OS/Forth
OS/4th

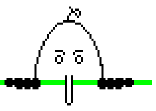
The Complete Operating System.

"*Microsoft - Breakfast of Champions*"

Presented by: Christopher Passauer

You hold the tools and future in your hands ... so ... succeed!

The tenet: **Forth when used to implement, and be, an Operating System can displace all other current offerings of (Microsoft, OS/2, UNIX based) operating systems on the PC single user desktop platform, and farther**.



Statement

This paper presents the concept, history, direction and foundation to make Forth the next PC platform operating system to come into wide scale use. It is fully realized that some may have trouble grasping such a lofty concept, but, to dispel that parallax view is the exact purpose of this writ. I will include as little connotation and equivocation as possible. Through the words presented within, the concept no longer is lofty, but instead at hand.

Preface

During the hearings held in Washington, D.C. during the first weeks of March this year (98), Bill Gates stated, as part of his position on including Explorer in all OS offerings, that "Microsoft is rewriting all its' application offerings to use an advanced HTML interface and no longer the current windows based system." This tells us simply that, and by looking at the advances in web browser/viewer technology, everything will be HTML based. Simply examine X-Windows on top of UNIX and derivative platforms. This document is in a form of HTML just as all word processor documents are and have been, remember '*^B*' and the like under WordStar. Even a text file is 'marked-up' with simple *CR/LF* and *Tab* characters, basically, format.

Forth is, by definition of its' nature in execution, the perfect string processing language creation tool yet. What is HTML but a simple grouping of text with visual display markers and graphical object pointers contained within a document. A Forth generated application could literally execute an HTML document, unlike the process used in HTML browsers and the Forth derived PostScript language.

> **Did anybody ever tell you PostScript was originally slated to be what HTML has become, a textual, graphics and format language for all output devices, that's right, for the video monitor also. At the time it was just too cumbersome to generate so much output given the severe limitations on working storage. Apple Computers having a fast graphics display ability on the newly introduced MacIntosh needed some way to get that to a print device and chose PostScript as the output language of choice. Thus in their regular open system manner provided print device vendors universal method to connect with their new graphical system.**

Point I Convey

This document will at most times be in conversational mode - until it becomes truly technical by revision, your revision. Given the short time available for this effort a dissertation format is not followed here, or great effort towards literary style, we do write one way and speak another, thus direction is towards sharing of concept and not the mechanics of writing.

Sections in this Version (that follow)

Audience

This document is written at the level of persons with an experienced software and hardware background. This effort in no way addresses the uninitiated. No spoon feeding here.

Concepts and information presented here in are from one perspective and associated resources, there can be many in both, alignment and quality of content will occur with substantiative peer and participant review cycles.

Document Direction

It is understood and directed that this, to become a working document, will become more and more technical, with that, many sections will eventually be edited out. It should be noted that the best document editing, including for this section, should be done by striking-out and not by actual deletion. This allows the history of change to be noted and roll-back to occur in need. It should also be noted that a statement of intent be included with all material changes so that no one need figure out something that should be obvious, and hidden agendas are eliminated.

The Authors (not so carefully worded) Forward

It is rare that I write a document that contains my direct and personal views on things, I normally write in a presentation format from a technical, executive or investment perspective. I do fully express my opinions and views here in, but in constructive criticism format as possible. Those of you that know me or of me will no doubt see my forward and righteous manners rear in the words trailing off before you. Yes! I unleash my heart felt passion of the subject as guide. I do not place my ego or vanity in this so that the true artistic nature of my heart and logic may speak, and I hope through revision yours also.

If you find I move across multiple subjects yet bind them together, surprise, it's the way I think. I feel now, after others have shown me, that I have three gifts: 1) I am able to see, and continue to see, the big picture even while concentrating on the smallest detail. 2) I inspire others to think differently, be challenged and then act. 3) I naturally think in a totally abstract, yet logical, manner. I do rely on my intuition.

Hey, spelling errors are inherent, expect some, however, I respect and accept all input as to format, spelling and content. Your input is not only accepted but expected, speak up please, or don't complain later. That's right get involved or Bite-It. And expect that I will not let this go the way of committees, the warm-n-fuzzys, or anything similar. Why? Because OS/Forth accommodates anything, truly, just as its' parent Forth does, as Chuck intended. So remember extensions are a fact and a design premise, not required in the heart of it.

> **Chuck took time and great pains to allow me my freedom of thought without rebuke, all the while instilling in me the beauty of minimalism, may you find that beauty in the concepts here.**

Premise for the Design

Let's face it, there are a large number of people in the world today that resent being forced to use an 85% functional solution called Windows. I admit I am one. I will say that Windows has its' place (just like UNIX, OS-9 and even MacIntosh, Sun, or Silicon Graphics does) but to the exclusion of all else - I object. I will cover many of these points in more depth later on, perhaps scattered, so bare with me.

On the television program Uncommon Knowledge, #207, created by the Stanford Hoover Institute and produced by KTEH channel 54 PBS Station San Jose, hosted by Hoover Institute Fellow Peter Robinson, points out that the Monopoly called Microsoft sells 90%+ of all operating systems world wide and 80%+ of all desktop operating systems. The cost to the manufacturer for those operating systems has risen, not decreased, from ~$4.00 per to over $50.00 per. The manufacturers do not give a component cost breakout so the increases are hidden from the consumer.

My father, Buzz, a new friend of mine, stated to me, and I had to laugh - at myself - because it was absolutely true, 'every software manufacturer out there today wants a new operating system because it helps them do nothing more than generate more revenue - it's another opportunity to them.' This points out that most software vendors will create versions of existing software applications for any new Operating System that catches on. I ask you - would they convert software over if was really easy to do so?

Software vendors can not thrive unless you continue to purchase something from them. If they create the perfect program they would go out of business after the initial sales were over, thus their continued program of bugs and improvements. Microsoft almost seems to have invented the strategy, 85% completed, so purchase the next release for only $69.95 and so forth.

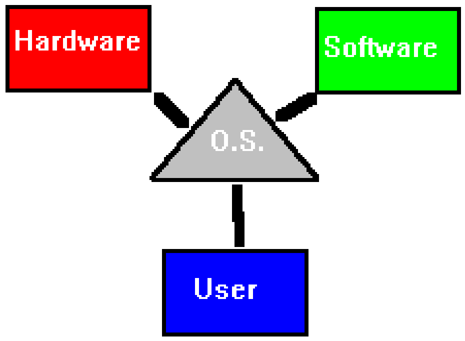
> **It is interesting to note that TurboPascal and most all versions of MS-DOS sold generally at $69.95, seems to be the magic figure for painless sales. Today most CD based games sell for within a few dollars of this amount and sales are brisk.**

I fully believe that if this new operating system is to be offered for free - it will fail and we will all lose. Forth in general, even today, is still looked upon as Hobby Time, and even treated as such by most of you whom will read this document. I implore you with all that sings out from your creativity to stop playing and get really, really serious. Professional software is not free, when it is it gains and holds no real respect in the world market. Free will not help us all retire. $69.95 a copy can and will. Lots of people, read firms, have made a reasonable living offering Forth products, but with on going difficulty because of FREE. Let us all drive forward into the future together and make Forth pay us, we have paid into it long enough, we have supported it too long as members of our own non-profit organization, we have cuddled, petted, cherished and loved it all too long, now lets reap the rewards of all Forth has to offer from what we have made it. Look how much Bill Gates has made with Microsoft, even 1/10 of 1% of that would set us all up beyond our wildest expectations. IT CAN BE DONE - LETS DO IT!

We need one solid Operating System core to build upon, not twenty different ones, we must be of one mindset, with all the personally derived extensions one wishes to create, or can imagine, to follow.

The Concept "Operating System"

Simply stated an operating system is "an interface between some hardware, an application program and a user."



It's interesting that back in the day, when CP/M was the common operating system, I could load and run an application on any system vendors offering, while every vendors offering was a completely different hardware configuration, even disk format, except, for the processor and Operating System. As an applications creator I had to generate no extra code to accomplish this. I could query the display device to determine if it could perform graphics and/or color. Simple. All the display routines worked regardless. The hardest part was to copy my application code to the appropriate format diskette the customers hardware required, this really was never a big issue.

In reality all applications, especially under Windows anything, is nothing more than an extension to the operating system. With the exception of communications programs, and a few utilities, no application ever connects directly with either the hardware or the user devices.

Bill Gates maneuvered all applications programmers into having reliance on the operating system to such an extent, in the form of DLLs - also know as code you do not have to create we provide you - that Windows became entrenched. All of us have seen graphics panel based programs that run fine under DOS with no Operating System extensions, and almost every one is smaller than its' Windows counterpart, and easier to write I will say.

Small Machine Models

During the CP/M days MS-DOS was in wide usage in Europe, and unlike CP/M running in under 2KB, required 48KB memory for the Operating System and all the resident utility code, leaving only 16 KB for applications. The CP/M approach of "load only what you need at the time" was disparate in the MS-DOS memory pig world. Already the writing was on the wall about poor code design resulting in using lots of memory. Microsoft operating systems designs have continued this glutinous trail right to the present where unimaginable amounts of memory and storage are in use today for even simple applications.

Remember MP/M and Cromix, four users running just fine on a 4MHz Z-80 with 48KB of bank switched memory each. CP/M-86, aka Concurrent Dos or CDOS, came out for the PC platform just as Clone Turbo PCs were unveiled, a single user version of MP/M, I still have a copy, which allowed you to run four things at once and hot switch between them, two years before windows existed. Interesting that CP/M always accessed disk faster than MS-DOS.

