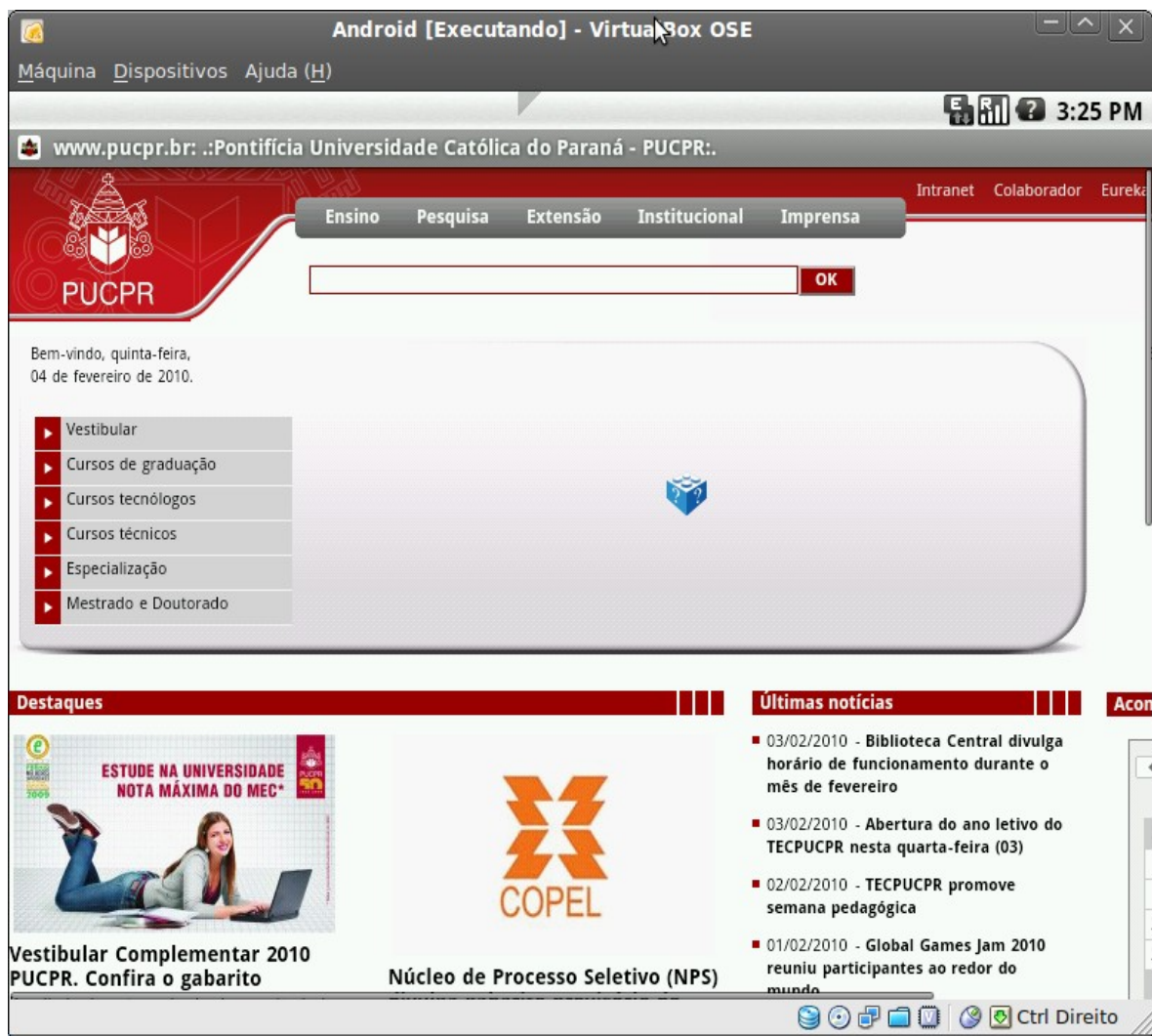


Figura 22: Android navegando na Internet (configuração de rede via DHCP)

Nota: Para livrar o mouse da máquina virtual, tecle o botão Ctrl 'direito' do seu teclado; Para reutilizar o sistema Android, não utilize a opção Máquina / Desligamento por ACPI, mas a opção “Salvar estado da máquina”, exibida ao clicar no botão fechar da janela da máquina virtual.

