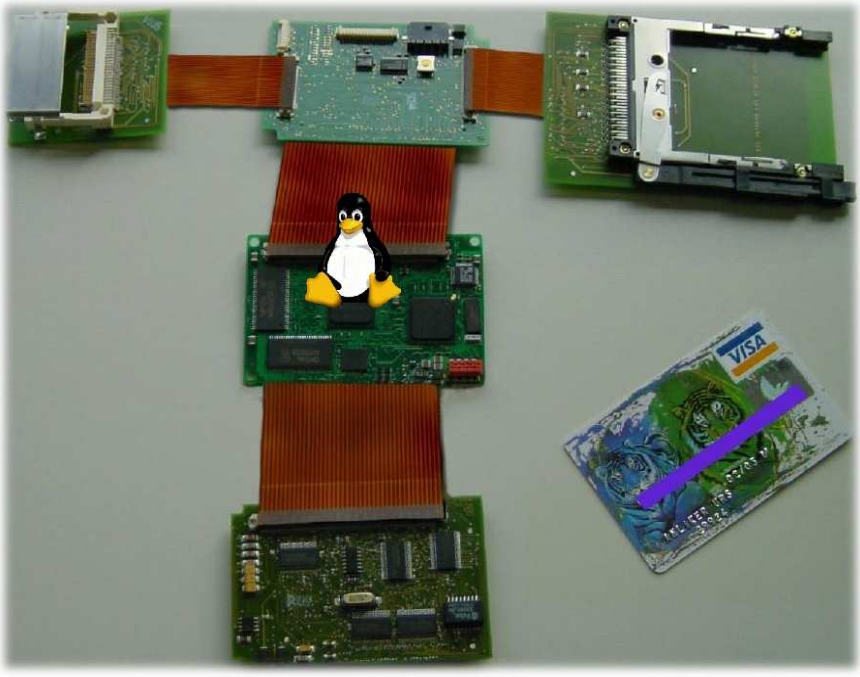
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Perceptual Computing
Computer Vision

Semester Thesis

Intimate Linux on the WearARM Platform

March 21, 2003



Author: Bernhard Wymann
Advisor: Nicky Kern

Prof. Bernt Schiele
Institute of Scientific Computing
ETH Zürich

Table of Contents

Table of Contents

1 Introduction

1.1 The WearARM Intimate Linux

The WearARM Intimate Linux is a Debian based Linux system for the WearARM. There is nothing special about the system, it contains all facilities you would expect from any "normal" Linux system. The WearARM is an attempt to build a computer which combines wearability and high speed. The hardware consists of the StrongArm SA1110 processor, the SA1111 companion chip, 2MB Flash−RAM, 128MB SDRAM and a video digital−analog converter (DAC) for VGA video output.

1.2 Document Overview

This chapter will introduce you briefly in the chapters of this document:

- Report: This documentation is the result of a semester thesis. This chapter contains the details about the the semester thesis.
- Intimate Linux Installation: Explains how to install the existing Intimate Linux images on the WearARM and how to start the WearARM up.
- The Operating System: That is the technical documentation to the system, you learn how to build, configure and maintain the system.
- The Boot Process: Describes the boot process of the system.
- Problems: Descriptions of discovered kernel and bootloader problems.
- Hardware: What devices are known to work, how do I use a certain device.

1.3 Existing WearARM Documentation

1.3.1 Hardware Documentation

A nice overview of the WearARM architecture can be found on
http://www.wearable.ethz.ch/projects/research/weararm.

1.3.2 Software Documentation

Documentation about the "old" Linux for the WearARM, the kernel and known problems is available from:

- http://www.wearable.ethz.ch/projects/research/weararm/linux/index.html
- http://colo.heeltoe.com/weararm

2 Report

2.1 Abstract

The main result of this thesis is a clean and well configured and documented Intimate Linux system for the WearARM. It is configured to work with various network hardware in the IFW building, so you can simply hotplug your wireless or ethernet card and it should work. Because of the testing there is now a good understanding of the problems of the kernel and the hardware, e. g. a serious kernel memory management problem has been discovered. To ease the device handling there is the device file system integrated in the system and various devices has been tested. To enable USB hotplugging an USB manager is installed.

There have also been built Opie packages for the ARM and x86 platform to enable people to crossdevelop Opie applications. Another important point is the available documentation of the system.

2.2 Zusammenfassung

Das Resultat dieser Semesterarbeit ist ein sauber zusammengebautes, gut konfiguriertes und dokumentiertes Intimate Linux Betriebssystem für den WearARM. Verschiedene Netzwerkhardware ist vorkonfiguriert, so dass Ethernet und Wireless PCMCIA Erweiterungskarten nach dem Einstecken automatisch aktiviert und für den Gebrauch im IFW−Gebäude konfiguriert werden. Durch die vielen Tests des Systems wurden Hardware− und Kernel−Probleme entdeckt, wie z. B. ein ernstzunehmendes Hauptspeichermanagement−Problem des Kernels. Um den Umgang mit Geräten zu vereinfachen wurde das "Device File System" in das System integriert und mehrere Geräte wurden getestet. Um USB "hotplugging" zu unterstützen wurde ein USB−Manager installiert.

Es wurden auch Opie−Pakete für die ARM und x86 Plattform gebaut, um plattformübergreifende Opie−Applikationsentwicklung und Portierung zu ermöglichen. Weiterhin ist die verfügbare Dokumentation des Systems zu erwähnen.

2.3 Task Description

2.3.1 Introduction

The vision of a wearable computer is that of an intelligent assistant which is always with the user and helps to solve everyday tasks. In the context of the ETH Wearable Project (http://www.wearable.ethz.ch) a proprietory hardware plattform has been developed. The so−called WearARM see (http://www.wearable.ethz.ch/projects/research/weararm/index.html) is a StrongARM−based computer, running under the Linux operating system.

The Linux installation on the WearARM has evolved over time. Now libraries and applications have become inconsistent with each other. The goal of this semester project is thus to setup a clean working environment on the WearARM, that allows for further investigation.

2.3.2 Task Description

During this semester project the existing StrongARM Linux Distribution *Intimate* shall be adapted and installed on the WearARM. More specifically:

1. Setup a base system, using the Intimate distribution (http://www.handhelds.org) and the existing running kernel as a start point.
2. For each of the following items, verify that they work correctly or otherwise set them up accordingly:
    - Serial ports
    - PCMCIA Support, especially Wireless LAN

- ♦ USB for the following devices: Audio Device, Mouse, Keyboard
- ♦ Opie: Opie is a graphical environment for portable devices, similar to KDE for desktop computers.
- ♦ QT Embedded

The X−Window System need not be set up.

3. To test the above setup, port an existing iPAQ application for activity recognition to the WearARM. Verify, that it is working by performing some small test runs.
4. This semester project should put special emphasis on the documentation of the system. Therefore a web−page shall be created, that documents the different parts of the system, their setup, and any known bugs of both hard− and software. This page should be used as reference by future users and developers.
5. Port the Hidden Markov Model Toolkit to the WearARM plattform.
6. In the smart−its project Blue−Tooth nodes have been developed. Connecting them to the WearARM and setting up an *existing* (partial) Blue−Tooth protocol stack would greatly enhance the communication capabilities of the WearARM.
7. The boot process is currently quite complex, starting several boot loaders and intermediate root images. Simplifying this process would greatly speed up the boot process and reduce the system's complexity.
8. The current kernel can access only half of the available memory. Removing this barrier would allow for more complex and memory−intensive applications to run on the WearARM.

2.3.3 Environment

The hardware platform is the WearARM. System configuration will require a multitude of programming languages and configuration files. Software development will be done C++ using the QT toolkit.

