
Debian on Handheld Computers (The ‘‘Pocket Workstation’’ project)

Klaus Weidner - kw@w-m-p.com

Abstract

This talk describes the results of a porting effort that makes a full Debian Gnu/Linux environment available for the Sharp Zaurus 5x00, C700 and HP/Compaq iPAQ handhelds.

The new generation of handheld computers is sufficiently powerful to run a full operating system, offering many more features than just PDA applications. In contrast to other projects that aim to build an environment suitable for untrained end users, the ‘‘Pocket Workstation’’ project is intended for users comfortable with Linux who want an environment on their handheld that is as close as reasonably possible to the desktop environment they are used to.

The talk describes the current project status, describes technical challenges and solutions involved in running Linux on a handheld, briefly compares the different approaches taken by other distributions, and invites discussion on future directions and suitability for various real-life scenarios.

More information about the status of the project is available on the web site:

```
http://www.w-m-p.com/pocketworkstation/
```

Introduction

Warning and Disclaimer: This paper occasionally deals with highly inflammatory subjects, including the words ‘‘KDE’’ and ‘‘Gnome’’ appearing in the same sentence. Since it would have tripled the length to include all the ‘‘on the other hand’’ explanations, I’ve instead chosen to present everything from a very subjective point of view. For every choice described, there are plenty of alternatives, and just because I considered a particular component to be unsuitable for my purpose does not mean that it’s bad. Feel free to choose whatever works for you.

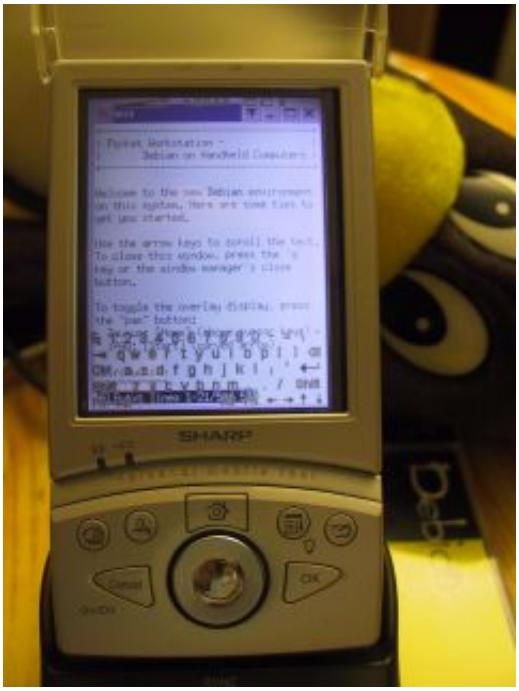
Why??

Having owned a number of handheld devices, starting with the MsDOS-based HP 200lx, I had learned to appreciate having an operating system on a handheld that resembles a desktop computer. You’ll instantly have a much larger selection of powerful software you can run on the handheld, and are not limited to programs that were specifically designed for it.

Of course, an application designed for a desktop computer will not necessarily be very pleasant to use on a handheld, but it’s still nice to have the choice to at least try it.

Why Debian? It’s the Linux distribution I’m most familiar with, and has the advantage of a huge archive of prepackaged software available for the ARM architecture.

Is it something for you? If you want a hassle-free electronic organizer, you won’t be happy with it. If you enjoy experimenting with strange and unusual applications on the handheld just because you can, you can have a lot of fun with it.



Limitations of the platform

Porting a full Debian distribution to a handheld was somewhat challenging for a number of reasons. Even though current handhelds are very powerful machines, they've been designed for tasks much different than a desktop computer, and those design decisions must often be worked around.

Screen resolution and format

The current standard for handheld devices is a 240x320 color display in portrait format. This is adequate for PIM tasks, but not well suited for generic applications.

Fullscreen text-based applications are especially troublesome, since these usually expect a terminal window 80 characters wide. This is impossible in portrait mode, and even if you rotate the device to landscape mode, the resulting 4x8-pixel character cell is at the lower limit of legibility.