Until Windows NT Microsoft products did not really do true multi-tasking, event driven background tasks just do not qualify. Why did it take Microsoft ten years to catch up to the technology Digital Research was using, still not being as fast or compact even yet?

Just for clarification: Tasks are user or application initiated, while processes are O.S. initiated.

When Novell introduced their first version of Netware, version 1.01a on about thirty 5.25" diskettes, I installed it on a 6 MHz true AT with 640 KB memory. That network server with 10 MHz Ethernet interface cards could get data to my PC-AT workstation faster than I could load it off the local hard disk, while supporting seventeen users, nine doing accounting functions. These are our roots. Imagine that same operating system on a 300+ MHz Pentium. This does point out a special thing I am trying to get you to notice - that all processors are I/O bound, and that more code does not mean better. All Microsoft did was use the extra processor time to draw pretty panels, until the panels became so complex that the processor became the bottle neck, so they revised it and called it Windows 3.0, newly rewritten in 16-bit code. It was the newest innovation for non-Mac users, and so it took off as everybody wanted the *new interface toy*.

Medium Machine Models

Now where would we be if I did not bring mainframes and minicomputers into the picture? Lets cover UNIX and similar operating systems first.

In 1980 when the new UNIX was just being turned out for nothing or close to that to the universities, the new IBM PC came with only 64 KB of memory and a cassette port, the Apple II+ was a big hit in educational circles, up popped a little operating system called Qunix, which Bell Labs later insisted was named too close, and thus QNX. When first released this Operating System was multi-tasking with all the standard directory capabilities, networking and device driver shuffling of its' big brother, however, it did not suffer from disk partitions at that time. It did lack a "C" compiler at that point, which slowed its' introduction for about a year. Unfortunately the two partners residing in Canada decided to screw the one in the U.S. and shut down the operation just as it was taking off, right after two Americans wrote the "C" compiler for it - investing and losing everything they had, it has never regained any real market share since. Today QNX in used embedded.

We hit 1981-83. At this time, also, several Motorola 68000 based Bell Labs licensed UNIX machines came out like C.C.C. and the Venture Systems single user unit for accounting. CP/M-86 and MP/M-86 were just introduced about this time, just before Godbout started sinking, Digital Research had just laid off about seventy percent of its' staff, about eleven hundred people, a tragic dead end, cast so into the dirt. SCO introduced Xenix running on a custom box, and at that time it was a screamer. Banyan, 3-COM and others joined the network market with dedicated servers running proprietary multi-tasking operating systems.

HP., DEC, Wang, Perkin Elmer and Sperry/Univac all had proprietary operating systems that will not be talked about, all had their place. So where are we now? Well, we have Sun Solaris, the HP. 9000 flavor, Linux, Bell Labs UNIX versions on various platforms like UniSys and Silicon Graphics, and even Amdahl's version.

So you are now saying 'what's so wrong with UNIX'? Well let's see, besides its' cryptic naming conventions, file oriented everything, lousy disk partioning with slow access, impossible utilities, the ability to create circular directory paths, the carry over of its' original design of being an Operating System for multiple people on terminals, perhaps nothing. It has a place, just not on my desktop. It should interest you to know that most UNIX based engineering software vendors have been converting everything over to NT. The great following for Linux points out the fact that people want something other than what is available. Note that Linux is successful because it is not completely free. Just try to download enough Linux stuff with directions to boot and install it, good luck, not a task for the feint of heart. Linux was written largely by UNIX oriented people whom wanted something else or newer, and also saw that existing and low cost equipment could be leveraged. This effort, however, is to be admired.

Large Machine Models

I must talk of the Burroughs Medium Systems Operating System called MCP (Master Control Program), the main writer of which implemented the Operating System for the HP. 1000 and 2000 series and its' derivative is still in use today on HP. 9000 series machines. The MCP runs in 100 KD (kilo digits, nibbles) or 50 KB of memory. The concept and execution here is pure beauty. The MCP and the machine are very tight in operation. The medium series, 4700, 4800, 4900, V, mainframes were, in essence, state machines with just enough operating system to invoke task switches and deal with the operator, funny Intel processors run in protected mode almost the same way. The MCP did not directly swap code to disk, sort files or interact with IOPs (Input Output Processors) or DLPs (Data Link Processors). By the way, there are almost 400 commands available to the operator.

Lets look at this a little deeper. This is a little bit hard based on the fact that the 4000 series Burroughs was designed for a specific purpose - record processing. This specific machine was, and to quite an extent is, the mainstay of the banking world and the military back office accounting operations. Though dated, this line of machines designed in the early sixties, is very functional through an ongoing maintenance and upgrade program.

The CPU runs at a blinding 8 MHz, with a possible 4000 KD memory, yeah 2,000 KB, and is able to process up to 60 jobs at once. A miracle? No, just good engineering. The machine can move a sustained 60 MB per second across its' main buss, yes, it's a 64 bit machine with parity for a total of 72 bits on the buss. Can a Pentium do that, even close? Imagine seeing 16 GCR 200 MB 12 inch tapes spinning, 30 or 40 jobs active in the mix accessing 80 GB of diskpack and head-per-track drives, and four 2000 line per minute chain printers loaded with 132 column green-bar, all running full tilt, connected to this one machine. RESPECT! At 8 Mhz .

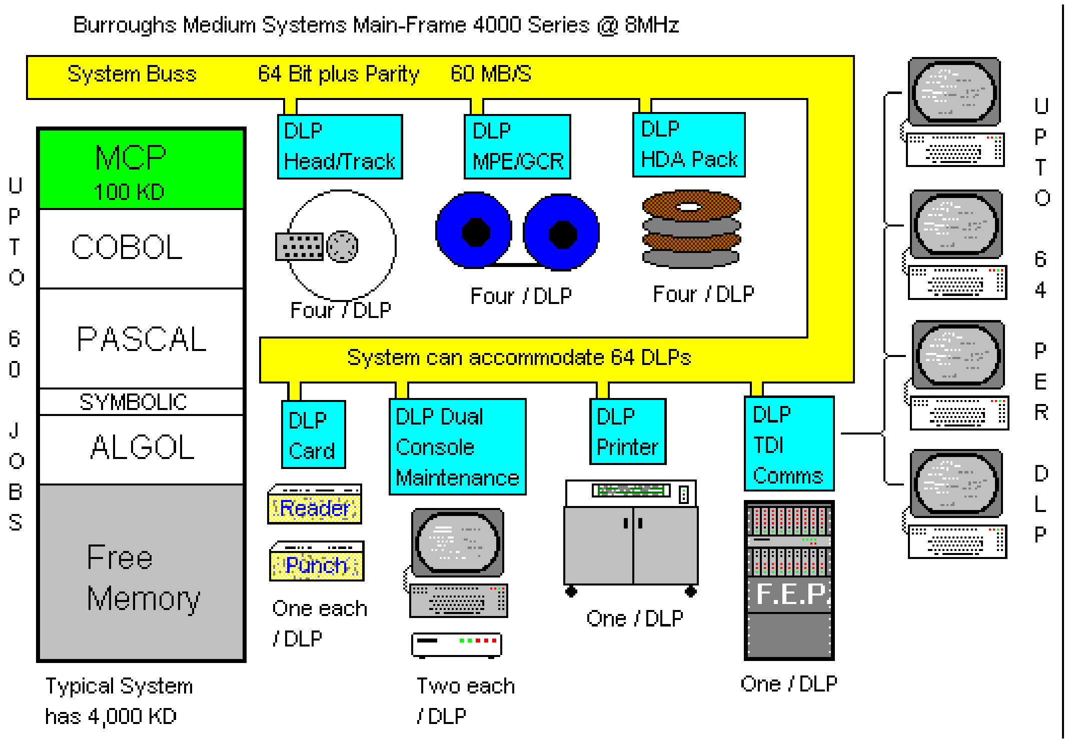
Here is the real trick to this - when an application, with a task clock running against it, does any access of any kind a task switch occurs while that access request is performed. The CPU turns control over to the MCP which then switches to the next application and its' last point of execution. Processing continues for the application when the access request is completed. Ahh, you see the ghost in the machine, the DLP. When a program executes say for instance an 'Open File' statement, that instruction is literally passed to the DLP to be executed, a program branch to MCP occurs, and another process takes over until the DLP is done. A disk, printer or tape controller is a DLP, one of many, each with a 4MHz 8080a and dual port RAM on board. If the clock runs out the task switch is forced. By the way the hardware has a sort intrinsic done by the DLP, and no accumulator, calculations are done in program memory.

An DLP is very much like modern co-processor interface cards of today, like the Intel Ethernet LAN cards, each which has an 805286 at 10 MHz, handling most all aspects of LAN communications of either 10 or 100 MHz. SCSI is another example, a block mode device setup, multiple master capability as a tape drive can backup a storage device with no host interaction, that normally uses the likes of an 80186 embedded. Even the keyboard interface is an embedded processor on the motherboard communicating over a serial LAN to an embedded processor in the keyboard, both usually 8048s. If you watch a statistics screen on a server for a time you will find that eighty percent of overhead is dealing with the LAN, unless you have a smart LAN card, almost all are today by employing a 10 MHz 80186 co-processor doing the overhead. An IDE port is a simple parallel port, the embedded controller on the disk drive itself doing the interface. If we view these smart devices in a little different way, then we can start to really use them better, and the OS benefits greatly, leveraged multi-processing, code size reduction and I/O bandwidth improvement.

