Technical Briefing

The TAOS Operating System: An introduction

October 1994

Audience

This document, which has been designed to address a wide audience, gives a brief introduction to the ideas behind Taos and its benefits. We have attempted to keep the technological language to a minimum although the very nature of the subject matter makes it necessary for explanations in some of the sections to be quite technical.

Overview

Computing has been getting steadily more restrictive. The move to so called 'open' systems has not helped improve portability, has not produced credible support for parallel systems and has not stopped the further development of increasingly complex interfaces between operating systems and applications. All in all, it has done nothing to improve the lot of the developer or the consumer.

The Taos operating system addresses these issues. Its existence and success is good news for technology from small embedded systems through to supercomputers and large scale network applications.

Taos is not conventional as it has not evolved from an existing operating system. Tao Systems developed Taos taking into account the commercial and technological realities of the time whilst also removing the limitations enforced upon the user market.

Key Features

Taos, a compact, general purpose kernel for parallel systems embodies a number of vital ideas.

Hardware Independence - Taos applications run on different processor architectures without any re-compilation of programs.

Load-Balancing - Taos provides an optimum distribution of processes over the network.

Heterogeneous Processing - Parallel applications are able to execute over networks of dissimilar processors.

Dynamic Binding - Only those parts of an application which are needed at any time are loaded into memory.

Multi-Threading - A piece of code loaded into memory, is available to all programs which need it.

Parallelism - Taos uses a process-based programming model.

Object Orientation - Taos has an object-based program design model.

Asynchronous Messaging - Messaging does not halt the sending process.

Minimal Kernel - Taos has a very compact implementation, optimising performance and minimising memory requirements.

The Challenge of Parallel Systems

Why has parallel processing remained on the periphery, given the obvious potential this technique has to increase the power of computers way beyond that possible from increasing the power of a single processor?

The answer is that, whilst it is comparatively easy to build massive parallel computers, programming them has proven immensely difficult.

This has been due to a number of factors including:

- A lack of a suitable general purpose programming methodology
- Programs written to run on one machine tend to be tied to that one machine, possibly to a single configuration
- Uncertainty as to which computation and communications model to use.

Parallel - Features & Benefits

Taos is a software response to the challenge of harnessing parallel hardware. Parallel systems are complex, comprising thousands of processors with differing amounts of memory, link standards and so on. Taos exploits the power of the hardware, whilst presenting the programmer with a clean, simple and powerful programming environment.

Taos enables programmers to think about the design of their applications in terms of parallel processes. They are completely free to build any structure of processes and communications they need. It is the programmer's responsibility to think about their application and identify the appropriate decomposition into parallel processes. But once this is determined, Taos will manage the distribution of these processes on any target network.

Programmers need not concern themselves with the details of the actual hardware and network architecture upon which their applications will run, including adjustments to the network size. Taos grows over all processors available on the network, and will expand if more processors are added; whilst applications grow over the available processors as they create objects during their execution. This means that it is easier to write 'shrink-wrapped' applications for parallel computers using Taos than it is to write them for single processor operating systems, as applications code will run on any processor supported by Taos.

Taos uses a process and communications model of parallelism. Each process can execute on a separate processor, and talk by sending messages to each other. The processes are fast and can only interact by passing messages. There is deliberately no concept of shared memory, though Taos can be implemented on shared memory machines.

Taos does not attempt to evolve from a sequential model; extracting parallelism from sequential programs does not and will not work. So, Taos has been designed as a perfect fit for parallel systems. Taos software exploits parallelism; creating parallel applications using Taos is easy. By creating objects and passing messages, the programmer generates the opportunity for Taos to distribute the objects over processors and thus generate parallelism.

The final important point to re-emphasise is that it is up to the programmer to write their programs so that the objects can execute in parallel. Parallelism is not automatically created, nor should it be. There are instances where a program must be sequential to behave correctly.

Portable - Features & Benefits

Portability is normally taken to mean that programs written in a language (such as C) to run on a particular processor, can be re-compiled to run on a different processor without the need to re-write any (or a small portion) of the code. This approach has its limitations as a new compiler needs to be written for every processor type introduced. This can consume a great deal of effort to achieve.

Taos takes a different approach. By targeting all compiled code at the Taos Virtual Processor, porting relates only to the VP itself and not to the applications, whatever language they may be written in. In effect the portability has been taken from the language level to the processor level. Furthermore, this low level portability facilitates heterogeneous processing support. See Heterogeneous Processing, below.