Keyboard missing or incomplete

The typical WinCE-style handheld has no keyboard at all, just a handful of buttons. Since very few Unix programs can be controlled completely with the mouse, a software-emulated keyboard is necessary. This should support all necessary key combinations (I've needed to reset the handheld due to being unable to interrupt a *ping* with `Ctrl-C`), be available whenever needed, and not interfere with the applications themselves.

Disk space

While this was the toughest limit on first-generation Linux-capable handhelds that were limited to 16MB onboard flash ROM, it is no longer a major issue on current systems. Most handhelds nowadays support SD/MMC storage with available capacities up to 512 MB. Large storage cards are however still expensive, and it would be nice to be able to run a system off a 128MB card.

For this project, I'm depending on having at least a 128 MB storage card available for the installation. You can theoretically get it installed in 64 MB, but won't be able to do anything useful with it.

Main memory and CPU speed

Most handhelds have at least 32 MB RAM and a 200 MHz CPU, which is sufficient for many programs. Newer devices have 64 MB and a 400 MHz CPU, which is the equivalent of a fairly nice desktop computer not all that many years ago. A problem is however that swap space is usually not available (swapping to a flash device is so slow that it's almost useless), so you'll need to be careful not to exceed the available RAM, or risk facing the wrath of the OOM killer.

Note that handhelds generally lack a floating-point processor, so that calculations which take a fraction of a second on a desktop CPU will be agonizingly slow on the handheld. This is not an issue for most applications since they use integer operations exclusively, but may strike in unexpected places (the ''links -g'' browser needs one minute to launch due to initializing gamma-conversion tables using floating-point math).

Expansion slots and networking

Nowadays, many handhelds manage to fit dedicated networking hardware and/or expansion slots into the small form factors. If you want to get on the Internet, you have the following options:

Wireless LAN (802.11b WiFi)
: Some models have WLAN support built in, most others can use expansion cards for the CompactFlash slot. Linux support is generally good, but be careful because larger cards may physically block the headphone plug, stylus or other accessories.

Cell-phone modems via serial cable, infrared or Bluetooth
: Internet access over a cell phone is rather slow and can be horrendously expensive, but it's still the most dependable way to get connectivity while you're on the road. If you don't have a flat surface to work on, keeping an infrared connection lined up while working is a balancing act, so go with Bluetooth or a good old-fashioned cable instead. Many cell phones can be controlled via AT commands over IRComm or equivalent and are fully supported by Linux, but beware of proprietary interfaces.

If your handheld's expansion options are limited, you'll need to swap cards frequently and may be prevented completely from some tasks - how do you download a large file if the WLAN card and Microdrive are competing for the single CF slot?

The dual expansion slots offered by the Sharp Zaurus are ideal - use a SD/MMC card for storage, and keep the CF slot free for networking or other expansions. If that's not enough yet, you also have the serial/USB port, infrared and bidirectional audio available at the same time.

Bootstrapping Debian

Once you are able to use the Debian package management system, life is easy, and programs are just an *apt-get* away. The trick is how to get that far...

Note that I'm assuming that you already have a minimal Linux distribution of some kind running on your handheld. If you're currently running WinCE or some other system, this method is not usable.

The Debian project has developed the *debootstrap* tool to download and configure a minimal Debian system from a different host. I didn't use this tool for several reasons - primarily because I did not know about *debootstrap* at the time I worked on my port, but also because it turned out that a different approach offered a bit more flexibility.

What I ended up with was the *chroot-debian* package - a shell script that prepares a Debian directory suitable for use as a *chroot*. It can be run on the Linux-based handheld, or alternatively on a desktop system. (It doesn't need to be run on a Debian system, even non-Linux Unix systems would probably work as long as they offer the few needed tools such as *wget*).

The result of the bootstrap process is a single directory containing a minimal Debian system. The *chroot* system call locks a process inside this directory, making it appear to be the root (/) directory of a complete filesystem. Inside the *chroot* environment, you can run programs, install additional software, and generally work almost as if you were on a natively-installed system.

There are a couple of important points to watch out for though - some resources are still shared between the native and the chroot environment (including network configuration and ports, kernel modules, filesystems and other kernel services). This can be an advantage, since the chrooted system does not need any hardware-dependent configuration or drivers, but can also be annoying if the service you want is not easily available.