One unique feature of the medium systems, that IBM still may need to figure out, is that when a job finishes its' memory is released and all the jobs below it (address 0 is at the top on this machine) are rolled out to disk and back in to fill the gap. No checker-board memory. All done through hardware, and embedded processors.

> **I helped a friend develop the method of doing this under program control. We went through the MCP source listing and found the instruction that does the roll out, and thus I introduced COBOL to the concept of heap, that can be resized during run time. COBOL has the need to have all data space predefined and sized at compile time, imagine being able to examine the data and then get only the needed amount of memory, never the always-the-maximum scenario.**

One of these machines with an SRI supplied HP. 1000 Front-end can support over 1,000 TDI (Terminal Data Interface) serial loop, block mode transfer terminal users across local and single-point or multi drop analog or digital lease lines. Remember all data to this machine is a record. Isn't all data on any machine? A byte or a thousand bytes, is there really any difference? Still just a record, yes?



Generic Model

Since I have used all the various types of operating systems I can draw parallels between them. Most of you have also, so you will see what I am describing. On a PC running DOS or CP/M you interact with a single process on the display, the same for UNIX with no graphics interface. The fact that things, such as daemons, are running in the background does not fully constitute a multi-tasking environment. A good example of this is a mouse driver, its' running, scanning a port and feeding information to an application, as an extension to the Operating System, but it has no user interface. On a main frame you never interact with a process directly, you instead are always viewing a systems log on the display and issuing commands to guide those processes, everything is running in a batch mode. What a windowed or multiple view environment provides is the ability to interact with running processes, see simultaneous updates. Humans are serial in action. Yet in all this we can, and do, only really deal with one process at a time, either by inference or bringing it to the foreground. I believe that every process should have a display that can be viewed by the user, no blind daemons or drivers or other processes, but you ***do not*** have to view. The fact that processes can be made to display provides the ability to interact, which provides the ability to stop or realign runaway or undesired situations.

As you can see from all this, in its' many forms, operating systems are implemented in many yet very similar ways, always for the same purpose, always to solve the same problems. All are oriented to servicing one, or multiple, users, or tasks, interacting with one, or more, applications and/or storage and/or communications devices.

I believe that I can state without rebuke that portability is not even a question. What program or application since the beginning of the industry has been moved to a different processor or operating system without serious modification? I can't think of one. So I state that the term portability not be used while the operating system resides on one processor platform. With some careful planning and much research OS/Forth could be designed to have continuity with the application program interface, much like COBOL, allowing a recompile on the new processor in order to port an application. Perhaps this is acceptable based on current processor speeds. But, I will mention what is known, that to gain real performance processor specific code must be written, and thus, the application is no longer portable.

Operational Modes

It is important to mention Real, UNReal, Protected and Virtual mode differences in here some place, pretty much an Intel specific issue, though other processors have differences such as Standard and Virtual modes, all similar to each other in concept and function.

In relation to Intel processors

Real mode, not the natural or power-up mode, of the iAPX86 line of processors is what most people are used to dealing with when in MS-DOS. In this mode the processor can only address the lower 1 MB of memory, without special extensions. The processor has default address ranges set when you enter into this mode. We have lived with this situation because the system BIOS is a real mode program, setting and leaving the processor in this state. Most of us are quite familiar with this mode of operation. Most memory managers, Expanded Memory Services (EMS) in particular, switch into protected mode and page or copy a block of extended memory into the 1 MB range for application access. Remember High Memory is relocated on most motherboards in hardware, not software, recovering the address range overlapped by interface card start-up ROMs in the address range of 640K - 1000K. Real mode does not allow the processor to do its' task table magic, thus the application must do any tasking, if any. Real vanilla DOS.

UNReal mode, is very much like Real mode, with one exception - memory availability. To get into UNReal mode the application, or O.S., switches to protected mode, sets all the memory limit registers to their maximum, which should be the physical memory size, and then re-enters Real mode. This leaves the application or O.S. with full directly addressable access to all memory in the system, no more 1 MB limit, if the BIOS does not get in the way. This is pretty much how processors like Motorola's offering operate, all of memory available, or as much as allocated, a linear or virtual space directly addressable, without the protected mode switching troubles for start-up.

Protected mode, is the '386 32 bit mode of Windows, which did start with the iAPX286, and continues today in all Microsoft offerings. The application or process is loaded into an allocated memory segment(s) and run. The descriptor tables, explained in the following section, used by the processor and loaded by the O.S. control the access, task switching, memory usage and exception handling for each task, process or application. O.S. service requests branch to the appropriate service code while invoking a task switch. This could of course be a single application utilizing the entire machine with no operating system. Ninety percent of the multi-tasking process is performed by the processor with no O.S. intervention.

Virtual mode, V86 or VME is a fall back extension to protected mode. If you are running Windows NT and open a DOS box session, that session or task is running in an emulated lower 1 MB address space. The DOS shell that comes up is only a partial DOS, with access rights and restrictions to most I/O and devices having been lifted. Stable DOS applications work fine, even accessing networks, all the protected mode service code is still running, and task switching continues in the standard protected mode manner. The normal DOS function requests are redirected to protected mode code.

Other types, are what the rest of the world does. It is interesting that even the 6809 operated in the linear address fashion. An early version of OS-9 ran on this processor and provided multi-tasking. Intel seems to have gotten stuck on the 8080 address method, it's still with us. Motorola, like others, provides nice processors with linear addressing, which provides a task with an offset base address, and perform multi-tasking just fine. An assigned address range is still assigned, even when in a linear space.

Multi-tasking, Really

This is a fun area, given so many different ways it can be done. Initially we must address things from an Operating System perspective, then from that of an application. Without going into jargon, there is so much of it, lets list the various common low level tasking methods. Specifically the point control is passed (remember we are dealing with Intel processors only at this point in the discussion).

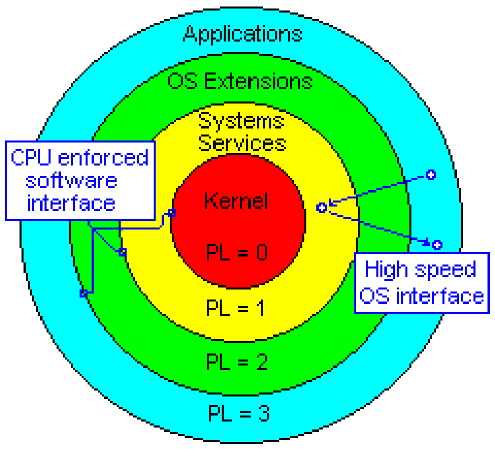
It is well know that table driven systems perform at the highest levels. The least number of times data is handled the faster things run, and less total code is required. That's right, you got it, we are right back to square one with a record processing environment. It should be no surprise therefor that the Intel processors use a table driven system to maintain task switching and memory management functions, other processors have similar schemes.

The functional areas covered by each hardware and software overlap, the intersection set in set theory, which is the common inter-operable area between the two. The Core or Kernel software sets up the task, privilege, interrupt, exception and memory usage tables used by running, or to be run, applications and processes and then turns actual tasking over to the processor which uses those tables in a state-machine style of operation. So the Core or Kernel software deals with memory allocation, task load-start-stop-flush and exceptions. Driver, service and process code run like application code but with higher privilege and access level.