As all semester projects it shall be concluded with a final report and presentation. See the internal PCCV guidelines for a detailed description (http://www.vision.ethz.ch/dasa/guidelines_dasa.pdf). This semester project is carried out under the supervision of Prof. Bernt Schiele and tutored by Nicky Kern.

2.4 Motivation

The WearARM is a StrongARM−based computer, running under a Linux operating system. The goal of the WearARM is to combine complex functionality and high speed with real wearability.

The Linux installation on the WearARM has evolved over time. Libraries and applications have become inconsistent with each other. The goal of this semester project was thus to setup a clean working Linux system on the WearARM, to set up and test the system with various devices, to gain experience with the system and to document it.

2.5 Overview of the Thesis

This rest of this chapter describes the work that has been done by me for the semester thesis, the results and the conclusions.

2.5.1 Setting up a Base System

The goal of this part was to understand the current state of the WearARM Linux in the beginning and to set up a clean Linux base system.

I started with studying the system that was already in place. To be able to track down problems of the new upcoming system I saved a copy of the old system, so I was able to compare the behaviour of the old and the new system. This was very useful to distinguish kernel problems from distribution and setup problems (the kernel version remained the same).

The next step was to dig into the Intimate system. The first problem was to find an image of the system, because the normal installation process works over the network with a running familiar distribution. Finally I took the only one (very old) image as base for the new system. I put it to the disk and tried to adopt the ramdisk and the kernel from the old WearARM Linux so that they work together with the new system. Because the bootprocess, the ramdisk and the kernel of the existing system did not work well it took a lot of time to make it work together with Intimate. On the original WearARM Linux there are inconsistent device files, libraries and configuration files on the ramdisk and on the harddisk. The kernel parameters are compiled right into the kernel, and it does not accept parameters given by the boot loader or the makefile. It was necessary to reconfigure and compile the kernel and to modify the ramdisk for the new bootprocess. I also set up the "Device File System" to handle devices.

After I was able to boot the system for the first time I recognized that on the Intimate distribution e. g. PCMCIA support is missing and not available. That is because of the boot process of the Intimate system. On the iPAQ the bootprocess mounts the Familiar image which is in flash memory. Familiar loads the PCMCIA and disk drivers, and after that the Intimate bootprocess gets started. Intimate is located on a PCMCIA harddisk. Because of that there are important things like PCMCIA support missing in the Intimate image. So I decided to build a combination of Familiar and Intimate. The Familiar distribution is just available as jffs2 image. To extract the Familiar distribution from the jffs2 image I needed to build a new kernel for the host system (jffs2 support is normally not enabled, experimental).

Because I combined Familiar and Intimate I had to clean up the system after the assembly. The result is a well configured and clean Intimate Linux image.

2.5.2 Setup and Test Hardware

The goal here was to configure and test various hardware devices and to gain experience with the system.

I started with the setup of the PCMCIA network cards. The reason for that was to be able to upgrade the system via the internet (the Intimate image was from late 2001). I compiled the needed kernel modules and also modules from the PCMCIA package, because not all available devices are supported form the drivers included in the kernel. There was the usual setup necessary like make the kernel load the right driver and then configure the network. A funny problem was the ssh client of Intimate which was not able to validate on valid.ethz.ch (for the wireless card). So I had to upgrade the system and ssh via ethernet, after that wireless and validation worked.

Next was to set up USB. This needed some tweaking, e. g. a USB manager that keeps track of hotplugged devices have been installed. I also had to "outsource" some drivers into modules, so that the USB manager has control over them. Tested devices are a keyboard, a mouse, a hub and two sound devices. From the sound devices one works well (the Griffin iMic) and the other (Telex USB Audio Device) does not work because of a known driver issue (look at http://www.qbik.ch/usb/devices/showdev.php?id=854).

The WearARM serial ports have also been tested. The two amplified ports (RS232 like) are working, the third (not amplified) caused a "kernel Oops". Since the ARM serial driver of kernel version 2.4.5 does not support the device file system the /linuxrc script needs to set up the device nodes at boot time.

The PS/2 keyboard and the twiddler are working without problems. The PS/2 mouse does not work without problems with gpm. One needs to restart it several times to make it work combined with keystrokes on the keyboard. To make sure that this is not a setup problem I checked it also with the old WearARM Linux, the result was the same.

2.5.3 Building Opie and Qt/Embedded

The goal of this part was to assemble Opie and Qt/Embedded for the WearARM.

Because I should port an application to the WearARM I built the Opie and Qt/Embedded for the x86 and the ARM platform to have the same environment on the development machine (a PC).

It was very difficult to put together a working Opie version. Opie is not yet released, so there is just the CVS version available. I needed many iterations, till I had a working system ready. Problems were that the CVS sometimes did not compile at all, that it was not clear what other library versions are required to compile and run it and much more. This required a lot of research in the mailing lists and trial and error. Finally I ended up with a nice version for x86 and ARM. I also built a version of Konqueror/Embedded. The problem was now that the ARM version did not work stable because of the framebuffer. All applications that use the framebuffer seem to crash the system at least at the second startup, and sometimes the filesystem gets damaged. This is a kernel problem (look up the "Problems" section).

2.5.4 The Framebuffer

The goal was now to collect more information about the framebuffer and the related system crashes.

I did some tests to get more information about the framebuffer problem. I have written some small test programs and run some available standard tests. I also searched the ARM−Linux mailing lists and studied the current kernel source. The standard tests showed that the memory timings are set up correct and the visible memory is accessible without problems.

With my test programs I was able reproduce the framebuffer memory problem. A detailed description is in the "Problems" section. My final conclusion about that error is that on munmap() the framebuffer pages are accidently released by the kernel, so that the applications or the kernel uses that memory for other stuff. But the framebuffer memory gets still updated so the memory becomes corrupted. This explains the filesystem damages and the crashes.

2.6 Conclusions

The main goals of this semester thesis were to make the Intimate Linux distribution work on the WearARM, configuration and test of various hardware devices and the documentation of the system. The Intimate system has been successfully adopted and configured, several images are available. Several kernel and hardware issues have been discovered.

There are also issues in the way people work on the WearARM. They discover problems and try to fix them but do not document their work well in a defined manner. This causes that in fact nobody really knows what has been already tried out to fix a problem and causes duplicated uncontrolled work. I would recommend (even if there is just one person working on it) to set up a versioning system to keep track of changes (e. g. CVS) and perhaps a bug and issue tracking system (e. g. Bugzilla). To make this tools really work you should also define the processes (written down) how to use this tools and track down problems.

I would like to mention here some remaining problems which should be addressed to get the most out of the WearARM. The major problems are related to the kernel memory management, like the framebuffer problem which is mentioned above. Since we use a quite outdated kernel (2.4.5) I recommend to port a newer kernel from the stable branch (2.4.20 at the time of writing) to the WearARM. There are also some other problems remaining, like the non working Telex USB Audio Device, the non working serial port and the PS/2 mouse problem. You can find details about these subjects in the "Problems" section.

I have learned a lot about the Linux boot process and the setup of various devices within Linux. A nice aspect was also the crosscompiling and to work with an interesting target platfrom. The closer look on the memory management was also quite interesting.

3 Intimate Linux Installation

3.1 Introduction

This section shows you how to install the available Intimate disk images on your WearARM. Intimate is a modified Debian Linux distribution for ARM based devices like the IPAQ. There are three images available:

- im−0.1.2: Minimal system, ideal to build your custom installation.
- im−0.1.2−with−apps: im−0.1.2 with X libraries, Emacs, Vim, Mutt, GCC, G++, header files, xosview, lsof, hdparm, make, autoconf, automake and libtool.
- im−0.1.2−with−opie: im−0.1.2−with−apps with Konqueror/Embedded, Qt/Embedded and Opie.