Choosing the applications

It was in some cases surprisingly difficult to find software suitable for use on the handheld. While the Debian archive offers a lot of choices, many programs were ill-suited for the handheld.

Applications that are high in the food chain require a large amount of infrastructure to be installable. This is most noticeable for those applications based on KDE or Gnome:

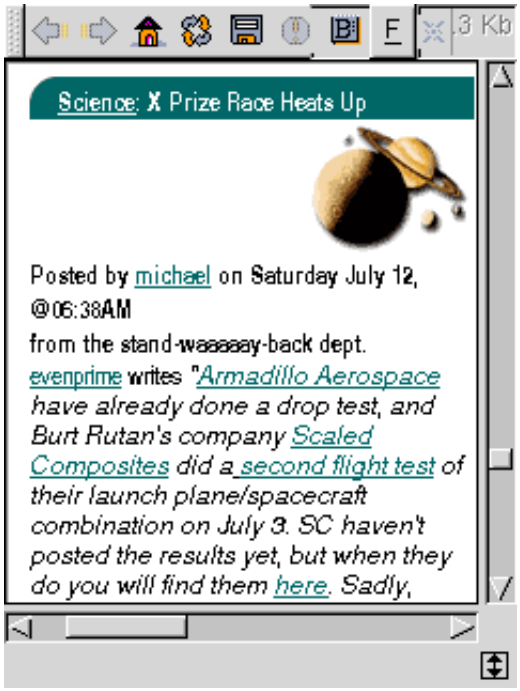
```
# apt-get install galeon
Reading Package Lists... Done
Building Dependency Tree... Done
The following NEW packages will be installed:
  docbook-xml esound-common galeon galeon-common gconf gdk-imlib1 gnome-bin
  gnome-libs-data gnome-mime-data libart2 libaudiofile0 libcdparanoia0
  libesd0 libgconf11 libglade-gnome0 libglade0 libgnome-vfs-common
  libgnome-vfs0 libgnome32 libgnomesupport0 libgnomeui32 libgnorba27
  libgnorbagtk0 libnspr4 liboaf0 liborbit0 libscrollkeeper0 libxml1 libxml2
  libxslt1 mozilla-browser oaf scrollkeeper sgml-base sgml-data
0 packages upgraded, 35 newly installed, 0 to remove and 163  not upgraded.
Need to get 17.1MB of archives. After unpacking 51.3MB will be used.
```

```
# apt-get install konqueror
Reading Package Lists... Done
Building Dependency Tree... Done
The following NEW packages will be installed:
  gcc-3.3-base kdebase-libs kdelibs3 kdelibs3-bin konqueror lesstif1
  libdb4.1 libfam0c102 libkonq3 liblcms libmng1 libpcre3 libqt2 libssl0.9.7
  libstdc++5 libxml2 libxslt1 python python2.2
0 packages upgraded, 19 newly installed, 0 to remove and 163  not upgraded.
Need to get 17.6MB of archives. After unpacking 56.7MB will be used.
```

This state of affairs was one reason that projects such as Konqueror Embedded were started to shrink down these applications, by removing features considered to be unnecessary and streamlining the dependencies. Following this to a logical conclusion resulted in the Qtopia (QT/Embedded) and GPE (GNU Palmtop Environment).

An alternative is to choose separate standalone applications, which in many cases still offer a well-rounded feature set and greatly reduced dependencies.

The *links -g* graphical browser.



dillo's line wrap method makes it useful even on a tiny handheld screen.

links -g

This little-known graphical web browser offers a lot of features in a single 3MB executable, including built-in anti-aliased scalable fonts, image resizing, SSL encryption, a subset of JavaScript adequate for most popular web pages, frames, tables, and good bookmark management. It's also very useful in text mode.

dillo

A tiny browser that is unfortunately still missing key features (SSL and data download), but is the only one that is useful for one very fundamental purpose - actually **reading** the text on web pages. All other browsers have the extremely irritating habit of formatting text paragraphs with lines longer than the screen width, requiring horizontal scrolling for each line of text. Dillo has an

option to limit the text line length to the display width, so you'll only need to scroll vertically to read a text column. The user interface can be turned off for full-screen use.