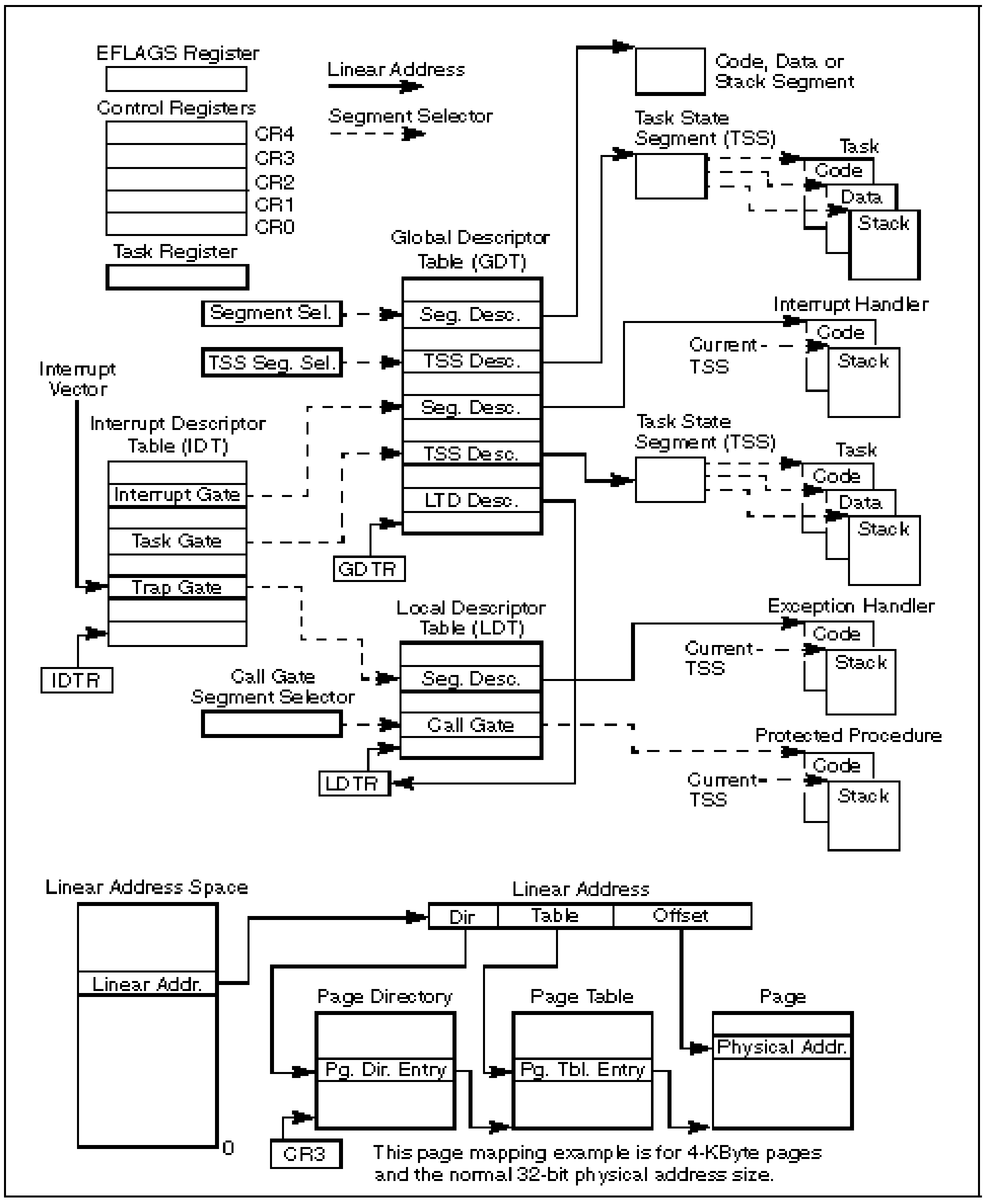
The Hardware End

I will not go into great depth here as all this information is conveyed in the Intel reference material and various books on the subject.

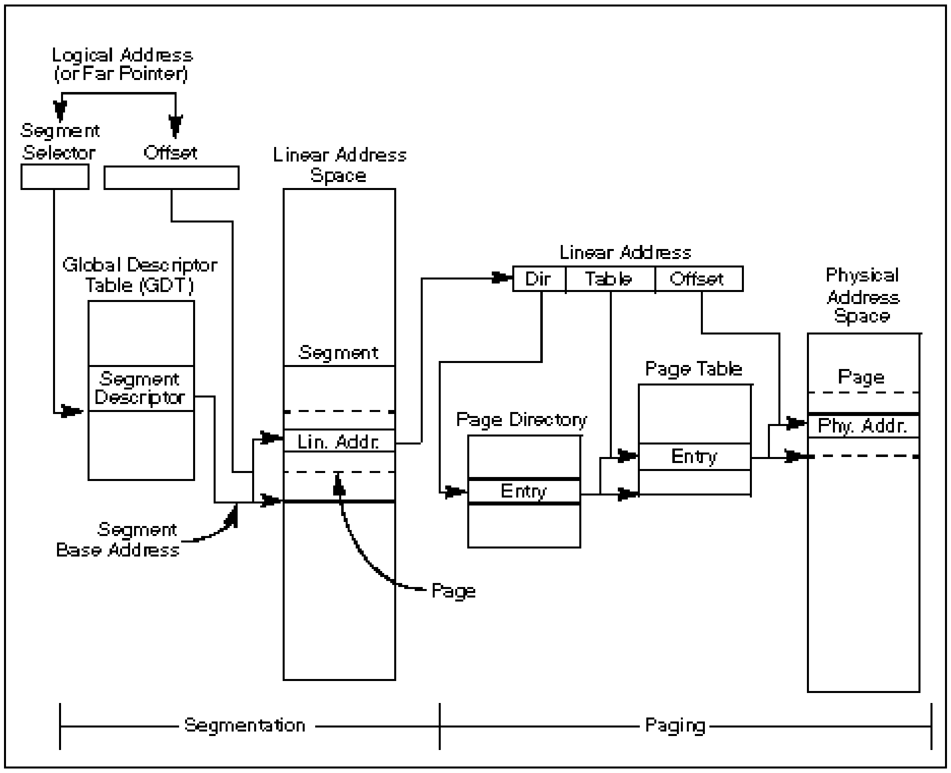
Intel designed in a four level hierarchical protection system, privilege levels 0 to 3, zero being the highest or most privileged level. The processor farther provides control as to which level of privilege can perform hardware I/O instructions. Diagram follows.



There are several Descriptor Tables defined each with its' own location register. These are the Global Descriptor Table (GDT), Local Descriptor Table (LDT) and the Interrupt Descriptor Table (IDT). These tables and the Task State Segment (TSS) are the core of the hardware portion of the tasker. The descriptor table size can change based on the number of task descriptors contained there in. These decriptors contain the segment base address and length assigned, flags as to task gate switching and activity semaphores, privilege level, I/O permission bit maps and the content of the registers if the task is inactive. A table driven system, privilege based, orchestrated by hardware, administered by the Operating System Kernel. Diagram follows.



The CPU also handles a memory page table scheme and an address facilitator cache called the Translation Lookaside Buffer (TLB) which holds the 32 most used memory segment and offset in the processor to improve task switching times. Also, there are memory access privilege flags held in each tasks Descriptor record. The normal methods of using a segment register and an offset in a second register, with all the common addressing modes, still works normally. The CPU actually is translating segment + offset addresses into linear addresses for physical memory access. In protected mode you can not use the Segment Register increment trick to move through a buffer in chunks, this will cause an address exception, you must use the Offset Register and mess with the Segment Registers as little as possible. Diagram follows.



The Software End

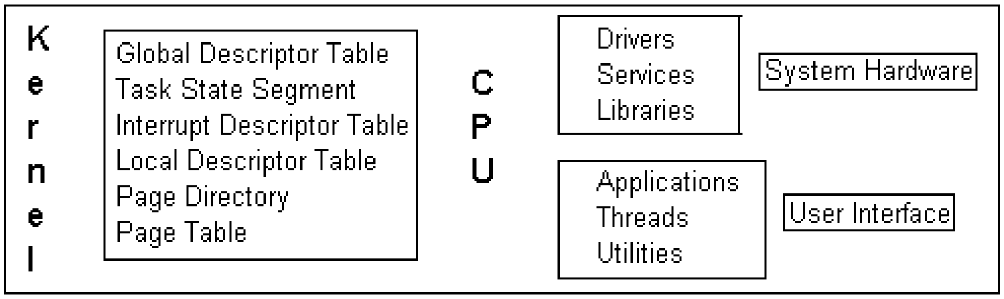
So this is where the Operating System Kernel comes in, the only software on the system that executes in ring '0', the highest privilege level. The number of times per period that any particular task is executed is the reason for a kernel. The simplest way to execute a task more times is to place it in the task table more times, preferably interleaved. In all truth loading an application, setting memory up for it and locating it in the task table are mundane executions for the Kernel not any more difficult than scanning a directory tree.

The CPU performs a task switch based on any of the following instructions or occurrences: CALL, JMP, RET, IRET, INT, a trap, fault, abort or external interrupt condition.

Task Switch Points

The application enters pause condition.
During application calls to a library or routine outside its' memory partition or privilege.
During or after any hardware or software interrupt occurrence.
Expiration of any task time slice clock.
Operating System forced.

There are several instructions, not found in many vended assemblers, that facilitate access of the Descriptor Segment content and CPU control, some are LGDT, LDS, LIDT, LLDT, LMSW, LTR, SGDT, SIDT, SLDT, SMSW, STR, WBINVD, and ARPL.



It really is interesting how we have come to accept the strange operation of software and hardware as either a user or developer. Different applications switching modes or screen display formats, and we don't really notice or care - given the transition time is short.

The Intrinsics of an Operating System

The intrinsics are basically the services that are provided, the actual interface options to the hardware, the basic machine presented to the applications programmer. All this comes together more or less as the personality of an operating system, the way it feels, the freedoms or constraints enforced, its' ability to help one complete work, the actual interface presented to the user. All in all more than look and feel, even the exact same models of automobile feel different in some incommunicable way.

