ACE UNIX for Embedded Systems

Hans Schipper
ACE Consulting bv
<hans@ace.nl>

1. Introduction

Since its early days, ACE has been involved in the development and the construction of UNIX systems.
The development of the ACE UNIX implementation started with a real-time network implementation for UNIX V6 (1978), followed by numerous 'regular' UNIX workstation implementations in the mid 80's (mainly for Motorola type processors) and a large heterogeneous multi-processor implementation (1990). Throughout these years, portability, scalability and processor (and hardware) independence have always been the decisive factors in the development direction of the source code of the system.

In 1998 ACE was asked to implement a kernel with a standardised (UNIX/POSIX) application interface for a mass-scale embedded system based on a 8051-processor. The main driving force behind this "move in architecture" was the short time-to-market, which is quite common for these types of products. The (standardised!) UNIX interfaces allow a highly paralleled development planning with independent sub-projects for the application, the system software and the hardware development. The interface standards between these sub-projects were readily available in the many books about UNIX-system architecture.
Emulation of the final system is also easily possible on any existing UNIX implementation, only some provision for the (of course quite special) IO devices is necessary.

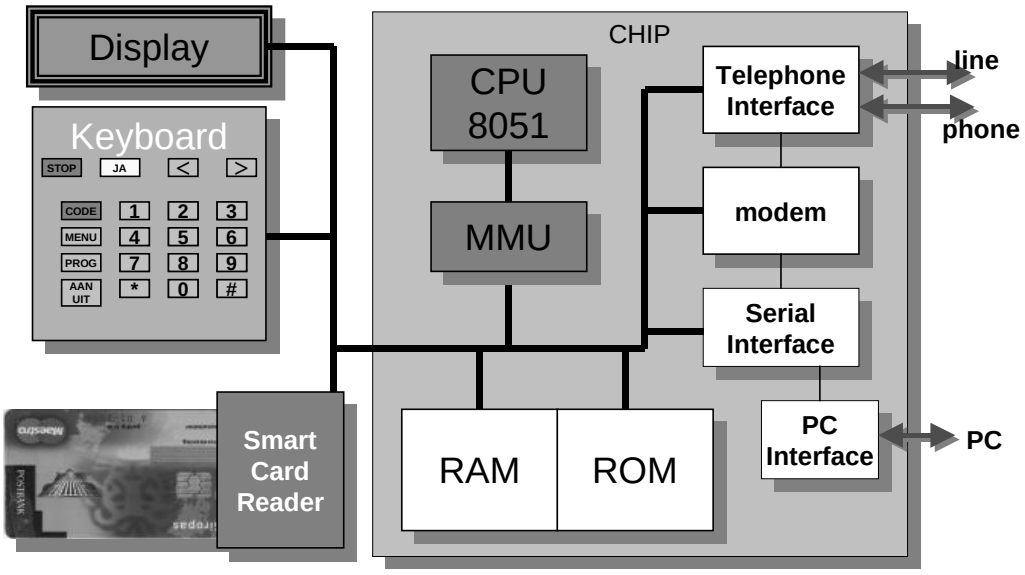
It came as no surprise that the ACE UNIX system appeared to be flexible enough to be downscaled to such a small embedded system with a very particular hardware interface and memory configuration.

In this overview of the architecture of the ACE UNIX product in an embedded environment, first the functional requirements of the embedded system are discussed, together with the way in which the ACE UNIX implementation matched those requirements. This section also discusses a number of special components of the system (in-core file system, the ability to download not only tasks but also libraries and kernel routines). Finally, the development environment, a crucial part of a high quality embedded systems offering, is described.

2. System Architecture

The embedded UNIX system forms the core of a hand-held pay-terminal capable of performing secure financial smart-card transactions, using a telephone-line or a PC-coupling. The terminal has a simple (mobile phone-like) keyboard, a 2x20 character based display with icons, a chipcard interface, an elaborated telephone line connection with modem and finally a connection to the parallel port of a normal PC.