To become familiar with Intimate I recommend you to start with im−0.1.2−with−apps. When you want to set up your custom image you should start with im−0.1.2, so you do not need to remove too many packages. The Qt/Embedded library in im−0.1.2−with−opie is a special stripped down version needed by Opie and Qtopia. If you need a full featured Qt/Embedded library, you can get and install it via apt−get. All these images contain the kernel with the memory management fix to allow access to the framebuffer.

3.2 Requirements

To install Intimate you require the following:

- A computer running Linux with a working PCMCIA slot.
- Root access.
- A microdrive.
- The Intimate image of your choice.
- A working WearARM.
- You need to be familiar with Linux.

3.3 Installation

3.3.1 Partitioning

Create with fdisk one big partition on your microdrive, set the partitions system id to Linux. The images are set up to run without a swap partition. Probably your drive is /dev/hde1. Fdisk is interactive and contains online help. Plug in your drive and do

```
# fdisk /dev/hde
```

If your drive is not /dev/hde you can look the name up in the console log or in the /proc/ide directory.

3.3.2 Formatting the Drive

Now format the drive with

```
# mke2fs /dev/hde1 −b 4096
```

3.3.3 Unpack and Install the Image

Unpack the image with

```
# tar xfvz im−0.1.2.tgz
```

Mount the microdrive on an available mountpoint in your system, e. g.

```
# mount /dev/hde1 /mnt/camera
```

Copy the image to the drive:

```
# cd im-0.1.2
# cp -a * /mnt/camera
# umount /mnt/camera
# cardctl eject
```

Now you can eject the drive.

3.3.4 Starting the WearARM

Connect the WearARM with a suitable power source. You have to run the WearARM with a voltage in the range from 6 to 10 Volts. Check that before you connect the WearARM! Connect the WearARM serial port 1 (/dev/ttySA0) with your computer, so that you can check the console log during the boot process. You can use the "minicom" application, set up the serial speed to 57600 8N1. Connect also a keyboard, mouse and monitor if you like. Plug in the drive and switch the system on. When the system is up you can log in as root with password "rootme" on both serial lines and on both framebuffer consoles. You should change the password now with "passwd".

3.3.5 Troubleshooting

I can not see anything in minicom.

- Is the serial speed set up to 57600 8N1? Check it in minicom.
- Is the serial port enabled on your system (on notebooks they are sometimes switched off in the BIOS to save power). Check the serial settings in your BIOS.
- Did you connect the correct cable from the WearARM? Try the other port.

The keyboard does not work.

- On some WearARM's the connectors for the mouse and keyboard are exchanged, so try to connect it the other way round.
- Check if the connector is plugged in well.

The screen flickers on system activity.

- That is perfectly normal. The screen data is held in main memory, so if another device is using the system bus the screen can not be updated.

4 The Operating System

4.1 Introduction

This chapter shows you how to build the WearARM Intimate Linux from scratch, how to configure and maintain the system. Finally it will show you how to use and build Opie and Konqueror/Embedded. "Intimate" is a Debian based Linux distribution which is built for ARM based handheld devices like the iPAQ. Documentation and the distribution itself is available from http://www.handhelds.org.

4.2 Building WearARM Intimate Image From Scratch

4.2.1 Overview

This chapter shows you how to build the WearARM Intimate Linux from scratch. But keep in mind that this will not work forever, because all required components are under development. In this chapter I will call the WearARM the "target" and the machine (in my case a PC or notebook) to compile and assemble the system the "host".

4.2.2 Requirements

- Bootstrap Familiar Linux image for the iPAQ from http://www.handhelds.org (I used version 0.6.1, file bootstrap−v061−01.jffs2)
- The Intimate Linux snapshot (tarball from late 2001, file intimate20011113.tar.bz2).
- A host system with Linux and root access.
- The WearARM Linux kernel with sources.
- The ramdisk from the im−0.1.2 image.

4.2.3 Extract the Familiar Image

The Familiar distribution comes as jffs2 filesystem image. To extract this you need to build and install a new kernel for your host system with the following options enabled:

- CONFIG_MTD=m
- CONFIG_MTD_PARTITIONS=m
- CONFIG_MTD_CONCAT=m
- CONFIG_MTD_REDBOOT_PARTS=m
- CONFIG_MTD_CHAR=m
- CONFIG_MTD_BLOCK=m
- CONFIG_MTD_BLOCK_RO=m
- CONFIG_MTD_BLKMTD=m
- CONFIG_FTL=m
- CONFIG_NFTL=m
- CONFIG_JFFS2_FS=m
- CONFIG_JFFS2_FS_DEBUG=0

You need to create the device nodes for the new drivers (if not yet available):

```
# mkdir /dev/mtdblock
# cd /dev/mtdblock
# mknod 0 b 31 0
# mknod 1 b 31 1
# mknod 2 b 31 2
```

After rebooting your host you should be able to mount the image. You can do this probably via the loopback device or the way I did. I first put the image on an empty disk partition:

```
# dd if=imagename of=/dev/hda6
```

After that we need to load some modules and to mount the image:

```
# modprobe mtdblock
# modprobe blkmtd device=/dev/hda6
# mount −t jffs2 /dev/mtdblock/0 /mnt
```

4.2.4 Assemble Intimate and Familiar

Before we start let us have a look on why we need Familiar and Intimate, and not just Intimate. On the iPAQ the bootprocess mounts the Familiar image which is in flash memory. Familiar loads the PCMCIA and disk drivers, and after that the Intimate bootprocess gets started. Intimate is located on a PCMCIA disk. Because of that there are important things like PCMCIA support missing in the Intimate image. Now make a copy of the mounted Familiar image:

```
# cd
# mkdir familiar
# cp −a /mnt/* familiar
```

Extract the Intimate snapshot (command depends on the linux distribution of the host system):

```
# tar xfvI intimate20011113.tar.bz2
```

Now we create a new directory and put in first the Familiar distribution, and after that we copy Intimate over it.

```
# mkdir im−0.0.0
# cd im−0.0.0
# cp −a ~/familiar/* .
# cp −a ~/intimate/* .
```

Now you can clean up the new system. Look at the Intimate original, which files in /etc/init.d, /etc/rcS.d and /etc/rc2.d belong to Intimate. Disable or remove the stuff you do not need from Familiar (at least you need PCMCIA support).

4.2.5 The Ramdisk

Put the ramdisk from the im−0.1.2 (/boot/ramdisk.gz) in the boot directory. If you want to edit the ramdisk you should copy it somewhere, uncompress and mount it via the loopback device:

```
# gzip −d ramdisk.gz
# mount ramdisk /mnt/ramdisk −o loop
```

4.2.6 The Kernel

Put the kernel from the im−0.1.2 (/boot/zImage) into the boot directory. Copy also the kernel modules directory /lib/modules/2.4.5−rmk4−np1 to the lib/modules directory. Correct configured kernel sources are available on the CD. Changes to the original WearARM kernel are the configuration, a new serial driver (linux/drivers/char/serial_sa1100.c) and some boot parameters. The boot parameters are located in linux/init/main.c:

```
static char mycommand[50] ="console=ttySA0,57600 init=/linuxrc";
```

That is because the kernel does not accept boot parameters from any other source in the current state (passing arguments from the bootloader nor from the Makefile works).

4.2.7 Remaining Tasks

Now we have to apply some changes to take the Device File System into account. Create the /boot/mnt/rdisk directory in the image and modify /etc/rcS.d/S01unmountboot, so that it looks like this :

```
#!/bin/sh
PATH="/sbin:/bin:/usr/sbin:/usr/bin"
umount /boot/mnt/rdisk/proc
umount /boot/mnt/rdisk/dev
umount /boot/mnt/rdisk
```

Change also in /etc/rc6.d/S40umountfs the following line from

```
umount -tnoproc $FORCE -a -r
```

to

```
umount -tnoproc,nodevfs $FORCE -a -r
```

Before you copy the image to the PCMCIA disk you should change the configuration files according to the next chapter. After that you should be able to boot up the system. I do not describe the whole cleanup of the image, because you have an image for the current system. What you can do next is connect to the internet and do an upgrade of the whole system (remember, the Intimate snapshot was from late 2001). A first test to check if the system is assembled well is running ldconfig. If the system works after that without errors the fitting libraries and applications are in place. Next you should clean up the system, like removing packages you do not need and customize the configuration. Depending on the Intimate snapshot and Familiar release you take it might be necessary to remove old files from Familiar manually (with the files I used the libraries did match after the upgrade, but that is not necessarily the case).