Taos can be ported with little effort to any processor or communications hardware. The only part of Taos that needs to be re-coded to support a new physical processor, is the Translator. The Translator is the program which converts VP code into the native code for the processor. Once the translator is written, all applications written for any processor, in any language which compiles to VP, will become instantly executable on the new processor type.

So, to reiterate; programs written on any supported platform will run on all other supported platforms without any changes; you don't even have to recompile.

Virtual Processor - Features & Benefits

The first Taos virtual processor was an imaginary 32bit microprocessor. It has already been extended to create a 64bit version and can be taken further with no limitation.

All programs are compiled or assembled into virtual processor code and are kept in this form on disk. The VP code is translated into the native code of the processor on which it is to run only when it is needed. The translation occurs as the VP code is loaded from disk, across the network and into the memory of the target processor. This mechanism is at the heart of Taos' dynamic binding facilities. See Dynamic Binding, below.

It would be wrong to think that this slows the system down. Most processors are able to translate VP into native code faster than the VP code can be loaded from disk and sent across the network, so there is no visible overhead. Indeed, VP code is often more compact than the native code; therefore less disk space is used and code is loaded faster than if it were the native for the processor.

If there are particular advantages in using a native version of the code, then this can be stored on disk and will be loaded in preference to the VP version. This would be because the performance of the code would benefit from specific instructions supported in hardware by a particular processor.

A VP version of the code will run on any processor for which a translator is available. There is no need for re-compilation. see Portability, above.

Dynamic Binding - Features & Benefits

Some readers may be familiar with dynamic linking. Taos dynamic binding is more than dynamic linking. Code units are brought into memory only when needed. This is how it works:

As a process executes it will ask for a named piece of code, a tool. This will be brought in from disk and translated into native code, before being executed. This tool may, in turn, call further tools, following the above procedure. Once a tool has finished being executed, it may be removed from memory if no other process is referencing it.

Dynamic binding implies that many processes may share code (tools), i.e. multi-threading of code. This is highly memory efficient. Once a tool has finished being executed, it stays on the list of available tools. Should it be needed again, this local copy will be used, thus avoiding a re-load from disk. Only if memory gets filled up will the tool be flushed and subsequent calls require a re-load from disk.

Taos' dynamic binding and VP code combine to enable Taos to exploit heterogeneous parallel hardware.

Heterogeneous Processing - Features & Benefits

Taos is able to run on any hardware for which there is a translator. This provides the basic facility which enables programmers to write code which can run on a wide range of hardware. This, together with Taos' dynamic binding, enables programs to run over a network of differing processor and

communications types without any changes to the code, or any re-compilation.

Heterogeneous processing is the exploitation of a range of different types of processor memory and communications hardware to create a versatile parallel computing machine.

Such machines may be built from a variety of hardware to support different kinds of computation. For example, a vision system requires a front end to do the initial work on the raw images being received, to find edges, shapes, etc. This stage of the problem is well suited to data parallel hardware, whereas the back end will need to extract meaning from the shapes, which is more suited to implementation on hardware supporting list or network processing. The ability of software to run a variety of hardware is essential for this type of machine to be successfully exploited.

Furthermore, Taos provides access to information about the hardware available in the network. So, programs can take advantage of special purpose hardware to run specific objects. This means that sophisticated adaptive programs can be written within the Taos model to get the most from heterogeneous machines.

Heterogeneous processing is the complete Open System, in which all aspects of the processing are distributed across dissimilar processors, networks and architectures. It benefits consumers who can select processors on the basis of current price:performance or whichever criteria they select as their priority, without prejudicing future decisions. For manufacturers, it gives them the flexibility to improve hardware design and not become locked into historical decisions for the sake of software compatibility.

Minimal Kernel - Features & Benefits

The key to Taos' applicability to so many areas lies in its compact kernel. This is all that is needed for a processor to provide all of the Taos services. The Taos kernel provides all the facilities needed to support its simple execution model, yet the whole kernel is only 12K.

The kernel executes on every processor in the network from boot, and its facilities include memory management and caching, object creation, distribution and execution, object message passing and tool calling. There are also calls to provide global name management, local timer and scheduling, and access to network hardware information.

The programmer is free to add objects to the system as appropriate. Examples of these may be hardware drivers or file system objects.

Objects and Messages - Features & Benefits

Taos objects are also parallel processes. Each object which Taos creates is given its own process to execute it. The difference between objects and processes lies what they are about; objects are about data and code, they occupy memory space; processes are about processors, they consume processor time. Put the two together and you have an object which is executing, consuming memory space and processor time. A process brings an object to life.