The on-board memory consists of battery backed-up RAM and (EEP)ROM, in total about 148Kb large. In order to be able to handle the larger than usual memory space (the 8051-processor has only a 64Kb addressing capability) a simple memory management unit was added.
Next to the financial transactions, also a number of telephone related functions (number recognition of incoming calls (CLI), a short dial function, storage of telephone numbers on the chipcard, etc.) are supported.

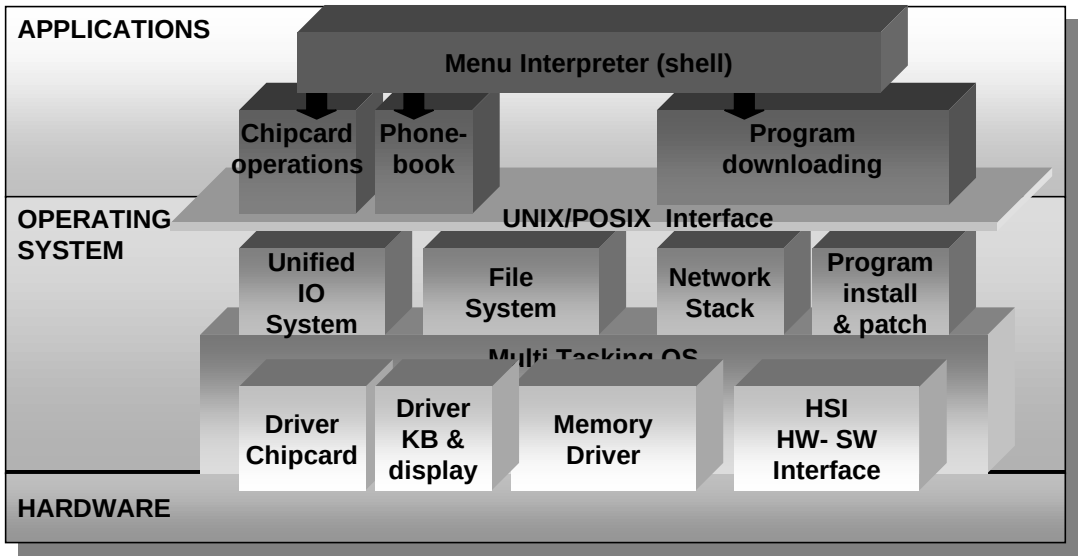


All user functions are to be implemented as separate programs (UNIX style executables) under the control of a simple menu application (functionally equivalent to the UNIX-shell). For future in-field upgrade and/or extension of the functionality it is possible to dynamically download (via the telephone line or the PC interface) new applications into the system. As new applications may need additional system functions or library routines, even parts of the in-core (shared) libraries and most parts of the kernel can be downloaded. This ability also allows for "field upgrading" of most of the ROM-based software in the system, without compromizing the quite strict security requirements for the system.

3. The Embedded ACE UNIX Implementation

Like in any normal UNIX system, the ACE embedded UNIX implementation has the notion of a root hierarchical file system that contains all executables, libraries, device nodes, persistent configuration information and applications data, nicely structured in separate directories.

The ACE UNIX implementation is especially suited for embedded systems because of its high modularity: when a library function or a kernel API is not needed by the application, it is not included in the system. This approach dramatically reduces the 'footprint' (i.e. the size) of the kernel and support environment. However, if (even after the system has become operational in the field) a new application is to be downloaded, and when that new application requires kernel- or library functionality that is not yet included in the base system, the necessary routines can be downloaded and installed into the system and made available to the applications.
A package concept is used to structure the download process. A package can contain all above mentioned parts to upgrade the system to the required level. The device itself keeps track of its state and "asks" the download host for the required packages.



Through the same mechanism also bug fixing (replacement of existing modules) can be done, even if these routines are originally based in the (mask) ROM. The restrictions for patching are mainly a consequence of security issues, and are implemented to prevent the deterioration of the security level of the device. For that reason the packages are heavily checksummed and cryptographically signed by the download host system and thoroughly checked before any installation will occur.