4.3 Configuration Files

4.3.1 Overview

This chapter shows you which configuration files have been changed.

4.3.2 /etc/inittab

The changes in inttab are:

- Reboot on CTRL-ALT-DELETE.
- Set up ttySA0 and ttySA2 to enable login.

```
# The default runlevel.
id:2:initdefault:

# Boot-time system configuration/initialization script.
# This is run first except when booting in emergency (-b) mode.
si::sysinit:/etc/init.d/rcS

# What to do in single-user mode.
~~:S:wait:/sbin/sulogin

# /etc/init.d executes the S and K scripts upon change
# of runlevel.
```

```
#
# Runlevel 0 is halt.
# Runlevel 1 is single-user.
# Runlevels 2-5 are multi-user.
# Runlevel 6 is reboot.

l0:0:wait:/etc/init.d/rc 0
l1:1:wait:/etc/init.d/rc 1
l2:2:wait:/etc/init.d/rc 2
l6:6:wait:/etc/init.d/rc 6
# Normally not reached, but fallthrough in case of emergency.
z6:6:respawn:/bin/sh

1:2345:respawn:/sbin/getty 38400 tty1 vt100
2:2345:respawn:/sbin/getty 38400 tty2 vt100

# What to do when the power fails/returns.
pf::powerwait:/etc/init.d/powerfail start
pn::powerfailnow:/etc/init.d/powerfail now
po::powerokwait:/etc/init.d/powerfail stop

#T0:2:respawn:/sbin/getty -L tts/0 115200 vt100
T0:2:respawn:/sbin/getty -L ttySA0 57600 vt100
T1:2:respawn:/sbin/getty -L ttySA2 57600 vt100
#T1:3:respawn:/usr/bin/stowd

# enable ctrl-alt-del
ca::ctrlaltdel:/sbin/shutdown -t3 -r now
```

4.3.3 /etc/fstab

The changes in fstab are:

- Take into account devfs.

```
/dev/hda1       /               ext2                    defaults 1 1
none            /proc           proc                    defaults 0 0
none            /dev/pts        devpts                  mode=0622 0 0
none            /dev            devfs                   defaults 0 0
```

4.3.4 /etc/securetty

The changes in securetty are:

- Enable logins from ttySA0, ttySA2, tty1 and tty2.

```
# /etc/securetty: list of terminals on which root is allowed to login.
# See securetty(5) and login(1).
tty1
tty2
tty3
tty4
tty5
tty6
tty7
tty8
tty9
tty10
tty11
tty12
tts/0
ttySA0
```

```
ttySA2
```

4.3.5 /etc/pcmcia/*

The following files has been overwritten with copies from the newer pcmcia package pcmcia−cs−3.1.34 (in order to support the D−Link DFE−670TXD ethernet card).

- /etc/pcmcia/config.
- /etc/pcmcia/config.opts.

This works because there is just a new association between the name of the card and the driver to load.

4.4 The Device File System

The Device File System is a filesystem which is mounted on the /dev directory. Instead of creating device nodes in the /dev directory manually, the Device File System does this automatically for you (if the device driver supports it). The most obvious advantage is that in /dev are just device nodes for existing devices, e. g. if you hotplug an USB device the device appears automatically in the /dev directory. This makes it easy to see which devices are actually recognized by the kernel. Be aware of the changed locations of the device nodes.

4.5 Intimate System Maintenance and Usage

4.5.1 The Package Manager

Because Intimate is a Debian distribution, it works with the Debian package management system. I will just mention the commands here and some examples. You can get more information the usual way (manual pages or −−help option).

- apt−get
- apt−cache
- dpkg
- apt−*
- dselect

Here are some useful examples:
Serching package database for gcc: `apt−cache search gcc`
Update package database from the net: `apt−get update`
Install package gaga: `apt−get install gaga`
List files from package usbutils: `dpkg −L usbutils`
Description of package usbutils: `apt−cache show usbutils`
List installed packages: `apt−show−versions`
To which package belongs file gaga: `dpkg −S gaga`
Cleanup downloaded packages: `apt−get clean, apt−get autoclean`

Detailed documentation of the Debian package manager can be found on
http://www.debian.org/doc/manuals/apt−howto.

4.5.2 System Upgrade

To upgrade the whole system run `apt−get dist−upgrade`. Just do that if you know what you are doing, you can easily mess up the system. It would also make sense to have a formal (written down, defined) configuration management and issue tracking process, so that all people run on the same image. If you produce such an "official" image take as base a clean (clean means never crashed) image, do the upgrade and shut down the system

to save it somewhere.

4.5.3 Filesystem Check

Put the microdrive into a laptop and do a forced filesystem check, e. g.:

```
# fsck −f /dev/hde1
```

4.5.4 Emergency Shutdown

In case your WearARM hangs try the following shutdown procedure:

- ALT−PRINTSCREEN−s (sync filesystem)
- ALT−PRINTSCREEN−u (remount read only)
- ALT−PRINTSCREEN−b (reboot)
- Switch the WearARM off
- Remove microdrive and check the filesystem

If there is too much filesystem damage, format the drive and put the image on it. All features of magic sysrequest are documented in linux/Documentation/sysrq.txt in the linux kernel source tree (in any version).

4.6 Applications

4.6.1 Applications on im−0.1.2−with−apps

Here is a list of the additional installed applications of the im−0.1.2−with−apps Intimate image.

- gcc 3.2
- glibc header files
- g++ 3.2
- vim
- emacs
- mutt
- xosview
- lsof
- hdparm
- make
- autoconf
- automake
- libtool

4.6.2 Applications on im−0.1.2−with−opie

Here is the list of additional packages (added to im−0.1.2−with−apps) in the im−0.1.2−with−opie Intimate image.

- openssl 0.9.7
- libmng
- libxine
- Opie
- Konqueror/Embedded

4.6.3 Starting Opie

Because of kernel framebuffer memorymapping problems Opie will crash after the first munmap() call. If you want to play with it anyway, install the im−0.1.2−with−opie image and do after system startup:

```
# cd /
# touch .Qtopia-ignore
# cd
# . opie
# export QWS_SIZE=640x480
# qpe >& /dev/null
```

All the QWS_* variables are explained on http://doc.trolltech.com/3.1/emb−envvars.html.

4.6.4 Starting Konqueror/Embedded

Because of kernel framebuffer memorymapping problems Konqueror/Embedded will crash during shutdown. If you want to play with it anyway, install the im−0.1.2−with−opie image and do after system startup:

```
# cd
# . opie
# konqueror -qws >& /dev/null
```

4.6.5 Building Opie for the WearARM

This section will guide you trough the build process of Opie for the WearARM. Opie is a graphical environment for Linux driven PDA's. For crosscompiling I used the arm−linux−toolchain−post−2.2.13.tar.gz toolchain. It contains a crossplatform compiler, linker, archiver and libraries for the ARM target platform. You can download it from ftp://ftp.handhelds.org/pub/linux/arm/toolchain. I will assume that our building directory is ~/opie−arm and the users name is berni.

