

# Indikernel Design Principals And Reference Platform

By Simon Jackson, BEng.

## Abstract

This paper sets out a kernel methodology which relies on the protected mode environment having message passing queues. With these queues it is possible to contact other user-mode tasks, which may perform functions which normally would be done by an OS kernel. Kernel design is also greatly eased by the modern trend for devices to use DMA and buffering, allowing polling.

## General Outline

The kernel will be implemented as a FORTH language system based on blocks with no file support, with a Java-esque application programming language. A threaded compiler would load a class file and provide bytecode conversion into execution vectors. As long as the called classes existed the Java class file would run.

The kernel is a one call thing where any pending user requirement for service is placed at the beginning of the system request queue by a TX method invocation. The operating system in its turn picks up the items in the system queue of the process which last run, and puts them in priority order. These queue items then receive method calls on there RX callback method of the queued object when the OS has finished processing the service request. This has a corresponding user space request to provide service, where the RX method is called when service needs to be provided.

The OS loader maintains a list of objects to be loaded at particular memory addresses, so allowing user mode device drivers to occupy the correct memory areas. The virtual or real architecture is a four segment design with the code segment (P), the quality segment(Q), the return stack segment(R) and the data stack segment(S). Although enforcement of these assignments is limited to the fact that P is read only, and must be the code segment. All the architectures instructions are read, then modify and then write delayed by one cycle. This allows a read before an overwrite, for a convenient swap of variables. Writing to the P segment, performs no writing for a convenient drop, but switches into supervisor mode and back again when the carry flag is set. 'Writing' to the P register for no write cycle should be done with care as supervisor mode entry could result with careless attention to the carry flag. The accumulator (A) is passed to supervisor mode.

## Primary Objects And The Naming Service

There are four primary objects at the system programming level. These are BLOK for memory storage, TASK for processing, VIEW for output and data transformation and ACTI for action input. All used classes would be derived from these base classes. The design of the system should not encourage similar concepts to have separate interfaces and APIs e.g. Icons and Fonts. Icons should be considered to be characters within a font in this case. Ask the Chinese!

The first registered service, or service 0, is the naming service. It handles name registration, and is responsible for turning names into numbers on RX or DOES>. Another service is responsible for turning numbers into names. With these primary services registered, a standard dictionary of named services becomes callable.

A more complicated namespace service deals with general text searching in unicode (16 bit) and language with translation attributes, while returning associated values. This service supports part expression matching, and comes in useful for parsing.

Named objects are movable while locked, and all pointers are refreshed. No find target!

## Collision Arbitration And Code Execution Firewall

Security is a major concern of any OS, and to this end the excessive requirements of file systems are to be abandoned. An apparent file support is within a protected application group, so that open season on any file is just not on. An application must choose to place data in an open namespace, and not be forced by design or accident.

Another common problem is with intercept vectors for application functionality. The ordering should not be on the common last come first served basis. A collision of services (intercept vectors) should be presented to an arbitration, where user intervention can decide the ordering. This ordering could even be different in differing application contexts or spaces. Each application space, containing multiple applications, is in full isolation from other application spaces except via kernel go between. Applications can be moved into more reliable spaces when they have been time proved. This goes for library updates too, and backward compatibility could be provided in an application space.

This strategy allows a code execution firewall, where services can provide an alternate service which, depending on the callers application space, provides a blank or null service request. The collision arbitrator making each application space have access to various services.

## The GUI Speed Angle

The trend for ever more complicated GUI elements which leave people with poor hardware performance out in the cold is not to be encouraged. To this end the basic toolkit uses fixed size fonts as per the 80's. Character grids start at 1\*1 and go up to a general  $2^x * 2^y$  size. This allows the basic character drawing routines, to be used for screen tile placement. All tiles being powers of two along an edge, and aligned at multiples of the character size on screen. Some people may consider this too restrictive, but do remember this is the basic GUI. The standard window is a standard grid size. All of the grid is available for application drawing, with there being no title bar. This extra information and window control is available through menus, which in turn do not appear until accessed. Grids also have a  $2^c$  color dimension and a  $2^a$  time animation dimension, with over sized grids reducing to a particular  $\frac{1}{2}$  or  $\frac{1}{4}$ , etc.. The coordinate space is 16 bit, with one nibble sizing for each dimension.

More complicated GUI services are designed in such a way as to have alternates in the basic GUI. This is eased by the application not being allowed much specific GUI object placement. For example, a proportional font is rendered by having connected grids with over and underflow shifted between them, a custom grid view if you see. This leads us on to the next topic of removing excess tart, for speed, while maintaining functionality.

## Generalized Binary And Common Understanding Dialog

Each class required for operation of an application can be scanned, and replacement classes can be arbitrated to recompile the binary. This is aided by all replacement classes, and class mapping. Each new tarty feature library must provide a class map for reduction of code into basic service calls, as well as a class map to convert basic service calls into 'tarty vision'. This forms part of the standard and is not optional. If a feature can not be converted and basic services needs an additional growth, then this must be provided as example, but it is open to further minimization.

A classic example of when 20 ways of doing things, not only causes excess consumption of resources, but also is obviously the wrong way, is the 'available – busy – away' status of many communication solutions. This should be a common OS control. The common understanding dialog is a simple dialog where drag 'n' drop of a control onto it dissects code to place the control there with a standard interface for naming the common control, and providing access to its status. A much better solution.