Design decisions

If the requirements of the application and the features offered by your platform don't match, you have two choices:

The *Procrustes* method
: Modify the application until it fits the features offered by the platform. Chop off all the bits that don't fit. The disadvantage is that this is very painful to the user and also to the developer of the program. If the resource limitations are no longer relevant due to technological progress, the work is obsolete.

The *Matrix* method
: Create a virtual platform that simulates the environment expected by the application, and tools so that the user can interact with the virtual environment. Also painful for the user, but they may stop noticing it after a while, and it keeps the developers happy.

Qtopia, GPE and TinyX are using the *Procrustes* method. Applications are heavily modified and occasionally re-created to fit the platform. This includes a stripped-down software manager (ipkg), incompatible GUIs that aren't capable of running unmodified Unix software, and the risk of being obsolete when the next generation of handhelds becomes popular.

For the Pocket Workstation project, I'm using the *Matrix* method. Applications can use a virtual screen larger than the physical one, and the user can interact using a virtual keyboard and virtual three-button mouse. This at least makes most software at least theoretically usable, even though it may be unsuitable in practice. But at least you can try it out with a minimum of effort.

The virtual environment

There is already an existing framework for applications that need to deal with virtual displays, keyboards and mice - the Virtual Network Computing (VNC) environment. Instead of drawing directly on a screen, a virtual framebuffer is created in a block of memory, and the applications change the data there as if they were modifying a physical framebuffer. A VNC client can connect to this server, and will receive screen updates in the form of rectangular blocks of pixels that were modified by the applications. The client can send mouse and keyboard events that will be forwarded to the applications running on the virtual screen.

This approach has several advantages:

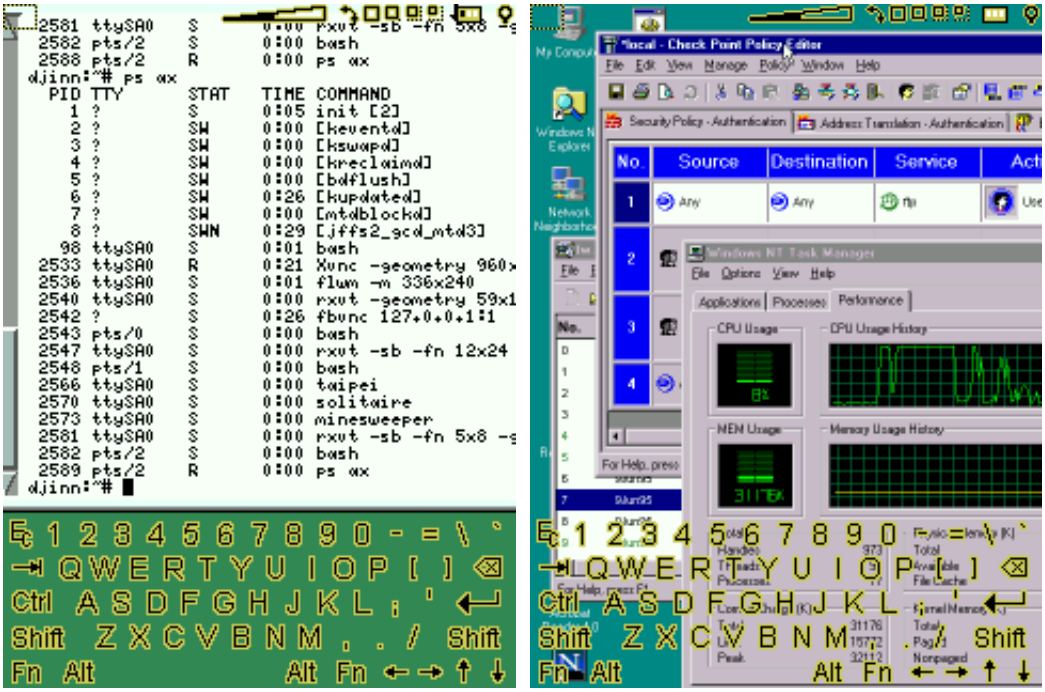
- All hardware dependencies are concentrated in the VNC client. The client program is extremely simple, and can easily be adapted to work with a new type of hardware.