You need to download the following packages into ~/opie−arm:

- zlib−1.1.4.tar.bz2
- libpng−1.2.5.tar.bz2
- jpegsrc.v6b.tar.gz and libjpeg−6b−arm.patch
- libmng−1.0.4.tar.gz
- qt−embedded−2.3.4−snapshot−20030114.tar.gz
- e2fsprogs−1.32.tar.gz
- Linux−PAM−0.77.tar.gz
- xine−lib−1−beta2.tar.gz
- freetype−2.1.3.tar.bz2
- tmake−1.8.tar.gz
- Do a CVS checkout of Opie according to www.handhelds.org in the ~/opie−arm directory.

4.6.5.1 Installing the Toolchain

Install the skiff toolchain in the / directory and do

```
# cd /skiff
# chown -R berni:users * (so you can write here)
$ cd ~/opie-arm
```

4.6.5.2 Environment

Put the following in a file called env−arm.sh located in ~/opie−arm:

```
#!/bin/bash
export OPIEDIR=`pwd`/opie
export QTDIR=`pwd`/qt−2.3.4−arm
export PATH=$PATH:`pwd`/tmake−1.8/bin
export TMAKEPATH=`pwd`/tmake−1.8/lib/qws/linux−arm−g++
export PATH=$PATH:/skiff/local/bin
```

and run or source it with

```
$ . env−arm.sh
```

4.6.5.3 Install tmake

You can download tmake from ftp://ftp.trolltech.com/freebies/tmake. It is a tool from Trolltech to create and maintain makefiles. It is used by the Opie build scripts. Extract the package in ~/opie−arm.

```
$ tar xfvz tmake−1.8.tar.gz
```

4.6.5.4 Building zlib

Download zlib from http://www.gzip.org/zlib. The zlib provides data compression and decompression support. Extract the package in ~/opie−arm.

```
$ tar xfvI zlib−1.1.4.tar.bz2
$ cd zlib−1.1.4
$ ./configure −−shared −−prefix=/skiff/local/arm−linux −−libdir=/skiff/local/arm−linux/lib \
  −−includedir=/skiff/local/arm−linux/include
```

Replace all gcc occurrences in the Makefile with /skiff/local/bin/arm−linux−gcc.

```
$ make
$ make install
```

Create symbolic links libz.so and libz.so.1 in /skiff/local/arm−linux/lib to /skiff/local/arm−linux/lib/libz.so.1.1.4.

4.6.5.5 Building libpng

Download libpng from http://www.libpng.org/pub/png/libpng.html. It provides support for handling images in the "Portable Network Graphics" format. Extract the package in ~/opie−arm.

```
$ tar xfvI libpng−1.2.5.tar.bz2
$ cd libpng−1.2.5/scripts
$ cp makefile.linux makefile.arm
```

Now edit makefile.arm, change CC and prefix

```
CC=/skiff/local/bin/arm−linux−gcc
prefix=/skiff/local/arm−linux

$ cd ..
$ make −f scripts/makefile.arm
$ make −f scripts/makefile.arm install
```

In case the creation of the symbolic links fails create them manually.

4.6.5.6 Building libjpg

Download libjpg from http://www.ijg.org. It provides support for handling JPEG images. Extract the package in ~/opie−arm. The patch form the ARM platform is available from http://www.lunar−linux.org/cgi−bin/viewcvs.cgi/moonbase/graphics/jpeg.

```
$ tar xfvz jpegsrc.v6b.tar.gz
$ patch −p0 < libjpeg−6b−arm.patch
$ cd jpeg−6b
$ CC=/skiff/local/bin/arm−linux−gcc ./configure −−enable−shared −−enable−static \
  −−target=arm−linux −−host=i686−pc−linux−gnu −−prefix=/skiff/local/arm−linux \
  −−libdir=/skiff/local/arm−linux/lib −−includedir=/skiff/local/arm−linux/include
$ CC="/skiff/local/bin/arm−linux−gcc" ltconfig ltmain.sh −−no−verify i686−pc−linux−gnu
$ make
$ mkdir /skiff/local/arm−linux/man
$ mkdir /skiff/local/arm−linux/man/man1
$ make install
```

Create a symbolic link libjpeg.so.62.0.0 to libjpeg.so in /skiff/local/arm−linux/lib.

4.6.5.7 Building libmng

Download libmng from http://www.libmng.com/downloadpublic.html. It provides support for handling images in the "Multiple−image Network Graphics" format. The format is a PNG−like image format supporting multiple images, animation and transparent JPEG's. Extract the package in ~/opie−arm.

```
$ tar xfvz libmng−1.0.4.tar.gz
$ cd libmng−1.0
$ CC=/skiff/local/bin/arm−linux−gcc ./configure −−enable−shared −−enable−static \
  −−target=arm−linux −−host=i686−pc−linux−gnu −−prefix=/skiff/local/arm−linux \
  −−libdir=/skiff/local/arm−linux/lib −−includedir=/skiff/local/arm−linux/include \
  −−with−zlib=/skiff/local/arm−linux −−with−jpeg=/skiff/local/arm−linux
$ make
$ make install
```

4.6.5.8 Building QT/Embedded for Opie

QT/Embedded is a GUI toolkit for embedded systems. It supports the handling of input devices and drawing windows and widgets to framebuffer devices. Download it from ftp://ftp.trolltech.com/qt/snapshots. Unpack QT/Embedded in ~/opie−arm and apply the patches from the Opie project. If you do that with a newer version check the Opie documentation. Because we need also a static linked uic for the host (the x86 box) additional work is necessary. The uic ("User Interface Compiler") is needed to convert form designs stored in .ui files into C++ header and source files.

```
$ tar xfvz qt−embedded−2.3.4−snapshot−20030114.tar.gz
$ mv qt−embedded−2.3.4−snapshot−20030114.tar.gz qt−2.3.4−arm
$ cd $QTDIR
$ cat $OPIEDIR/qt/qte234−for−opie091−config−linux−x86−g++−shared.patch | patch −p0
$ cat $OPIEDIR/qt/qte234−for−opie091−gfxraster.patch | patch −p0
$ cat $OPIEDIR/qt/qte234−for−opie091−listview.patch | patch −p0
$ cat $OPIEDIR/qt/qte234−for−opie091−makefile.patch | patch −p0
$ cat $OPIEDIR/qt/qte234−for−opie091−menubar.patch | patch −p0
$ cat $OPIEDIR/qt/qte234−for−opie091−override.patch | patch −p0
$ cat $OPIEDIR/qt/qte234−for−opie091−popupmenu.patch | patch −p0
$ cat $OPIEDIR/qt/qte234−for−opie091−setpalette.patch | patch −p0
$ cat $OPIEDIR/qt/qte234−for−opie091−style.patch | patch −p0
$ cat $OPIEDIR/qt/qte234−for−opie091−unpolish.patch | patch −p0
$ cp $OPIEDIR/qt/qconfig−qpe.h $QTDIR−arm/src/tools
$ cd ..
$ cp −a qt−2.3.4−arm qt−2.3.4
```

Now we need also an environment setup for the x86 build. Put the following in the file env.sh:

```
#!/bin/bash
export OPIEDIR=`pwd`/opie
export QTDIR=`pwd`/qt-2.3.4
export LD_LIBRARY_PATH=$QTDIR/lib:$LD_LIBRARY_PATH
export PATH=$PATH:`pwd`/tmake-1.8/bin
export TMAKEPATH=`pwd`/tmake-1.8/lib/qws/linux-x86-g++
```

Source it and compile uic

```
$ . env.sh
$ cd $QTDIR
$ ./configure -qconfig qpe -qvfb -system-jpeg -system-libpng -system-zlib -gif \
  -system-libmng -no-vnc -no-xft -static
```

Answer the questions with "yes" and "8,16".

```
$ make
$ cd tools/designer/uic
$ make
$ cp $QTDIR/bin/uic ~/opie-arm/qt-2.3.4-arm/bin
$ cd ~/opie-arm
$ . env-arm.sh
```

Now we are ready to compile QT/Embedded for the ARM.

```
$ cd $QTDIR
$ ./configure -qconfig qpe -xplatform linux-arm-g++ -no-qvfb -system-jpeg \
  -system-libpng -system-zlib -gif -system-libmng -no-vnc -no-xft
```

Answer the questions with "yes" and "8,16". After that run make.