## Standard Advertizing Interface

The only people who do not get value for money on advertising is the customers. To this end a standard brokerage for attention is built in as a service. Any site or application requiring advertisements for operation must request a priced ad (for fee extraction), such add being displayed while application or site being used. The content provider must accept the nearest offer or close with an "I'm too precious error." A system which does not support the ad interface is non standard. All other advertising is optional.

## Virtual Machine Instruction Set

| <i>Bit7</i>          | <i>Bit6</i>    | <i>Bit5</i>         | <i>Bit4</i> | <i>Bit3</i> | <i>Bit2</i>                         | <i>Bit1</i> | <i>Bit0</i> |
|----------------------|----------------|---------------------|-------------|-------------|-------------------------------------|-------------|-------------|
| <i>ALU Operation</i> |                | <i>IN Selection</i> |             |             | <i>A OUT Selection (Next Cycle)</i> |             |             |
| 00                   | A=IN AND A     | 0 Direct            | 00 P Reg    | 0 Direct    | 00 P Reg (Special)                  |             |             |
| 01                   | A=IN AND NOT A | 1 Indirect          | 01 Q Reg    | 1 Indirect  | 01 Q Reg                            |             |             |
| 10                   | A=IN + A + C   | (++, --             | 10 R Reg    | (++, --     | 10 R Reg                            |             |             |
| 11                   | A=IN - A - C   | Auto)               | 11 S Reg    | Auto)       | 11 S Reg                            |             |             |

This instruction set is compact and, although simplistic, does contain enough functionality for Turing completeness. Conditional branching for example is done via an and of the branch offset with the -1 produced when C=1 when a number is subtracted from itself. This only works due to there being two instructions per 16 bit word, big-endian execution first. Further code compaction could be achieved by having individual bytes index a word array with two for one instructions gain.

The fetch cycle is simplified by all indirect access of registers having an automatic post-increment or pre-decrement of the indexing register, allowing indirect P fetch to the IR and byte select in the IR by clocked flip-flop. A simple busy circuit allows a single or more external cycle memory access, no cycle overhead for direct register access, and a single internal cycle for cache access. An external clock cycle multiplier makes the internal clock. All subroutine calling has to be handled by user code.

All kernel accessible special registers appear at a memory block which is located in memory. This includes a 256 words of 'microcode' for use in 2:1 instruction translation mode and an instruction mode register.

## Task Register Banks And Memory Architecture

Each of the PQRS registers is selected from 1 of 16 by one nibble in the 16 bit user or super bank control special function registers. This gives fast task switching of less than 16 tasks not including the supervisor task. This speed is taken advantage of by grouping tasks into burst processing groups, or common application spaces. Multiple application spaces have isolated security, but threads or applications within a space share common segment registers.

Although the memory and register space is 16 bit, the address space is 32 bit. Each of the registers PQRS has an upper word special register. A post-increment or pre-decrement which carries into the upper address word, causes a memory access into one of 16 (4 Register \* under or over \* user or super) 64K word segments, pointed to by special page over and page under table pointers, recalling a new value, by index, for the upper word special register of the register concerned. This allows program execution across page or segment boundaries, while providing isolation and segment relocation functionality. This mode along with a special 8 bit instruction word (LSB) mode (for ROM on boot) are controlled by bits in the instruction mode register.

This mostly 16 bit method of operation allows for a faster memory cycle externally as only the RAS has to be asserted most of the time on modern DRAM. A 16 bit refresh special register will

occasionally be asserted on the CAS. The address bits are interleaved for two stage output on the 16 bit address bus which directly interfaces with DRAM. A further eight bits in the instruction mode register select, for each of PQRS \* user or super, the 16 bit or 32 bit addressing mode. 16 bit mode does not interleave the address lines for CAS RAS memory, but only uses the lower word and R and not W data flow signals. To simplify design four chip selects are provided four up to 16 memory spaces. These are controlled by user and super chip select special registers with each of PQRS accesses using one nibble. Paging under and overflow use the chip select of the P register.

Four small on chip caches are provided, one for each of the PQRS registers. These caches are at least single associative, but to save chip area have no empty flag. To avoid cache 'errors' each cache has a two di-bits in a special cache control register (user or super). The following table shows the cache mode control.

| <i>Bit1</i>   | <i>Bit0</i> |
|---|-------------|
| 00 Cache Off (but reads and writes made to cache to fill it)          |             |
| 01 Cache On (write thru mode)   |             |
| 10 Cache On (write back mode)   |             |
| 11 Cache As Memory (the memory bus is switched off for this register) |             |

This design provides enough flexibility for all uses.

## Standard Hardware Features

Remember all that searching for a hardware driver? It would be eased by having a checksum printed on the hardware, of the first working release of a driver. This is standard behavior. It also encourages hardware suppliers to have correct drivers on release. The hardware presents a standard BLOK in the memory space, which contains the name of the class file to be loaded. This gives two ways of finding a driver for a piece of standard hardware.

All hardware must handle its own interrupts and buffer or throttle data until service is provided. The hardware implementation decides what to do on buffer under run or overflow. The OS allocates a free DMA BLOK pool, reads from filled BLOK queues and writes to various outbound BLOK queues. The processor accepts busy assert bus mastering devices. One BLOK is 256 16 bit words.

Two timers generating hardware user to super (timer) mode switching, with two special timer bank control registers and two timer external clock divider registers completes the scheduling features for hardware.