> Most of us have written assembly language, dumped programs in hex, been through the guts of both CP/M and MS-DOS, even Windows, and spent untellable hours in front of debugger and blue ice boxes. This stuff is pretty generic as far as code goes, 32 bit is the only really different aspect, though not even that anymore.
>
> > I remember the first Forth system I had, still have. It was MVP Forth, which Glenn Hayden had come to my house and personally installed. I remember he had to make three trips, the first to get things started, the second to accommodate the 128 KB provided for CP/M-3.0, and the last to allow access to my 10 MB hard disk. It booted on a floppy and used sector based blocks. He did this all for the very low price of $39.00, the price of the package I purchased. This is a text book example of personal integrity. My gratitude and respect to you Doctor Hayden, although, I may not thank you for this strange path you long ago dumped me upon. How many sleepless, restless and heart felt frustration wrought nights have I spent tinkering, creating, implementing, dreaming, planning, seeing the clarity of concept, at 3am finding myself yelling at an empty room, having visions with no words for expression, image of concept others never seem to see, artistic expression.
>
> **The Punch Card way** - Eighty columns, key it wrong and start over, many boxes full, read 'em in, compile, test run, print them out, scratch head, punch changes, collate, repeat.
>
> **The Teletype way** - Supervisory Process Operator (SPO) Log running on, page after page, read, enter direction commands, transmit, more SPO output, repeat.
>
> **The CP/M way** - Because UNIX serial terminals were widely available, had standard control codes, and did not use a very fast interface they became the choice. TeleVideo, Adds, Beehive, etc. The 8080, and shortly the Z80, microprocessors operating at 4MHz could easily handle 9.6 and 19.2 Kbps data rates with hardware or x-on/x-off handshake. The screen scrolled off the top, like DOS still does, like a teletype SPO log. Soon programs started to use direct cursor addressing and so the birth of formatted screens with line character panels. Some terminals had multiple screen buffers and display screens could be preloaded and switched to in a refresh clock time. Worst case the screen could be rewritten or drawn in about one second. Later CP/M versions had a scroll-back feature so one could look at commands or listed items that had scrolled off the top already. Some terminals had graphics capabilities and thus the foundations of Windows.
>
> > **What you didn't need you didn't load; so why does MS-DOS keep all the utilities loaded? With an Operating System that normally fit in 2 KB of memory it is easy to see how simplicity works, and there were just as many I/O ports and storage devices then as now.**
>
> **The MS-DOS way** - On the PC platform, larger programs but not really any different than CP/M in fit, function and form. The display speed improved as it was parallel and operated at peripheral buss speeds. Some of the functions of the serial terminals got lost in translation and all the display functions take main processor time, a trade off for interface speed.
>
> **The Concurrent DOS way** - This was a nice mix of CP/M and MS-DOS commands with the ability to have multiple sessions display on one screen or on several serial terminals. I used CDOS on a single display system and ran background sessions that could be hotkeyed to the foreground, or tiled. I could compile, copy to diskette, print and edit all at the same time on a 4.77 Mhz machine with no problem whatever. In the last version Windows could also be run, excellent memory and processor time management, a smooth fast environment. Too bad about DRI.
>
> **The Windows way** - You know this one. The screen is laced with a background, then the application panels are laid on top. Panel, icon, .gif or .bmp .. there is really no difference, the screen is the main panel and everything is laid on top. Some items invoke action, icons for example, some are interfaces to running processes or applications. Enough.

The OS-9 way - I can't say much here. My short exposure was on a GIMIX system that had multiple terminals, it acted and felt very much like a terminal serving UNIX box, just different commands. All the programming languages were re-entrant resulting in very good memory utilization.

The UNIX way - In operator, or super user, mode this is a cross between DOS and a mainframe, a single scrolling screen, task lists with the ability to control, mainframe mode. Direct execution of an application places it in the foreground and the operator interacts with only it, DOS mode. X-Windows was the big leap, being able to interact with panels representing multiple running applications or processes. Windows is as Windows does, panels. Users can still be connect by terminal via serial or LAN and run in terminal or X-Windows modes, display device capability dependent. The command set is still script or batch oriented when in command mode, it still rubs me wrong, so much ability traded for cripticism.

The Plan-9 way - Very X-Windows like environment but with the underpinnings of a true multiple processor or distributed processor environment. All things are broken down into client or server parts and flung about. I question this approach with today's cheap and very fast processors, it seems this approach would have been better suited in the age of slow processors, oh well a day late and a dollar short. I can see this approach working in very heavy processing environments that are doing heavy calculations and graphics displays while looking up data, and the technique is part of how OS/Forth works on a single CPU, but LAN limitations could quickly show up. Functionally impressive but expensive.

The MCP way - A teletype on a screen with the ability to scroll back as far as you like between log roll-outs. Commands are directed at the MCP or jobs in the mix. A rich command set, about 400, makes for excellent control, but this is a batch machine with no direct interaction between the keyboard and applications. Terminals that are connected to the system interact with an application, in semi batch mode one block transmission at a time, typically CANDE, the terminal services application.

The OS/Forth way - I should comment ... but instead I suggest you keep this thought and read on. All is revealed a page or two from here following.

Programming Languages

Unlike other languages Forth takes time to learn, really learn. If you are proficient in one of the common languages, you can become good with another in two, three months on the outside. But, Forth takes a year, truly, to grasp the real nuances, the true capabilities available, the freedom. Forth - highly data-typed, yet unstructured, with complete freedom of syntactical expression, uses unhindered data passing and access models, is truly extensible, and is result oriented. It must be stated, however, that a Forth program becomes, in a unique way, very structured from the perspective of the data being dealt with as once the data handling interactions are described they must be closely followed.

It takes time to unlearn the ways of wagon pulling and invent the rocketship. If going back to the common languages, where one writes for a complier, you constantly search for and invent Forth like things to end or reduce the constraints ever present. The mind is the forge that melds concept, machine, data, form and process into a salient solution through the use of Forth as hammer and anvil. Adding words to the compiler itself is true extensibility - not the act of calling function or code libraries.

If examining all the programming manuals, references and book offerings it becomes very clear that all programming languages, completely independent of the physical processor type, refer to creating code for a specific operating system. Even books on specific subjects, such as EGA and VGA programming, constantly refer to the operating system interface to perform functions or gain access. Does this not then point out all that is really ever done is creation of code that runs under the operating system? Which really means that all applications really only are extensions to the operating system. It is commonly referenced that the application runs on top of the operating system, but in actuality it runs under the operating system. The concept is simple - if a program runs on top of the operating system then it is superior and does not need the operating system. This is to say, as example, the operating system runs on top of or around a device driver, an application also runs inside or under an operating system, merely doing something different than a device driver. Most embedded applications have built in device drivers and do not need an operating system, just operating system like extensions, a reversal and the reason most programmers never fathom embedded concepts.

Look at all the programming languages that you know, except assembly flavors, and then tell me the percentage of function extensions available that do not deal with the operating system. Following I use the DOS version 5.5 of TurboPascal, not including the overlay, graphic or object functions, as an example for clarity.

O.S. Impervious (48 ea.) -- abs addr arctan chr concat copy cos cseg dec delete dseg exp fillchar frac hi inc insert int length ln lo mark move new odd ofs ord pi pos pred ptr random randomize round seg sin sizeof sptr sqr sqrt sseg str succ swap swapvectors trunc upcase val

O.S. Interacting (101 ea.) -- append assign assigncrt blockread blockwrite chdir close clreol clrscr delay delline diskfree disksize dispose dosexitcode dosversion envcount envstr eof eoln erase exec exit fexpand filepos filesize findfirst findnext flush freemem fsearch fsplit getcbreak getdate getdir getenv getfattr getftime getintvec getmem gettime getverify gotoxy halt highvideo insline intr ioresult keep keypressed lowvideo maxavail memavail mkdir msdos normvideo nosound packtime paramcount paramstr read readkey readln release rename reset rewrite rmdir runerror seek seekeof seekeoln setcbreak setdate setfattr setftime setintvec settextbuf settime setverify sound textbackground textcolor textmode truncate unpacktime wherex wherey window write writeln

What About JAVA

Lets discuss some about Java, and other languages as well, comparison is needed. Because of the subject matter at hand this was not included in the 'Programming Languages' section.

So I ask and answer "what is Java"!? A new version of 'C' that compiles to a non-processor specific output and has standardized service library interface. Java can not be executed as Forth is, but almost, only after target compilation, but that's true about any language code fragment. The term 'Java Script' continues to pop up in all forms of communications, and I find humor there in, as any computer veteran will tell you: a script runs like a batch file, it is source text that is interpreted and executed. Java is not a script language, it is a diminished pseudo code or p-code, having been compiled, to be interpreted by a host application. Funny, to think Forth can be processor targeted, p-coded, scripted, interpreted, linked and even assembled, or all of these, based on the requirements of the application. P-code compiler output can be arranged such for efficient interpretation, but we are back to the question 'are we writing code for the compiler and libraries or for the user, application and data!?'

Another of Forth's strengths in this situation is the ability to create and test code interactively, then save same in a compiled format for later use, graphically speaking - not unlike a graphical button face in .GIF format.

The beauty of OS/Forth is that you can embed any language or instruction constructs in the display pages and have them still work fine. To that, you can have application code written in any language, with a services interface library. To make it all work only requires the correct execution module be loaded or available. Forth just happens to be best for such display formatting as the code layout is almost completely linear, there is very little execution overhead, whether interpreted, compiled, indexed or address threaded - all of which means more time being devoted to actual application generation and execution, not compiler pleasing. Languages such as Java, and any 'C' flavor, Pascal or even Basic, if interpreted, could never execute as fast because of their structure, all of these languages were designed to be compiled. That code oriented structure is the impediment, while Forth is data oriented and processing flows around the data stream.

I would rather see Open Firmware executed than Java based on its' linear design, not that it is of Forth heritage. It can be read on any platform, it is p-code like, can be reverse compiled, is compact and executes quickly ... well it's Forth, right? This all points out that we need to restructure existing languages, or create new ones, that are more linear in design. Forth is not the only answer, just a good one, but one size does not fit all. so .. Java ain't it.

Consider the current situation with browsers, they fetch and read a file, display the information according to the embedded tags, place non-scaling graphic images as designated, interpret any code fragments or macros, wait for the user. That really is all that they are doing isn't it? The most difficult of this whole business is the browser must change the HTML display information given, and place it in, the display constraints of the host operating system. *The* reason we think of a browser as an application, not unlike a wordprocessor or database program. You do see the problem here: we continue to interpret the display source format data for the operating system instead of integrating it into the operating system, or actually, the operating system into it. I have always considered that the data was more important than the operating system, that vantage seems to have changed along the way for others, thus Windows. HTML works, maybe not as well as wanted, but it does work. In future version releases more and more functionality will creep in until there is no difference between any display source format.