4.6.5.9 Building libuuid

The libuuid is part of the e2fsprogs-1.32.tar.gz package. Download it from http://e2fsprogs.sourceforge.net. The libuuid library is used to generate unique identifiers for objects that may be accessible beyond the local system. Extract the package in ~/opie-arm.

```
$ tar xfvz e2fsprogs-1.32.tar.gz
$ cd e2fsprogs-1.32
$ CC=/skiff/local/bin/arm-linux-gcc ./configure --target=arm-linux \
  --host=i686-pc-linux-gnu --prefix=/skiff/local/arm-linux \
  --libdir=/skiff/local/arm-linux/lib --includedir=/skiff/local/arm-linux/include \
  --enable-elf-shlibs
$ make
$ cd lib/uuid
$ make install
```

4.6.5.10 Building libpam

Download libpam from http://www.kernel.org/pub/linux/libs/pam. It provides libraries that handle the authentication of applications and services on the system. Extract the package in ~/opie-arm.

```
$ tar xfvz Linux-PAM-0.77.tar.gz
$ cd Linux-PAM-0.77
```

Edit configure.in. Change all occurrences of gcc to arm-linux-gcc and all ld to arm-linux-ld.

```
$ autoconf
```

```
$ CC=/skiff/local/bin/arm-linux-gcc ./configure --target=arm-linux \
  --host=i686-pc-linux-gnu --prefix=/skiff/local/arm-linux \
  --libdir=/skiff/local/arm-linux/lib --includedir=/skiff/local/arm-linux/include \
  --enable-includedir=/skiff/local/arm-linux/include
$ make
$ make install
```

4.6.5.11 Building libxine

Download libxine from http://xinehq.de. It provides decoders for many multimedia file formats like MP3 or AVI. Extract the package in ~/opie-arm.

```
$ tar xfvz xine-lib-1-beta2.tar.gz
$ cd xine-lib-1-beta2
```

Change in ./src/input/librtsp/rtsp.c <time.h> to <sys/time.h>.

```
$ CC=/skiff/local/bin/arm-linux-gcc LD=/skiff/local/bin/arm-linux-ld \
  ./configure --host=arm-linux --build=i686-pc-linux-gnu --prefix=/skiff/local/arm-linux \
  --libdir=/skiff/local/arm-linux/lib --includedir=/skiff/local/arm-linux/include \
  --without-x --disable-alsa
$ make
$ make install
$ cd src/video_out
$ make clean
$ make
$ make install
```

4.6.5.12 Building libfreetype

Download libfreetype from http://www.freetype.org. It provides services to draw fonts (glyph images). If you want to enable the patented algorithm for hinting do that according to the instructions in README.UNIX file. Extract the package in ~/opie-arm.

```
$ tar xfvI freetype-2.1.3.tar.bz2
$ cd freetype-2.1.3
$ CC=/skiff/local/bin/arm-linux-gcc LD=/skiff/local/bin/arm-linux-ld \
  ./configure --host=arm-linux --build=i686-pc-linux-gnu \
  --prefix=/skiff/local/arm-linux --libdir=/skiff/local/arm-linux/lib \
  --includedir=/skiff/local/arm-linux/include
$ make
$ make install
$ cd /skiff/local/arm-linux
$ ln -s libfreetype.so.6.3.2 lib/libfreetype.so
```

4.6.5.13 Building openssl

Download openssl from http://www.openssl.org. It provides an impementation of the "Secure Sockets Layer" and "Transport Layer Security" protocols as well as a full-strength general purpose cryptography library. Extract the package in ~/opie-arm.

```
$ tar xfvz openssl-0.9.7.tar.gz
$ cd openssl-0.9.7
$ CC=/skiff/local/bin/arm-linux-gcc LD=/skiff/local/bin/arm-linux-ld \
  ./Configure linux-elf-arm shared --prefix=/skiff/local/arm-linux \
  -L/skiff/local/arm-linux/lib -ldl
```

Now edit the Makefile, change all occurrences of gcc, ar and ranlib to the tools with the prefix /skiff/local/bin/arm-linux-, e. g. change gcc to /skiff/local/bin/arm-linux-gcc.

```
$ make
$ make install
```

4.6.5.14 Building Opie

```
$ cd $OPIEDIR
$ ./configure −xplatform linux−arm−g++
$ make
$ cp bits/mman.h /skiff/local/arm−linux/include/bits/mman.h
$ make
$ ln −s libpam.so.0 lib/libpam.so
$ make
$ chmod u+s /home/berni/opie−arm/opie/bin/opiealarm
$ chown root /home/berni/opie−arm/opie/bin/opiealarm
$ mkipks
$ cd /tmp
$ mkdir opie
$ cd opie
$ mv $OPIEDIR/*.ipk .
$ for f in *.ipk; do ar −x $f data.tar.gz $f; tar xfvz data.tar.gz; rm *.gz; done
$ rm *.ipk
```

Now you can pick up the Opie installation from /tmp/opie.

4.6.6 Building Konqueror/Embedded for ARM

Konqueror/Embedded is a webbrowser for embedded devices. To follow the instructions below you need to build Opie first, so that everything is in place and the environment is set up well. Download konqueror−embedded−snapshot−20021229.tar.gz from http://konqueror.org to ~/opie−arm.

```
$ cd ~/opie−arm
$ tar xfvz konqueror−embedded−snapshot−20021229.tar.gz
$ cd konqueror−embedded−snapshot−20021229
$ CC=/skiff/local/bin/arm−linux−gcc LD=/skiff/local/bin/arm−linux−ld ./configure \
  −−host=arm−linux −−build=i686−pc−linux−gnu −−libdir=/skiff/local/arm−linux/lib \
  −−includedir=/skiff/local/arm−linux/include −−prefix=/opt/QtPalmtop \
  −−enable−qt−embedded −−enable−qtopia −−enable−qpe −−with−qt−dir=$QTDIR \
  −−with−qtopia−dir=$OPIEDIR
$ make
$ export PATH=$PATH:$OPIEDIR/scripts
$ make ipkg
$ cp ./konq−embed/ipkg/konqueror_snapshot_20021229_arm.ipk /tmp
$ cd /tmp
$ ar −x konqueror_snapshot_20021229_arm.ipk data.tar.gz
```

Konqueror/Embedded is now ready in data.tar.gz.

4.6.7 Opie for x86

To have a full crossdevelopment environment we need also the same Opie version for the x86 box. The working directory is this time ~/opie−x86. I will just mention the differences to the ARM build. One major difference is that you can install most of the libraries from your distribution (you do not need to complie e. g. zlib, libpng, ...). Configure the source with

```
$ ./configure −xplatform linux−x86−g++
```

After that change all occurrences of "−DHAVE_MMX" in core/multimedia/opieplayer/libmpeg3/Makefile to "−UHAVE_MMX". Edit also the noncore/games/solitaire/Makefile, remove all occurrences of −fno−default−inline. Now you can run make. It will stop with an error in opieplayer2, simply copy the last linking command to the command line and link it manually with additional "−L/usr/X11R6/lib −lX11 −Xext" arguments.

After that you can rerun make. You will be able to start opie directly from the bin directory, but I recommend you to create a package.