- The hard work is done by the server, which is a portable C program with very few operating system dependencies, and no hardware dependencies.

- The VNC protocol is based on very fundamental basics (blocks of pixels, mouse and keyboard events), and can easily be adapted to work with nearly any type of GUI. VNC servers exist for X11, MS Windows, and other GUIs.

To avoid reinventing the wheel, I decided to use a combination of a VNC X11 server together with a VNC client doing framebuffer-based graphics. I was initially doubtful about the performance, since it is a rather indirect approach, but this turned out not to be an issue except for games and movies.

The VNC client *fbvnc* includes a builtin virtual keyboard and 3-button mouse emulation, along with on-the-fly anti-aliased scaling and panning to handle virtual displays larger than the physical one.



The *fbvnc* client.

Virtual Opie

One annoying problem turned out to be that Qtopia/Opie did not coexist well with the virtual display. It is impossible to launch fbvnc while Qtopia is running (due to a bug in QT/Embedded's virtual terminal handling), so you'd need to shut down Qtopia to switch to the X11 environment.

You could keep the X11 environment running in the background while using Qtopia and access it using the *keypebble* VNC client from Qtopia, but that did not work as well as *fbvnc* and also needed too much memory to be useful on a 32MB system.

I had just discovered recently that QT/Embedded supports a VNC server mode directly as a graphics back end. This way, you can launch individual applications from within the Debian environment, and communicate with them using a separate VNC session via *fbvnc*. Launching just the application you need needs far less resources than running the full environment. Unfortunately the default configuration for QT/Embedded does not enable this mode, so you'll need to recompile the library to make it available.

Virtual Paper

Since the *fbvnc* client already supports scaling and panning of images, it was very easy to extend it for uses other than virtual desktops. By reading bitmap images from a FIFO, PDF/PostScript and image viewers viewers could be implemented as small shell scripts, based on GhostScript and the netpbm library.

The next-generation handhelds

A worthy successor to the HP 200lx has finally arrived - the Sharp SL-C7x0 series of handhelds offers a 640x480 landscape display, a full keyboard, and 64 MB RAM in the newer models. It's even shipped with a Linux system on it. Officially it's only sold in Japan, but several online stores can import it for you.

My new C750 has convinced me that the *Matrix* method was the right approach - I was able to continue using the same software as on my old SL-5000D, just much more comfortably.

Where to go from here?

PIM applications are needed

While the Debian handheld works well for many advanced tasks, it's not well suited as a personal information management (PIM) tool.

The Qtopia applications are currently the most advanced PIM apps developed specifically for a handheld, but unfortunately they are not easy to get working in an X11 environment.

The GPE applications are IMHO currently still in a proof-of-concept stage and not suitable for day-to-day use.

Currently I'm using Jpilot, which was originally designed to be a X11 replacement for the Palm desktop. It's usable on the C7x0 with their high-resolution display, but it lacks features such as IR beaming of information. It can sync with a Palm handheld, but unfortunately not with a desktop Linux machine.

I haven't found a desktop PIM suite that's a good fit for the handheld device - Evolution and the KDE tools are rather big, and would likely be too slow, especially on startup.

Performance improvements

Running applications through the VNC server and client is inefficient for one primary reason - all the modified pixels are stuffed through a TCP socket, which results in a huge amount of needlessly copied data.

This is not an important issue for most applications, but for games, movie players and other programs that need full-screen redraws several times per second, it's not fast enough.

The VNC protocol could easily be extended to communicate via a shared memory region when the client and server are on the same machine. The TCP-based protocol is used in the same way as now, but instead of using one of the existing pixel encodings, the pixels are read directly from the shared memory virtual framebuffer. This will permit much faster updates, and also save memory since only one copy of the virtual framebuffer is kept in RAM. The resulting performance would likely be nearly equivalent to direct framebuffer access or native X11.

