UNIVERSITY OF ARIZONA
STUDENT SATELLITE PROJECT

**Technical Note**

Doc. no.: DCH-009

Revision no.: 1.0

Status: Draft

---

**Subject:** DCH Software Overview **Date:** 6/29/98

**Written by:** Mickey Britt

**Reviewed by:**

---

**Revision history:**

Revision 1.0: Initial Draft

---

# 1. Document Overview

The DCH System Kernel is a small software unit responsible for system chores at boot-up time, and the lowest level of run-time task maintenance and support. This document does not detail any hardware systems. However, the kernel interacts directly with hardware and manages higher-level hardware requests; thus, hardware considerations and design bear directly upon the kernel. This document describes the specific duties of the DCH kernel, and its interactions with hardware as well other software modules. The kernel is only one part of the DCH software as a whole.

# 2. Requirements

The kernel is the innermost foundation for every other Data & Command Handling software module. Its duty is to boot those modules into memory at startup, maintain a stable running environment, and serve as an interface for hardware systems. Other programmers should be able to code DCH algorithms, unconcerned by the technicalities of the underlying hardware. The microprocessor itself requires the maintenance of various system data tables in a way that must be handled with assembly language code.

Generally then, the kernel programmer's job is to write everything that cannot be written in C, and then link this code to C modules, forming a complete boot-loadable system.

Specific kernel duties:

— Boot-up chores and board initialization
— Software environment initialization
— Low-level runtime maintenance & hardware interface

— Handling of critical software faults

# 3. Descriptions/Designs/Discussion

# 4. Lists

# 5. Interface Requirements and Specifications

The kernel assumes an Intel386 with a coprocessor, in a somewhat ISA compatible motherboard. ROM issues: All onboard software must be ROM-burned and the boot-up code must be vectored to by a bootstrap at FFFFFFF0h. The boot-up code must be above FFFF0000h if the bootstrap is a near jump, or somewhere within the first megabyte if the bootstrap is far. Of course, there will not be 16 gigabytes of ROM or RAM, but the chip should be selected when the upper 16 address lines are high on a memory read. It is important to note that, in general, no memory above the first megabyte can be accessed unless the CPU is in protected mode (thus the infamous 1meg DOS limit). The kernel must not be required to access such memory until fully initialized (i.e., in protected mode, and stabilized).

*Protected mode:* A real-mode kernel would be far easier to write, but would lack the processor's 32-bit capabilities and build-in fault protection mechanisms, and its 1meg memory limit would have to circumvented with no small difficulty. The kernel will run in protected mode. This adds immense complexity, but the system should be more stable and have better performance.

*C code patching:* Unless a C compiler meeting certain requirements can be found, higher-level modules will be written in Borland C, compiled to assembly, and textually patched up, before a final assemble and link. The compiler would have to be: 1) ANSI compatible C, 2) 32-bit capable and not restricted to flat mode (i.e. far calls and data are possible), 3) Able to compile into legible, TASM or MASM 32-bit Intel assembly code (not gibberish "intermediate" code). I do not believe such a compiler exists. A special program must be written and maintained, to edit semi-compiled code so it will ultimately run in the satellite environment. This is just as ugly as it sounds. C coders will use special nomenclatures to make system calls. This is still in the works.

# 6. Current Status

The O/S has so far been developed on a Pentium 90mhz machine. The text patcher is underway, in Borland C++ for DOS, and partially works. The O/S can enter protected mode; paging is not really used though it is activated. It can run small Borland-compiled 32-bit C program (from DOS). This code cannot easily interact with hardware, or with the kernel for that matter, but it is a big first step. There is minimal board initialization because the program is not invoked from boot-up, though

the O/S is written to be ROM-ized. Interrupts are received and reported on-screen (and the 8529 interrupt chip is specially setup for this kernel), although only Level 0 can receive interrupts at this time (the C code runs at Level 1, which is less privileged). As an added bonus, the program returns to DOS when finished, rather than blinking out and rebooting.

The C program is the start of a simple text-windowing system, which will be helpful for debugging and testing. Pentium-specific performance stats are queried and reported, which obviously cannot be used in the final version but will also be helpful for timing analysis.

# 7. Test Plan

No specific test plan yet. Since the purpose of the kernel is only to support other modules, testing of those modules should bring to light faults in the kernel. Errors tend to cause the system to halt suddenly or become conspicuously unstable, thus making fault detection relatively simple.

# 8. Concerns and Open Issues

*Nature of hardware/motherboard:* The O/S interacts intimately with hardware and the O/S programmer must know how to program and relevant hardware unit at the assembly level. If a programmer is not alerted to a change in mother-board design, the entire satellite is likely to be rendered useless or damaged.

*Nonstandard C coding:* Other higher-level programmers must understand that they are not programming under an ordinary environment. A simple "include" statement will cause the entire package to miscompile, and the offending module may have to be rewritten. DCH programmers will be writing in a "proprietary" version of C and must remain in contact with the kernel programmers. These C coders are likely to become frustrated and feel that the demands are extraordinary, but keep in mind that a mission-critical system with no BIOS and no hard disk is DRAMATICALLY different from a home PC. Also, the program to patch up compiled C code must be maintained.

# 9. References

1. Shanley, Tom and Don Anderson. (1995) *ISA System Architecture*. Reading, Mass.: Addison-Wesley.