4.6.8 Building Konqueror/Embedded for x86

You have to do this after the build of Opie, so that all is in place and the environment is set up well. Download konqueror−embedded−snapshot−20021229.tar.gz to ~/opie−x86.

```
$ cd ~/opie−x86
$ tar xfvz konqueror−embedded−snapshot−20021229.tar.gz
$ cd konqueror−embedded−snapshot−20021229
$ ./configure −−enable−qt−embedded −−enable−qtopia −−enable−qpe −−with−qt−dir=$QTDIR \
  −−with−qtopia−dir=$OPIEDIR −−prefix=/opt/QtPalmtop
$ make
$ make
$ export PATH=$PATH:$OPIEDIR/scripts
$ make ipkg
$ cp ./konq−embed/ipkg/konqueror_snapshot_20021229_arm.ipk /tmp
$ cd /tmp
$ ar −x konqueror_snapshot_20021229_arm.ipk data.tar.gz
```

Konqueror/Embedded is now ready in data.tar.gz.

5 The Boot Process

5.1 Introduction

This chapter introduces the bootprocess of the WearARM.

5.2 Overview

The boot process consists of four major stages:

- The bootloader starts up. It is responsible for basic hardware setup, loading the kernel image and the initial ramdisk from the harddisk.
- The kernel starts up. It does further hardware setup and mounts the ramdisk.
- The /linuxrc script on the ramdisk gets executed. It sets up the filesystems and devices. It changes the "/" directory from the ramdisk to the harddisk. After that it executes /sbin/init (from the harddisk) in its process context.
- Init starts processes according to /etc/inittab (e. g. login).

The ramdisk is needed because the harddisk is attached to the PCMCIA bus. To set up the PCMCIA harddisk one has to run the "cardmgr" utility. But because we need the utility before we can access the harddisk we have to get it from another location, and that is the ramdisk.

5.3 The Bootloader

The bootloader (a modified version of blob) is located in the flash memory of the WearARM. When the system gets switched on, the execution of the bootloader code starts (like the BIOS in PC's). It sets up the hardware and loads the compressed ramdisk (/boot/ramdisk.gz) and the kernel (/boot/zImage) from the PCMCIA harddrive into the RAM. It should also set up the boot parameter block. The boot parameter block is a chunk of memory where the bootloader can fill in information about the hardware (e. g. the size and the base addresses of the memory banks) and boot parameters for the kernel (at the moment the memory banks are set up in the linux−2.4.5−rmk4−np1/arch/arm/mach−sa1100/weararm.c during kernel initialization, but this should be done by the boot loader according to the ARM kernel developers). After that the bootloader passes the control to the kernel image. The kernel image decompresses itself and starts up.

Optionally you can enter boot parameters via a command prompt on startup. The whole parameter passing to the kernel does not work at all. The parameter block seems to hold the wrong information (e. g. just 16 MBytes RAM) and the kernel ignores the parameters anyway (look up the above mentioned file weararm.c). This should become fixed in the furture.

5.4 Starting the Initial Ramdisk

After the bootloader passes the control to the kernel, the kernel starts up, mounts the ramdisk and executes the /linuxrc script. At this stage the kernel is not able to access the PCMCIA harddisk. The linuxrc script loads first the driver for the harddisk and mounts it to /mnt. After that the root filesystem gets exchanged via the pivot_root system call, so /mnt becomes / and / becomes /mnt/boot/mnt/rdisk. Now the proc and the device filesystems are mounted. On the mounted device file system the script creates the missing nodes (because some drivers does not support devfs) and some compatibility links. Finally init is executed in the same process context via exec. Because the / is now the PCMCIA harddisk the system on the disk becomes started.

```
#!/bin/bash
echo "Starting linuxrc"
export PATH=/bin:/sbin:/usr/bin
```

```
echo "Mounting /proc filesystem"
mount −t proc /proc /proc

echo "Mounting root filesystem"
/sbin/cardmgr −o
/sbin/pivot_root /mnt /mnt/boot/mnt/rdisk

echo "Remounting proc and dev"
mount −t proc /proc /proc
mount −t devfs /dev /dev

echo "Setting up devices"
ln −s /dev/vc/0 /dev/tty0
ln −s /dev/vc/1 /dev/tty1
ln −s /dev/vc/2 /dev/tty2
ln −s /dev/vc/3 /dev/tty3
ln −s /dev/vc/4 /dev/tty4
ln −s /dev/vc/5 /dev/tty5
ln −s /dev/fb/0 /dev/fb0
ln −s /dev/sound/dsp /dev/dsp
ln −s /dev/sound/mixer /dev/mixer
ln −s /dev/v4l/video0 /dev/video0
ln −s /dev/v4l/video0 /dev/video

mknod /dev/xconsole p
mknod /dev/ttySA0 c 204 5
mknod /dev/ttySA1 c 204 6
mknod /dev/ttySA2 c 204 7

echo "linuxrc done"
echo "Executing /sbin/init"
exec /sbin/init $*
```

5.5 Init

The system on the harddisk gets started according to the /etc/inittab. From here nothing special happens, it just works like any other Linux system.

5.6 Remaining Issues

5.6.1 Reduce the Ramdisk Size

One could reduce the ramdisk size by putting just the tools used from the /linuxrc script on it. To get rid of the libraries I recommend to link the tools statically.

5.6.2 Kernel Parameter Passing

Passing parameters to the current kernel does not work, because of the implementation of linux−2.4.5−rmk4−np1/arch/arm/mach−sa1100/weararm.c and the incorrect initialized boot parameter block. One should also remove the "keepinitrd" flag to be able to release the ramdisk memory. This flag is hardcoded in weararm.c and prevents the release of the initial ramdisk.

6 Problems

6.1 Introduction

All known problems are related to the kernel or bootloader. For real usage of the system porting a new kernel is suggested, since the current one is by now quite outdated (2.4.5 instead of 2.4.20). Meanwhile there has changed a lot which could be related to the problems described below. Especially there are changes in the ARM specific memory handling routines. To reproduce the errors reboot the system, log in and run just the commands mentioned.

6.2 Memory Management

The memory management of the current (2.4.5) WearARM kernel is broken. This follows from the "kernel null pointer dereference" kernel messages, caused by the following command:

```
# find / −name "*" −exec grep "gaga" {} \;
```

The command crashes with signal 11. After that the system becomes unstable. It can not be an Intimate problem, because if the process runs out of memory the kernel should kill it.

```
# find / −name "*" | grep "gaga"
```

causes "VFS: Disk change detected on device 03:00". The filesystem gets some damage and the system becomes unstable. This could also be caused by the memory management.

6.3 System Hangs on Opie Startup

```
# cd; . opie; qpe >& /dev/null
```

causes a system hang. It seems to be a combination of the framebuffer problem discussed below and the broken memory management.

6.4 PS/2 Mouse Problem

```
# /etc/init.d/gpm restart
```

works after several runs combined with keystrokes. The kernel complains about "IRQ LOCK: IRQ68 is locking the system, disabled" and "sa1111_mask_lowirq(irq=0x44)". Candidates are the kernel or hardware.

6.5 WearARM Hangs on Reboot

This can have several reasons according to the ARM−Linux related mailing−lists. One is a kernel memory management bug which has been fixed. Other candidates are the bootloader or the hardware.

6.6 PCMCIA Access Causes Display Flickering

That is a known WearARM design issue. The flickering is caused by too long memory transfers from or to the PCMCIA bus. There is no solution available.

6.7 Framebuffer Problem

To gather more information about the system crashes and filesystem damages caused by framebuffer access I decided to run the following tests:

- Run a memory tester to find out if the memory timings are set up well.
- Run tests with allocating memory with the brk/sbrk system calls.
- Run tests with allocating memory via mmap system calls.
- Combine the above tests with framebuffer activity.

First I ran tests with a normal memory tester to check if the memory is set up correct (timings, etc). These tests passed without problems.

After that I checked with the following test programs if getting and accessing memory via brk/sbrk (membrk.c) and memory mapping (memmmap.c) works. Without touching the framebuffer the programs work without errors. In a nutshell they allocate two 16 MBytes blocks, write something to it, compare it with the values written and then release it in a loop.