Objects can only interact with other objects by sending and receiving messages. As objects can exist on separate processors, memory can not be shared between objects. Taos provides a light-weight-mail system to communicate between processors.

Messages are just like letters, you send them by posting them in a mail box with an address on it. You use letters to communicate with people to whom you cannot talk directly. So it is also with Taos objects; you use messages to communicate with other objects, as you cannot talk to them directly since they can be on separate processors.

When you send a message you specify the mail address of the recipient. The sending of mail is equivalent to popping a letter into a post-box. Conversely, each process has a mail-box in which mail is placed by the messaging system. Objects receive mail by checking their mail-boxes. Messages are typed, so the object can distinguish between a variety of possible incoming message types.

Messages conform to the basic node format, plus extensions, to hold the sender's address and the message's destination address. When mail arrives it is left on a list for the receiving process look at.

Taos provides a simple mechanism to send messages, there being only two facilities in its messaging system, to send and receive messages. Messages are sent synchronously; once a message has been sent on its way the sending process is free to continue its execution. Synchronous

communication can be achieved by waiting for a returned acknowledge message. Taos' method of communication ensures that parallel processes execute independently of one another for as much of the time as possible. An object may need to wait for a message before it continues processing, but a sending object will not wait until the destination has received it.

The mailing system uses a distributed algorithm which finds multiple paths to a destination and may use more than one route for each message sent, thus making best use of all available communication paths. If one route is bottlenecked then the message will get through via another.

Distributed Processing - Features & Benefits

The whole conceptual approach to the design, implementation and execution of Taos is organic. A program evolves to fill the network during runtime. Other systems introduce bottlenecks by requiring a system-wide time-stamp on all messages, or by maintaining a central control over the distribution algorithm.

Taos does not attempt to impose a central control over the execution of an application. The kernel is small enough to exist on every processor, providing local services to the objects on its processor and interacting with adjacent processors' kernels to provide message passing and process distribution.

Distribution is based on processes which pass messages. Load balancing (the distribution of processes over the network to maximise the performance of the system) is achieved via a simple algorithm based on the computation and communication requirements of the objects. When an object is created, the loading on the local processor is compared with the loading on neighbouring processors and the object is allocated to the most suitable processor.

Each processor holds minimal routing information for each communications channel to enable messages to be forwarded towards their destinations. Messages are routed to their destinations in much the same way as water flows down through pipes under the effect of gravity.

During the execution of a program, processes distribute themselves over the available processors as they are created. The net effect is that objects spread out over the available processors in much the same way as a liquid spreads out over a surface.

Object Orientated Design - Features & Benefits

Taos was conceived to take advantage of the reusability and robustness provided by an object based approach to design. Object ideas are exploited at all levels of Taos, from kernel data structures to high level classes. Kernel objects are used to build higher level messaging passing objects.

The basic data node structure used by Taos is inherited by all objects in the system. This enables Taos to manipulate all entities which conform to this very simple structure. Message passing objects, the messages which they pass to each other and the tools which they use to process their data all conform to this basic data structure. New types of object may be introduced by the programmer.

Data nodes are the basic object in the system.

Tool objects are bits of code, like formal object orientated programming's methods, but without any restrictions on their use.

Control objects, as they are known, are objects which have a process associated with them. They communicate via messages and can be distributed over processors. Such a message passing object will typically contain several components which are references to tool objects. See Objects and Messages above.

Classes provide higher level functionality such as the Window and PolygonWorld. These are formed of message passing objects bound together to form the class object. A class may make use of many objects working in parallel. These objects are made available to the user in the form of calls to the class to, for example, create new windows and manipulate them using method calls. So the user just sees the functionality such as 'open window' and does not needed to be concerned with underlying parallelism generated by the execution of the objects in the class.

Programmers are encouraged to use existing messages, tools, objects, and classes, to create their own new ones. There are presently over 3000 tools covering a wide range of basic functionality from string and file handing to classes supporting 3D polygon worlds. Re-use and relax! A program to fly-through a fractal landscape is a under 100 lines

long, when written using the existing objects.

Object and Memory Management - Features & Benefits

Taos executes objects and manages memory in a very consistant manner.

The Taos software model uses processes, messages and objects, whilst its hardware model uses processors with local memory and communication channels between processors. The way to view objects is that they consume memory space, whereas processes consume processor time. An object needs a process to enable it to execute.