During the so-called factory-reset (the first time booting of the device), the RAM based memory pool is initialized and the in-core file system is recreated, using the file system information (prepared by the build tool) from the ROM memory. Note that there is no need to copy the actual data of the files from ROM to RAM: the file system implementation can use ROM and RAM addresses in a transparent way, even showing the modes of the files as "read-only" in case of ROM.

When a (data or executable) file is to be downloaded, the download application informs the host system about the current level of its software (version number). The appropriate package(s), containing one or more executable- and/or library files comprising a complete update of the system, is (are) then downloaded as a regular file and stored in the **/tmp** directory. Once the packages are completely received, the checksum and the security key are checked, the packages are unpacked and the components installed in their appropriate directory in the file system.

Contrary to 'regular' UNIX systems, the ACE Embedded UNIX system does not copy executable files into RAM upon execution; in stead, during the installation phase (either statically by the build tool, or dynamically by the download application) executable code is loaded at a suitable memory address, and then logically relocated to contain the correct references for functions and data addresses.

The IO system is UNIX-like constructed, the drivers implement the normal standardised system call interfaces, like *open*(2), *close*(2), *read*(2), *write*(2) and *ioctl*(2). The interrupt system is highly optimised to allow for the relatively high interrupt rates and required short latency of the smartcard and serial interfaces. Interrupts are actually "decoupled", which implies that the code that runs with the interrupt system disabled, is as short as possible (only dealing with the hardware handshake) and as soon as possible a low level task is activated to handle the rest of the IO transfer and further handling. The kernel uses the same low level tasks also for its own administrative processes like scheduling, watchdog, idle etc.

4. The Development Environment

One of the major reasons to select a regular OS with a standardized (POSIX compliant) application interface for the embedded system is the fact that this approach allows for application development on a normal UNIX workstation with all the standard debug facilities. And (almost as usual) since the applications have to be developed well before the actual embedded platform is available, all development of the applications is done using this standard UNIX workstation.

In the application development area, strictly separated from the normal program development area of the host, a special cross-platform C libraries are installed that contain the additional (IO-)functionality and additional checking on functionality that is not included in the actual system; in this way erroneous calls are easily discovered. Special attention is given to the fact that the target system is a 16 bit C language implementation, which will imply for instance different round-off behavior as compared to most current UNIX workstation which are commonly 32 bit implementations.

For all the IO-devices of the embedded device an emulation version is available. For instance the keyboard and display are emulated through a mock-up by a graphical interface on X-windows and Tcl/TK based routines; the smartcard is accessed through a serial port of the workstation and a library routine written in C. These emulations are accessed using exactly the same interface semantics (with calls like *open*, *close*, *read*, *write* and *ioctl*) as implemented for the actual devices in the embedded system.

For the debugging of the final system a 8051 simulation is available to debug the low-level code even before the actual hardware becomes available. This simulator includes simulations not only for the CPU but also for the memory management, IO hardware and interrupt system. The hardware-generated interrupts are under full control of the user of the simulator or can be automatically timed.
Only during the (short) initial integration testing of the first hardware and software, a logic state analyzer was used.

Next to the ACE Software Control system CADESE and a C compiler for the targetsystem, the development environment also contains an elaborate build tool. This build tool first creates a kernel/library "virtual" image of the system, containing only those functions that are actually needed by the system (including all the special runtime support functions expected to be present for the code generated by C compiler). Then, using the (relative) addresses and sizes from the first step, the applications are all linked together, taking into account the addressing restrictions imposed by the hardware capabilities. Finally the images are packed together into one final ROM burn load-image with a checksum and an encrypted key.
The build-tool can also prepare downloadable package files containing a set of executable and/or library files that can be used to upgrade an already installed system in the field.

5. Conclusions

- The UNIX/POSIX interface is very well suited as operating system interface in embedded systems.

- The memory requirements do not have to compromise the quality of the architecture of a system.

- It is possible to design a field upgradeable system and development environment, without putting the entire burden on the head of the application programmer.

- Well-engineered modularity is the bottom line.