The test program fb.c simply maps the framebuffer memory via mmap into its address space. After that it fills the framebuffer memory with the colour blue. Then it releases the address space with munmap.

When you now start first membrk or memmmap (does not matter which one) and then the fb program, after the termination of fb memory errors will occur, and that synchronous with the blinking cursor. The display content looks broken. That happens after the munmap, you can check that with inserting a delay before the munmap in fb.c.

Probably the kernel accidently releases (marks it as free) with the munmap the framebuffer. Because of that the application membrk/memmap gets with the next malloc the framebuffer memory and writes to it. There is the cursor blinking, which destroys the memory content of the application, so memory errors occur.

There are several candidates for that bug. Before I would put any effort in fixing that, I would take the most recent kernel and port it to the WearARM (there has changed a lot).

The membrk.c listing:

```
#include <malloc.h>
#include <stdio.h>
#include <sys/sysinfo.h>
#include <unistd.h>

int main(int argc, char** argv)
{
        int i, j;
        int k;
        int *a1, *a2;
        struct sysinfo sinfo;
        struct mallinfo minfo;

        /* set up malloc behaviour, never do memory mapping */
        mallopt(M_MMAP_MAX, 0);

        for (i = 0; i < 100; i++) {
                sysinfo(&sinfo);
                minfo = mallinfo();
                printf("a: %ld free ram, mmap: %d, arena: %d\n",
                        sinfo.freeram, minfo.hblkhd, minfo.arena);

                /* getting memory chunks */
```

```
                j = sizeof(int)*1024*4096;
                a1 = (int *) malloc(j);
                a2 = (int *) malloc(j);

                printf("base a1: %p, base a2: %p\n", a1, a2);

                sysinfo(&sinfo);
                minfo = mallinfo();
                printf("a: %ld free ram, mmap: %d, arena: %d\n", sinfo.freeram,
                        minfo.hblkhd, minfo.arena);

                /* write something */
                j /= sizeof(int);
                for (k = 0; k < j; k++) {
                        a1[k] = k;
                        a2[k] = k;
                }

                /* verify memory content */
                for (k = 0; k < j; k++) {
                        if (a1[k] != k) printf("error in 1 at %p\n", &a1[k]);
                        if (a2[k] != k) printf("error in 2 at %p\n", &a2[k]);
                }

                /* release memory */
                free(a1);
                free(a2);
        }
        return 0;
}
```

The memmmap.c listing:

```
#include <malloc.h>
#include <stdio.h>
#include <sys/sysinfo.h>
#include <unistd.h>

int main(int argc, char** argv)
{
        int i, j;
        int k;
        int *a1, *a2;
        struct sysinfo sinfo;
        struct mallinfo minfo;

        /* set up malloc behaviour, do always memory mapping */
        mallopt(M_MMAP_THRESHOLD, 0);

        for (i = 0; i < 100; i++) {
                sysinfo(&sinfo);
                minfo = mallinfo();
                printf("a: %ld free ram, mmap: %d, arena: %d\n",
                        sinfo.freeram, minfo.hblkhd, minfo.arena);

                /* getting memory chunks */
                j = sizeof(int)*1024*4096;
                a1 = (int *) malloc(j);
                a2 = (int *) malloc(j);

                printf("base a1: %p, base a2: %p\n", a1, a2);

                sysinfo(&sinfo);
                minfo = mallinfo();
                printf("a: %ld free ram, mmap: %d, arena: %d\n",
```

```
                        sinfo.freeram, minfo.hblkhd, minfo.arena);

                /* write something */
                j /= sizeof(int);
                for (k = 0; k < j; k++) {
                        a1[k] = k;
                        a2[k] = k;
                }

                /* verify memory content */
                for (k = 0; k < j; k++) {
                        if (a1[k] != k) printf("error in 1 at %p\n", &a1[k]);
                        if (a2[k] != k) printf("error in 2 at %p\n", &a2[k]);
                }

                /* release memory */
                free(a1);
                free(a2);
        }
        return 0;
}
```

The fb.c listing:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <linux/fb.h>
#include <unistd.h>
#include <sys/mman.h>
#include <sys/ioctl.h>

int main(int argc, char **argv)
{
        int fd = open("/dev/fb0", O_RDWR);
        struct fb_var_screeninfo screeninfo;
        ioctl(fd, FBIOGET_VSCREENINFO,

        if (screeninfo.bits_per_pixel == 8) {
                int width = screeninfo.xres;
                int height = screeninfo.yres;

                unsigned char *data = (unsigned char *) mmap(0, width*height,
                PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0);

                for (int row = 0; row < height; row++) {
                        for (int col = 0; col < width; col++) {
                                data[col + row * width] = 0x01;
                        }
                }
                munmap(data, width*height);
        }
        return 0;
}
```

6.8 Kernel Parameter Passing Fails

Passing parameters to the current kernel does not work, because of the implementation of linux−2.4.5−rmk4−np1/arch/arm/mach−sa1100/weararm.c and the wrong initialized boot parameter block.

7 Hardware

7.1 Tested Devices

7.1.1 Network Cards

7.1.1.1 D−Link DFE−670TXD (PCMCIA Ethernet Card)

Works without problems. Within the IFW building simply plug in the card, no further setup is required. On other locations you may change the network setup. For PCMCIA ethernet cards the configuration is stored in the file /etc/pcmcia/network.opts.

7.1.1.2 Cisco Aironet 350 (Wireless LAN)

Works without problems. Within the IFW building plug in the card and authenticate via valid.ethz.ch.

```
# ssh −l username valid.ethz.ch
```

On other locations you may change the network setup or the wireless settings. The configuration is stored in /etc/pcmcia/network.opts, /etc/pcmcia/wireless.opts and /etc/pcmcia/wlan.opts.

7.1.2 Mice

7.1.2.1 Dell PS/2 Mouse with Wheel

Works within X, Konqueror/Embedded and Opie at startup without problems. You need to connect the mouse before system startup, hotplugging does not work. Do not forget to configure the applications to use this mouse. With gpm or if you restart one of the previous mentioned applications the following error appears in the kernel log:

```
IRQ LOCK: IRQ68 is locking the system, disabled
sa1111_mask_lowirq(irq=0x44)
```

This message is being repeated till you press a key on the keyboard for some time. After that the mouse may work or not. You can try it again till it works. The cause is probably a bug in the kernel.

7.1.2.2 Logitech Mouseman Optical Dual Sensor USB

Works without problems, hotplugging works. Do not forget to configure the applications to use this mouse.

7.1.3 Sound Devices

7.1.3.1 Griffin iMic

Works without problems. Do not forget to configure the applications to use this device.

7.1.3.2 Telex USB Audio Device

Does not work! If you try to play or record sooner or later the system will became unstable. This is a known bug in kernel 2.4.5 and has something to do with the ohci driver (look at http://www.qbik.ch/usb/devices/showdev.php?id=854).

7.1.4 Serial Ports

7.1.4.1 WearARM Onboard Serial Ports

Two of the serial ports (ttySA0 and ttySA2) are working without problems. The ttySA1 Port caused strange behaviour on the attached test device (a sensor) and a "kernel Oops". It seemed to be an electrical problem, so no further tests has been done.

7.1.5 Keyboards

7.1.5.1 PS/2 Keyboard

Works without problems. Hotplugging does not work, so it needs to be connected before system startup.

7.1.5.2 USB Keyboard

Works without problems. Hotplugging works. No further setup is required, even if more than one keyboard is connected.

7.1.5.3 Twiddler

Works without problems. Hotplugging does not work, so it needs to be connected before system startup. The first two keystrokes get lost.

7.1.6 USB

7.1.6.1 USB Hub

Works without problems. No setup required.