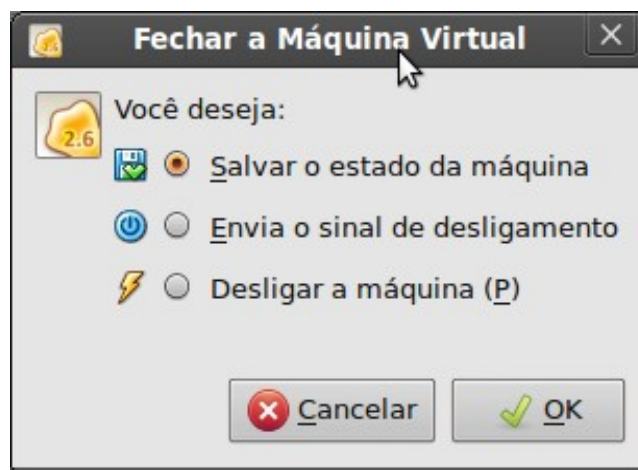


Figura 23: VirtualBox: Opção de salvar o estado da máquina virtual

8. Bibliografia

- [1] Android – Sites oficiais do Projeto. Acesso em Janeiro de 2010. Disponível em: <http://www.android.com> e <http://code.google.com/intl/pt-BR/android>
- [2] Wikipédia – A Enciclopédia Livre: Android. Acesso em Janeiro de 2010. Disponível em: <http://pt.wikipedia.org/wiki/Android>
- [3] Clube do Hardware: Entendendo o Google Android. Acesso em Janeiro de 2010. Disponível em: <http://www.guiadohardware.net/artigos/google-android>
- [4] Geek: Vulnerabilidades no Android permitem ataques. Acesso em Janeiro de 2010. Disponível em: <http://www.geek.com.br/posts/11091-vulnerabilidades-em-palm-e-android-permitem-ataques-e-roubo-de-informacoes>
- [5] e-Thesis: Pesquisadores identificam 5 famílias de vulnerabilidades do Android. Acesso em Janeiro de 2010. Disponível em: http://www.e-thesis.inf.br/index.php?option=com_content&task=view&id=6376&Itemid=52
- [6] Security Focus: Android denial-of-service issues. Acesso em Janeiro de 2010. Disponível em: <http://www.securityfocus.com/archive/1/506948>
- [7] IT Web: Android apresenta vulnerabilidade. Acesso em Janeiro de 2010. Disponível em: <http://www.itweb.com.br/noticias/index.asp?cod=54913>
- [8] IDG Now: Android é vulnerável a ataque pelo browser, alertam pesquisadores. Acesso em Janeiro de 2010. Disponível em: <http://idgnow.uol.com.br/seguranca/2008/10/27/android-e-vulneravel-a-ataque-pelo-browser-alertam-pesquisadores>
- [9] World of Bit: Android na mira dos Crackers: Várias falhas na segurança. Acesso em Janeiro de 2010. Disponível em: <http://worldofbit.com/wob/index.php/noticias/android-na-mira-dos-cracker-varias-falhas-na-seguranca.html>
- [10] BeagleBoard – Site oficial. Acesso em Dezembro de 2009. Disponível em: <http://beagleboard.org>
- [11] IBM: Boot do Linux na BeagleBoard: Acesso em Janeiro de 2010. Disponível em: <http://www.ibm.com/developerworks/br/linux/library/l-beagle-board/index.html>
- [12] Free-Electrons: Arquivos do Beagledroid. Acesso em Dezembro de 2009. Disponível em: <http://free-electrons.com/pub/demos/beagleboard/android/>
- [13] G1 – Globo.com: Falhas de Segurança. Acesso em Janeiro de 2010. Disponível em: <http://g1.globo.com/Noticias/Tecnologia/0,,MUL1128175-6174,00-SAIBA+O+QUE+SAO+FALHAS+DE+SEGURANCA+DIA+ZERO+E+COMO+SE+PROTEGER+DELAS.html>
- [14] Terra: Dell apresenta tablet PC com Android. Acesso em Janeiro de 2010. Disponível em: <http://tecnologia.terra.com.br/interna/0,,OI4193011-EI4799,00-Dell+apresenta+tablet+PC+com+Android+na+CES.html>
- [15] VirtualBox.org. Acesso em Janeiro de 2010. Disponível em: <http://www.virtualbox.org>