Forth broke away, well Chuck did, from the norm of the computer world with its' record oriented punch card traditions. The beauty of structure is how each of us creates it, not what we are forced into.

So go ahead - Java, 'C', Algol, Pascal, Basic, Cobol, Fortran, Assemble, Lisp, Snowball, SmallTalk and Prolog your heart out - OS/Forth can accommodate them all, with faster execution than any other operating system, in less space.

Other Devices

Now here is an interesting area for appliance and equipment makers alike. Why can't I send HTML directly to a printer not unlike postscript is currently. Soon HTML will provide drawing commands, perhaps like Turtle graphics or defined possibilities, at that point postscript will be redundant. I think it bad design when I must convert a graphical image to text for the printer to deal with it, something HTML negates, send the image and anchor, screen or page - no different.

Everything was based on the NEC SpinWriter or Diablo 630 for a very long time, then Epson, followed by the first HP Laser printers, now all kinds. Each printer type has its' own graphics capabilities or emulates the HP series. The same goes for plotters, HP or Gerber, another I can't remember right now. This nigh on two decades, it's time for something new, more robust, oriented around the user and data.

One graphical format which is visual either displayed or printed and that everyone knows and understands.

The OS/Forth Concept

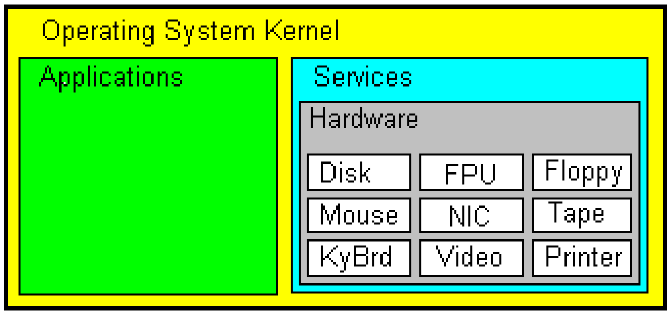
The OS/Forth concept is based upon, or a summary of, many of the concepts presented in the preceding sections all rolled together. That is kind of a broad statement I know, but lets see.

Understand first that OS/Forth is not a self-booting stand alone Forth implementation - it is so much more.

so why this

The entire thrust of the concept for this new operating system is to factor the problem at hand as far as possible, provide that position to the application programmer, and reduce total size of all code running while gaining speed. In the purest sense we are treating each aspect of the operating system and all applications and utilities as separate vocabularies. If the operating system can open, close, read, write, buffer, sort or index a file why should such code be in my program? Why should my program need to update the screen coordinates in the program control file when the operating system can, and should, as it has the information first and interfaces to the storage system, my program really does not care the position of the displayed panel. So we are back to the basis of Forth - don't duplicate code over and over, leverage what you have already created, refine and factor. In this method we immediately realize that a performance improvement or gain in one part of the system transfers to everything, and the reverse.

In the OS/Forth environment all processes that are not an integral part of the Kernel are tasks, which says that all code runs under the Kernel. All non-kernel code is a process, service, driver or application and all are both clients and servers depending on the moment of execution.



startup

Control files for each task designate the location of files, first screen format or file to display, location and type of the application code, normal memory requirements, last displayed panel position and its' details, general settings and other miscellany. Similar in operation to an .INI or registry file. The operating system deals with this, not the program. The control information is a record in memory and read or written by the Operating System, so let it deal with the updates, less code in my application. This provides the situation of stopping an application at a task switch opportunity and rolling it out to disk intact, able to be brought back up later, great for transfer to another system or processor, fault tolerance or memory cleanup.

screen

I like screens, lost of them, often several when doing systems and network administration work. There are many utilities that need to be run and monitored and one screen does not accomplish that. Do you open only one book at a time to one page, close it and open to the next page, or switch books in the same way while doing research - I think not. I can often cover a whole table with open textuals and scads of notepads with points or references jotted down. One screen only does not allow me to properly move through the data stream I use. Minimized or half visual panels are the same as closed books. I should be able to fill the buss with display cards. I do not operate in a completely serial fashion and need a system with more accommodation than current ones.

I must mention HTML I guess. Most applications today were designed with screen painters that generate display files, so is there any real difference deciding that HTML is a good all around non-proprietary format to use? Hitting version 3.x level in the HTML standards in such a short amount of time shows that as need arises the display capability will improve. An Operating System that does not need a browser is the natural next evolutionary step. But, remember, a firm such as Netscape could be the writers of the HTML service module, or of new and improved versions of it, and find a solid place in the industry. A small change in scope or direction but a large potential and growing market for them to service. Oh, Yeah, I want an advertisement blocking extension, like, real quick.

panels

The two aspect states a panel should exhibit are Scaling and Fixed with a Local, Remote or Shared Presence. To begin a reasonable description I must point out something I may be taking for granted - all panels are in graphics mode all the time, this based on the fact that current video controllers convert text to graphics for display anyway. So we may as well be in the same mode from jump. So to simplify this, all:

- Display screens are in graphics mode
- Panels can be dragged between screens
- Panels must be anchored on a single screen, no visual overlap (for now)
- Panels can be a desktop, like current backgrounds, and have icons hung on them
- Items hung on a panel move with it
- Panels are controlled by the display services and not the program

The concept of icons on panels is very powerful. If you have a panel up, perhaps in what seems text mode, and there is a help button in the upper left corner, the button as an icon can launch an unrelated or disassociated help application, spawn an associated help application or be part of the application and flag an input event. The Help function itself is part of the Operating System, looks like any HTML you design, and is extensible by you.

Aspect is a special feature that continues to be overlooked, but, no more. I can change the size of a graphical image but not a screen panel, I don't like that. We were given scalable fonts but were denied scalable panels. If a panels' aspect is in Scale mode then it and content should scale up or down from thumbnail to full screen. If a panels' aspect is in Fixed mode then the content remains the same scale, with scroll-bars when shrunk smaller than the display content, which means it looks like panels we see today on other operating systems. Both modes are needed.

Presence is a new way of stating something forgotten. What is meant by Presence here is the panel display source. Does the panels' attached application reside on the local machine, on another and you are remote viewing or one that is shared between machines. Imagine PC Anywhere that only affects a single panel on the screen, or a group screen interaction not unlike IRC but with full text and graphics, perhaps a process that is being viewed by multiple people, a panel that is being updated by multiple processes on different machines. The powerful concept of state of Presence.

> With Presence I can provide support to a person in word-processing application by seeing exactly what that person sees, and able to interact directly. I can run a backup process on a remote machine and not bother the user, able to monitor the process, interacting with it. Multiple people can collaborate on creation of the same document. And so on ....

storage

Sectors and tracks are items that should be ignored. An SCSI hard disk is a good example of how storage should work, access is according to block address or number. You can treat the content of the disk device almost like blocks of memory, a base block address plus an offset.

Remember sector skewing on early diskette drives? A practice required because of slow interface electronics, and not practiced anymore. The application never saw the fact that the sectors were arranged so that it took three disk rotations to read a track, the sector stream looked continual, the facts being hidden by the disk BIOS and a skew table.

It must be remembered that the disk directory presented us is a formatted entry point representation of what is stored and has nothing to do with the actual physical location of any data. So as long as your data is presented to you when you want it, in the format you desire, do you really care on which disk drive it resides? Perhaps we should think a little differently about how storage is referenced.

On large equipment the storage sub-system is called an array, you do not really care which spindle or spindles the file is residing on, it's all one large area. Gone are the days of drive letters to accommodate multiple 10 or 20 megabyte drives. Let us stride forward on this also.

network

As you are experienced with currently. I will state that I think the Novell method of addressing is much better than the others being employed. It was derived from TCP/IP and does similar address and routing functions. But, imagine 12 hex digits of LAN address and 12 more hex digits for station address, that's FFFFFFFFFFFF possible LANs and FFFFFFFFFFFF possible nodes, hard to run out of addresses.

services

Some of the types of services that will be needed are listed.

> Kernel (tasker and memory manager)
> HTML interpreters
> Forth interpreter
> Application loader
> Storage manager
> Storage device drivers

- Storage functions
- Floppy driver
- Display manager
- Display functions
- Keyboard manager
- Pointing device manager
- LAN manager
- LAN drivers
- Comms manager
- Comms drivers

development

Imagine F-PC, Borland 'C' or Pascal, Pygmy, E-Forth or even assembler. All with interface and storage request interface changes will work fine. Any language that has an Operating System interface library for its' standard functions will work. The development environment would need the most work as it is normally very heavily interface oriented. Compiler output needs only the services interface library.

library dichotomy

Consider a factored environment where the application held no extra code. The concept here is that once efficient code is created there is no benefit to multiple instances of that code running when not in use. That's not clear. I ask now, and discussed prior, why should my application need a record index mechanism when one can be provided by the Operating System, it has to do all the interfacing, buffering, reads and writes, so why not? This is factoring. Why do I need to qualify keystrokes when the Operating System can do it for me and only pass me data I have declared to have value? Why can't a communications program just open and read or write the Comms ports like a file? All these things point to the why of the split in philosophy of design. If I need to write a Comms driver to take advantage of specific communications gear, like a new style modem, then I need to write a process acting in server mode, not complicate my application. This allows me to change the driver dozens of times and never touch the application code. The real thing in client/server technology, every aspect of the Operating System OS/Forth steps up to this concept and philosophy.

thread model

Which kind do you like? You can do them all. Pick.