Upon creation of an object Taos allocates the object to a processor and then allocates a process to execute the object. A typical Taos object is a few hundred bytes in size. So, they lie between fine-grain Smalltalk-style objects and course-grain UNIX-style objects.

The lowest level object in Taos is the 'node'. This is the simplest entity with which Taos deals. It is a variable sized packet of data which can be placed in a doubly linked list. All Taos entities conform to this basic format. From this basic building block, other structures have been grown, such as tools, messages and other types of system object.

Nodes have a type field which identifies what type of Taos object the node holds and hence how it should be processed. Pre-defined types include: Tools, Control Objects, Bitmaps, Graphical Objects and Class Objects. The user can define new types.

Nodes are held in one of two forms. When the node is on disk or being communicated across a network, it is held in 'template' form, as it is loaded into memory and made ready to be executed it is converted to 'process-ready' form. As the template is converted to process-ready, any translation from VP code to the local processor's native code is performed, and the node is inserted into a list of other process-ready objects (see Dynamic Binding and Portability). Once a node is in a process-ready list it can be processed. The node type determines how it is to be processed. Two types to focus on are the Control object and the Tool object.

When a Control object is created the object's template is distributed and made process ready on a processor. A process is made available to the object and it starts to execute. A Control object is made up of one or more components, which are all Taos nodes of one type or another. Each component is executed in sequence until the last one is finished when the Control object closes and its process finishes. The components may be other Control objects, tools, graphic objects etc. Tools are bits of code which operate on the data defined in a Control object. For example, they may perform calculations and send and receive messages to and from other Control objects. All Control objects are created by another Control object, and each has the mail address of its parent, forming a tree. Tools, being nodes, can be manipulated by the kernel. A Control object may consist of some local memory space and some constituent tools which operate on the data.

Whilst the Control object is the smallest entity which can execute in parallel, it is not the finest granularity of memory management. Individual tools can be loaded from disk, as they are also Taos objects (conforming to the basic node format). A Control object template only holds the text names of its constituent components, not the actual code. As a Control object is created, the kernel checks to see if the tools which the components reference are already available in memory, and if they are, simply points to them. Only if an object is not present will it be loaded from disk and be made process-ready. So all Taos objects can be multi-threaded. You will never have two copies of the same object in the same memory space, unless you specifically request it (See Dynamic Binding).

Another feature of this execution mechanism is that only those components which are needed are ever loaded. If you design your application so that it is built of hierarchically structured Control objects, then code will only ever be loaded if it is executed. If the path of execution does not pass the particular component then it will not be loaded and thus occupy no space. So, the amount of memory consumed is kept to an absolute minimum and is driven by the execution of the program.

Conclusion

Taos is ground breaking. The push of new technologies and of parallel processing in particular has forced a profound re-think of what an operating system should be. Taos provides what is needed. Elegant, compact and versatile, it provides the programmer with

simple yet powerful tools to exploit emerging technologies.

For those carrying the torch of Open Computing, Taos provides the ultimate open platform through distributed processes across dissimilar processors to achieve Heterogeneous Processing. This is achieved by having totally Portable Code. Taos-based applications are written only once, so that software houses can now channel funding into the development of new products rather than having to allocate vast sums towards the porting of existing packages from one platform to another.

Taos' Parallel Processing facilities generate a Masterless Network with no practical limit on its size and providing Linear Scalability of performance. Developing products for a parallel environment has traditionally been a major stumbling block. But writing parallel programs for Taos is as easy as writing programs for a single processor environment; and once a program is written it will run on any processor supported by Taos without any changes. The Load Balancing techniques employed by Taos enable applications to exploit additional processing power as it is added, without re-compilation, even during the execution of a program.

Taos' Object Orientated programming techniques have led to the creation of thousands of reusable tools which will be used over and over again in future software developments. Other Object Orientated techniques have so far failed to live up to expectations, but Taos shows that this methodology can, if employed wisely, yield massive benefits to the programmer and end user.

Taos' lack of protocol layers makes it very reactive to stimuli and this combined with its highly efficient Dynamic Binding, provides the basis for truly Real Time systems.

Taos' Scalability enables it to underpin massive superscalable networks, whilst its Compactness makes it an obvious choice for embedded applications.

This only provides a brief overview of just some of the features and benefits of Taos; despite this document's limitations, what we hope it does emphasise is the remarkable flexibility of Taos and the broad range of markets for which it is ideally suited.

OCTOBER 1994

APP 764