Appendix

Compiling QT/Embedded and Opie

QT/Embedded and Opie are usually cross-compiled for the ARM handhelds, but the Debian environment also supports native compilation. Note that this will be rather slow and require over 200 MB of disk space (which can be on a network drive) in addition to the compilation environment.

The most important option is to enable support for the VNC server graphics backend, but you can of course also do other customizations at this point.

In these instructions, I'm differentiating between the target machine (the handheld you'll be running the programs on) and the build host (which in my case was an iPAQ handheld running from an NFS-mounted chroot Debian installation).

First of all, you'll need a couple of libraries on both the target and build machine:

```
apt-get install libstdc++5 libgsmme1 libbluetooth1 libopenobex1
```

On the build machine, you'll need a C++ development environment and some additional development libraries:

```
apt-get install g++-3.2
apt-get install libgsmme-dev libbluetooth1-dev libopenobex-dev
```

Download the current Opie source code from CVS as described on the web page:

```
http://opie.handhelds.org/wiki/index.php?SourceCode
```

You'll also need the qt-embedded distribution:
ftp://ftp.trolltech.com/qtopia/source/qt-embedded-2.3.5.tar.gz

In addition, you'll need the tool *uic*, the easiest way to get it is:

```
apt-get install libqt-dev
```

Before building qt, you should fix an alignment bug in the VNC driver. It'll work without the fix, but slower due to the misaligned memory access.

```
--- ../opie-cvs/qt-2.3.5/src/kernel/qgfxvnc_qws.cpp     Tue Apr  1 19:26:52 2003
+++ qt-2.3.5/src/kernel/qgfxvnc_qws.cpp Sun Jul 13 16:54:00 2003
@@ -70,8 +70,8 @@
```

```
 struct QVNCHeader
 {
-    bool dirty;
     uchar map[MAP_HEIGHT][MAP_WIDTH];
+    bool dirty;
 };

 class QVNCScreen : public VNCSCREEN_BASE {
@@ -1136,8 +1136,9 @@
     hdr = (QVNCHeader *) shmrgn;

     if ( virtualBuffer )
-        data = shmrgn + ( sizeof(QVNCHeader) + 7 );
-       return TRUE;
+        data = shmrgn + ( ( sizeof(QVNCHeader) + 7 ) & ~7 );
+
+    return TRUE;
 }

 void QVNCScreen::disconnect()
```

You'll also need some kernel header files for building hardware-dependent modules, in my case the following:

```
arm/sharp_apm.h
arm/sharp_char.h
```

Follow the Qt build preparation instructions for Opie at http://opie.handhelds.org/wiki/index.php/BuildingQtForOpie - but use a modified *configure* command:

```
./configure -qconfig qpe -depths 4,16,24,32 -system-jpeg -system-libpng -system-zlib -no-xft -qvfb -vnc
```

I've had a bit of trouble due to it calling the compiler with the wrong name, and *gcc* instead of *g++* - you can fix this in the Makefile, or alternatively use the following workaround:

```
ln -s /usr/bin/g++ $QTDIR/bin/arm-linux-g++
ln -s /usr/bin/g++ $QTDIR/bin/arm-linux-gcc
ln -s /usr/bin/g++ $QTDIR/bin/gcc
PATH=$QTDIR/bin:$PATH
```

After that you can start the compilation with *make*, which will take a couple of hours on the handheld.

Building Opie is then straightforward as described in the instructions mentioned above.

The Opie binaries will use direct framebuffer graphics by default, use the following environment variable settings to switch them to VNC mode:

```
QTDIR=/mnt/zdeb/src/src/opie-arm/qt-2.3.5
OPIEDIR=/mnt/zdeb/src/src/opie-arm/opie

LD_LIBRARY_PATH=$QTDIR/lib:$OPIEDIR/lib
PATH=$PATH:$OPIEDIR/bin
QWS_DISPLAY=VNC:2

export QTDIR OPIEDIR LD_LIBRARY_PATH PATH QWS_DISPLAY
```

Then you can launch individual Opie programs (you must add the `-qws` flag to the command line), and connect your VNC client to 127.0.0.1:2 to interact with them.