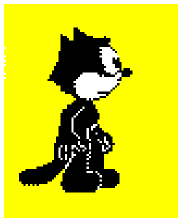
extensions

Shall we create one? How about a file copy program.

> **Copy** - Now think Forth, context and vocabularies. With the HTML editor we paint the HTML Page for the display panel, help information and an icon, then save them. We create our code fragment, as a Forth source text file, to run interpretively, we may test it interactively. We create the program Control File pointing to the HTML Page file, icon, source code fragment and its' execution type, help file, memory constraints, permissions and services needed. We register the control file with the Operating System. Upon icon invocation the Kernel opens the control file, loads the named HTML Page, loads the named code fragment with specified constraints, displays the HTML panel and then turns over control to the program, or interpreter in this case. The program requests input through Display services, displays directory information through Display services as provided by Disk services, accepts selection through Display services, invokes Disk services to complete the copy task, shows status and prompts for additional interact through Display services, exits which invokes a panel close as part of Kernel clean-up. The Kernel clears the active panel(s), frees memory, then updates the programs control file before closing it. The help function was designated as system default so the default Help Icon was displayed. If Help had been invoked then the Help services would have displayed the designated Help file through Display services in a separate panel. A program of this type would only be about fifteen lines long, less than 1 KB, since the Disk and Display services are doing 95% of the work. I only need to define a small TIB to work in and do not need to declare disk buffer space, or build a table and sort word for the directory information, or design display routines to place it on the screen, all that is already done for me in the Services, by much more efficient code. Besides, every time I make a service request then a task switch occurs, taking advantage of it, I let the requested service do things in a more efficient manner.

This code fragment as a utility could be added to a utility code vocabulary and stay resident in memory, being available for other applications to invoke, and sharing common HTML interface layout with other vocabulary words. So now applications can add File Copy to their file access interface with minimal or no additional code, explicitly called or an icon on the application button bar. A simple extension, or stand alone utility, either/or. Multiple control files can be designed for the same code fragment to suit different needs with no source changes.

OS/Forth Features



Here are **some** suggested highlights:

Displayed Operator Interface

- All HTML driven, even when appearing like DOS
- DOS style console screen panels
- Icons can be tagged to any panel
- Any panel can be full graphics or character, or both
- Panels can be tiled, stacked, minimized, hidden or full size single
- Icon based active panel icon button task switch bar
- Every application and associated panel(s) can have individual or different icons
- Full color with scaled fonts
- Sub panels or frames, all forms of menus and button bars
- Reduced HTML for character only mode
- All manner of HTML supported formatting as frames, tables, lines, wallpaper, etc.

- Multiple concurrent video cards and display formats
- Separate screens and panels or a virtual large screen
- Drag-n-Drop across displays
- Real time loadable/unloadable drivers

Storage System Format

- CP/M or UNIX style non FAT directory
- Block mode - no sectors or tracks
- Inverse tree sub-directories
- Long file names with extensions
- Textual file descriptions for display during directory functions
- Various accepted file flags such as read-only, system, hidden, etc.
- Directory entries display in color code format as to type
- Volume names not drive letters

Storage Devices / Sub-systems

- Real time loadable/unloadable drivers
- Mount/unmount capable
- Interactive setup and control of all devices
- All interface types
- All formats
- All device types
- Single or multiple drives, single use or arrayed

Input / Output Devices

- Real time loadable/unloadable drivers
- Any that can be plugged into standard interface ports
- Any buss pluggable device
- It's Forth, so, anything

Application Models

- Forth whether native or interpreted, threaded or not, tokenized or not, etc.
- Any language of Intel native code form when compiled with OS/Forth libraries
- 'C', Pascal or Assembler when compiled with OS/Forth libraries

Application Internal Interface

- Interrupts and calls
- Stack and register parameter passing

- Data pointer passing to buffers, control records or table entries
- System, application and private vocabularies
- Core file inter-process communications

Device Drivers

- For all popular and available devices
- Real time loadable/unloadable
- Table driven
- Monitorable and displayable
- Multiple per device
- Open firmware compliant

Printer Drivers

- Fostering of HTML driven printers
- Postscript

Smart I/O

- Embedded controller support
- Real time loadable/unloadable drivers
- Active monitor and control
- Open firmware compliant

Software Included

- Forth interpreter and compiler
- Interactive help system
- Tele-comm/net utility
- System activity, operation and exception logger
- Device driver monitor
- All manner of disk and subsystem utilities
- Editor - of basic function

What we have here is the best of all worlds. The chart below shows where different feature came from in concept more than actuality.

Forth	Need I actually say.....
HTML execution	See previous on Forth
Embedded controllers	Burroughs DLPs
Textual display	All consoles that emulated 80 column punch cards
Graphical display	Xerox Star and DOS graphics programs
Multiple display	We all wanted multiple displays per PC, EGA spec.
Inverted tree directory	UNIX
Long file names	UNIX
Real time device drivers	Netware 3.x and up
Display concept	Multiple page serial terminals and need
Block storage	BusLogic SCSI controller programming interface

A Mention of PC99

If you go to the web site of either Microsoft or Intel you will find in the developers section a .PDF document, locked to all editing functions, that details the design of the next versions of desktop PCs' to be offered to the public in the 1999 time frame. I guess the reason for the PC99 nomenclature. The information shows how PCs are to be built so they will work correctly with *Microsoft's* operating system offerings, or you are non-compatible. This means that *Microsoft* is even now dictating the direction that hardware is going to take, becoming more integrated with *their* operating system, and locking us into *their* products. The document states that the new specification does not address the likes of file and network servers - bunk, most servers are desktop systems with more expensive parts inside. So where are we, just corralled, feet lashed together, butt branded or in the feed lot looking at a butchers truck? I really hate this sneaky shit. We're not that dumb.

I see a life of misery when the largest manufacturer of operating systems is dictating the design of future hardware offerings. I think we can do something about this. And so comes OS/Forth, the Cowboy of the Operating System world.

The Payoff

The thing of most note here is - **our effort is not controlled by the perversionarys of the stock market or financial communities**.

I feel it a shame that so many CEOs and Chair of firms today allow those that are interested in only short term financial gain to manipulate their perspective of reality. What stocks do means nothing when solid sales are present in a growing market and the firm can respond with appropriate output and support. Laying off personnel just to make the quarterly earnings look better based on sales to expense ratios, or other tripe, is ludicrous. If you layoff personnel and the buildings burn down what do you have - ashes, but, if the personnel are there, they lend their shoulder and build it back up, better than before, as they are the only true asset. Listen to and rely on them.

And too this, the following -

> As an interesting story goes ... back about ten years or so, a little Silicon Valley founding firm called Varian Associates, started in a garage by the Varian brothers in the '30s, had a C.E.O. named Norm Parker. I admire this man to this day. During his leadership Varian offered up to 15% per year increase to employees that performed at the 'Excellence' level, there were always intern jobs for college students every summer in all company areas, there were no layoffs, the stocks paid every year, the company was adding new divisions. He followed a simple rule - if the division was not profitable, contract out their abilities to other firms, even if only at cost. People stay employed, the stocks pay, and the firm and its product base grow. Ten years ago there were people having forty year anniversaries with the firm. Then there was an executive *coup de grace* by some juniors, the head being the current or perhaps just past chair, Tom Sege. On his way out this honorable man, Mr. Parker, gave an across the board raise of 4% to all non executive employees, saying "This is the last thing I can do for you." In less than a year Sege and his henchmen were bragging about the purchase of another firm while closing down divisions and laying off people - oh, but the stocks went up. This is the executive trend and sin of today.

Being whom we are, creations can be made and presented, all the while we can not be manipulated by the financial market. That, in business terms, is real power. Based on the fact that the OS/Forth architecture is open, firms that play in the stock or financial markets can leverage products off it, fitting well within the vision, and not corrupt their standing. Everyone makes out without destruction or corruption of the vision. The best possible place to be for a new Operating System, all the possibilities and none of the limits, an international contender.

As a statement of direction FIG will need to create a foundation to own and receive the proceeds from the sale of this product. I would expect this foundation to distribute the dividends to the participants and deal with the manufacturing issues as contracts and rights. Proceeds are for grants, scholarships and research ventures. Proper representation and visibility in the industry costs. The Foundation would and will enter into and administer contracts with manufacturers for use of and distribution rights to the Operating System.

The numbers look like this: In the typical manufacturer/distributor world, given an MSRP of say $69.95, expecting a cost of medium and documentation to be under ten dollars per, the price to the distributor would be $34.97 which leaves ~ $24.97 for FIG and the development participants.

Some numbers:

MSRP	69.95
Cost of production	10.00
Vendor discount of 50%	34.97
Residual per	24.97
FIG portion of 25%	6.24
Participant portion of 75%	18.73
Conservative 1,000,000 sales	24,970,000.00
FIG portion of 25%	6,240,000.00
Participant portion of 75%	18,730,000.00
50 participants > each	374,000.00
Wild 100,000,000 sales	2,497,000,000.00
FIG portion of 25%	624,000,000.00
Participant portion of 75%	1,873,000,000.00
50 participants > each	37,400,000.00

I see the distribution of funds as 25% for FIG and 75% to be split among the participants. I estimated fifty participants, there could be many more, or less, and the numbers slide accordingly. Don't be scared of the extra zeros on the right, they are just zeros after all. If a strategic alliance situation comes into existence than these numbers must change based on the sharing of proceeds with the alliance partner doing the production or supplying the technology.

This effort will quickly gain wide visibility with consulting and development opportunities opening up for the Forth literate. All the vendors wanting drivers for hardware, new application development or software conversions will need plenty of warm bodies babbling Forth etiquette intermixed with statements of Stack-Power and mumblings of thread models. As acceptance occurs books will need written, applications imagined and instructors incited to orate. All residual opportunity.

Ownership

It is intended that the rights to this effort be held by FIG unless co-ownership is created with a firm(s) providing proprietary information for the effort.

I had started the contact process with Caldera (a Canopy Group venture) to try to secure the multi-tasker and memory manager written for Multi-User DOS, the last protected mode version of MP/M. After hearing what was intended they were very interested .. then complete silence. The code could have chopped six months off the effort with Caldera owning part of it all. So I wonder what nerve got banged when knowledge of this effort reached the executive level.

All participants will necessarily have to sign non-disclosure to protect the Kernel, the only part of OS/Forth will not be open or available in source. All else will be open. This is the only way to keep from being devalued by a slew of low or no cost me-too products.

Conclusion

We stand at a very special place in time and space, we have one of those rare opportunities to actually change the direction of computational equipment. Forth being a tool, and we the craftsmen, let us create the future.

Please - hear my orison; Do not think this project Horror, having come full circle and reached its' limits, to finally disappear up its' own sphincter.

Consider - the computer outside the context of our society has no value, is nothing.

With several million PCs in the world and thousands being sold daily one can see a potential market, a persistent one. Little mention has been made here about other platforms, present and quite capable and very much in need. Systems such as Sun, Apple, MIPS, Silicon Graphics and others, all the various mini-computers and main-frames, such a vast territory of potential computing landscape, one OS/Forth can address.

If we had all really pulled together with Tom Zimmer and F-PC imagine where it would be today, we could have competed with 'C'. I offer an apology from all of us to you Tom. You have spent much life and frustration in your effort. You took the original concept of a professional Forth development system and made it happen, gained select contributions from others such as Bob Smith, and did yourself one better with TCOM. My admiration Sir. I did not cover F-PC or other flavors of Forth because the issues are not the current implementations but the one to come. I sense that the experts in each area will feel the need to dabble, and I await your input.

Come Forth, let us break code together, cross compile falsehoods, dump all hex, trace hidden meanings, translate other tongues, decompile ancient words, and find enlightenment.

In conclusion, let me say, that Forth could soon, in essence, gasp its' last breath if a new and radical way of thinking about its' application is not accepted and applied.

Credits and Trademarks

If you see a graphic, word, phrase or product name that belongs to someone - well it's theirs.

This is group internal document and as such claims immunity from various small and/or circled letters.

I offer thanks and gratitude to you that were/are (un)knowing and/or (un)willing participants and/or supporters in this effort loosely referred to as a document.

References

Based on the fact the 'Concept of an Operating System' is quite large I have purposely not made specific references to or quotes from the data sources used. To understand everything one must be slightly more immersed in the data and associated information for the hardware specific environment. Remember this is a presentation to become a technical document, and so, please wade in.

--- HTML

Robert Mudry, *Serving the WEB* (Coriolis Group)
Mark Pesce, *VRML: Browsing and Building Cyberspace* (New Riders)
Tom Savola, *Using HTML* (Que)
Dave Taylor, *Creating Cool Web Pages with HTML* (IDG Books)

--- Processor

Intel Secrets Monthly Edition http://www.x86.org

Chipsets: The Most Important Components in a Computer System by Billy Newsom
Sizing Memory in Protected Mode by Robert Collins
Protected Mode Vitual Interrupts on the Pentium and SL-Enhanced i486 Intel Processors by Maciej W. Rozycki
Page Size Extensions on the Pentium Processor by Robert Collins
Virtual Mode Extensions on the Pentium Processor by Robert Collins
Paging Extensions for the Pentium Pro Processor by Robert Collins
Pentium Model-Specific Registers and What They Reveal by Ralf Brown
An Overview of Pentium Probe Mode by Robert Collins
The Probe Mode Control Register by Robert Collins
The LOADALL Instruction by Robert Collins
Protected Mode Basics by Robert Collins

Intel Architecture Software Development Manual 1997

Volume 1 Basic Architecture
Volume 2 Instruction Set Reference

Pentium Pro Family Developers Manual 1996

Volume 1 Specifications
Volume 2 Programmers Reference Manual
Volume 3 Operating System Writers Guide

Pentium Pro Processor BIOS Writers Guide Version 2.0 January, 1996

AMD BIOS Developers' Guide Revision C August, 1995
Elan SC300 Microcontroller Programmers Reference Manual (AMD) 1996
Elan SC300 Microcontroller Technical Reference Manual (AMD) 1997
Elan SC400 Microcontroller Programmers Reference Manual (AMD) 1997
Elan SC400 Microcontroller Technical Reference Manual (AMD) 1997

--- Processor (continued)

Microprocessors - Reference (Intel) 1990
Peripherals - Reference (Intel) 1990

--- Network

Dave Kosiur and Nancy Jones, *MACWORLD: Networking Handbook* (IDG Books Worldwide)
John Ruley, *Networking Windows NT 3.51* (Wiley)
Karanjit Siyan, *Netware: The Professional Reference* (New Riders Publishing)
Hal Stern, *Managing NFS and NIS* (O'Reilly and Associates)

--- Operating System

Stephen Coffin, *UNIX The Complete Reference* (Osborne/McGraw-Hill)
Brent Heslop and David Angell, *Master SunOS* (Sybex)
Stephen Kochan and Patrick Wood, *UNIX Shell Programming* (Hayden Books)
Mark Sobell, A Practical Guide to the UNIX System (Benjamin/Cummings)
Rebbeca Thomas and Rik Farrow, *UNIX Adminisrtation Guide for System V* (Prentice Hall)

SCO UNIX System V - System Administrators Guide (P T R Prentice Hall)

NetWare Version 2.x Reference Set (Novell)
NetWare Version 3.x Reference Set (Novell)
NetWare Version 4.x Reference Set (Novell)

Plan-9 Bell Laboratories	http://www.bell-labs.com/

Plan-9 Index Bell Laboratories	ftp://plan9.bell-labs.com/plan9/index.html
Plan-9 from Bell Labs	http://www.ecf.toronto.edu/plan9/
The Linux Documentation Project: Homepage	http://sunsite.unc.edu/LDP/
MyOS.com - The Operating System Website	http://www.MyOS.com/os.shtml

--- Software

Richard Ferraro, *Programmer's Guide to the EGA and VGA Cards* (Addison-Wesley)
Donald Gregory, *The Extended Algol Primer for Burroughs A-Series* (Greorgy Pub.) Vol.s 1-3
Douglas Hergert, *Visual Basic Programming with Windows Applications* (Bantam)
Leo Scanlon, *Assembly Language Subroutines for MS-DOS Computers* (Tab)
Robert Stevens, *Object-Oriented Graphics Programming in C++* (AP Professional)
Tom Swan, *Turbo Pascal for Windows Programming* (Bantam)
Allen Wyatt, *Using Assembly Language* (Que)

NEWP - Programming Reference (Unisys) Mark 3.7
Turbo Assembler - Users Guide, (Borland) ver. 3.2
Turbo Assembler - Reference, (Borland) ver. 3.2
Turbo Pascal - Users Guide, (Borland) ver. 7.0
Turbo Pascal - Reference, (Borland) ver. 7.0
Macro Assembler - Programmers Guide (Microsoft) ver. 5.1
Turbo 'C' - Reference Guide (Borland) ver. 3.0

Turbo Pascal Runtime Library - Source Code (Borland)
Turbo C+ Runtime Library - Source Code (Borland)
Compiled CBasic Runtime Library - Source Code (Digital Research)

--- Software (continued)

TCOM - Source Code (Tom Zimmer)
F-PC - Source Code (Tom Zimmer)
Pygmy - Source Code (Frank Sergeant)
MVP Forth - Source Code (MV Press)

--- Thought Oriented

Leo Brodie, *Thinking Forth* (Prentice-Hall)
Steve Maguire, *Debugging the Development Process* (Microsoft Press)

--- Some Stops for the Making

Stanford's Hoover Institute		http://www-hoover.stanford.edu
Engineering oriented data and links	The Engineers' Club	http://engineers.com
Protected mode programming	Intel Secrets Monthly	http://www.x86.org
Intel processor information and manuals	80x86 Index	http://www.sandpile.org
Motherboards and chipset data	Intel Chipsets	http://www.motherboards.org

--- Misc.

Intel HomePage	http://www.intel.com http://developer.intel.com http://download.intel.com ftp://ftp.intel.com
AMD HomePage	http://www.amd.com
Microsoft HomePage	http://www.microsoft.com ftp://ftp.microsoft.com

---

Contact Information

Home Base		http://www.forth.org
Skip Carter	FIG Chair	skip@taygeta.com
George Perry	SVFIG Chair	geoperry@iww.org
Christopher Passauer	The Bohemian Monk	lgicwvrs@wenet.net

and so ... it begins