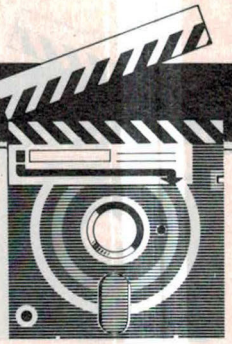


SCREENTEST

Tripos

Philip Sargent presents the history and capabilities of Tripos, Metacomco's British multi-tasking operating system which drives the Commodore Amiga.

Most people have heard of the Tripos operating system only recently as a consequence of AmigaDos, a version of Tripos written by Metacomco and released with the Commodore Amiga. Before Metacomco became involved, the Amiga development team had written their own message-passing and storage allocation 'kernel', and approached Metacomco only when the rest of their operating system development was behind schedule.

That Amiga kernel (called the Exec) had been written by people who had clearly read a few books on operating system design, so the job had been done properly and cleanly. As Tripos is the result of research projects in operating system design, the extremely close functional similarity between the Amiga Exec and the usual Tripos kernel is not entirely an accident. AmigaDos is thus the Exec kernel and Amiga device drivers (in assembler), plus the usual Tripos task handlers and file handlers written in BCPL.

AmigaDos differs from Tripos in having a completely rewritten kernel, but as far as the user is concerned, the differences are slight: AmigaDos time-slices tasks of the same priority, whereas Tripos does not permit different tasks to have the same priority. AmigaDos has 256 priority levels and Tripos has 65536, due to the fact that the original Tripos really only understands 16-bit words (as does BCPL), whereas the Exec has been written in terms of 8-bit bytes. Some recent versions of Tripos are more byte-oriented and are less closely integrated with BCPL.

The Amiga Intuition WIMP graphical user interface and the graphics support library reside in the same ROM in the Amiga as the Exec kernel, but are entirely independent of Tripos.

History

Tripos is a single-user, multi-tasking operating system for small computers, which originated in the University of Cambridge Computing Laboratory as part of a program of research into portable operating systems.

The name, Tripos, is widely thought to derive from Cambridge students' final year exams, which, in turn, are named after a (probably mythical) three-legged stool. In addition, both its detractors and its critics have sometimes referred to it as a *TRI*vial *P*ortable *O*perating *S*ystem: it lacks some of the features that are found in other operating systems, but its simplicity is closely related to its portability and is considered by many to be a great virtue.

What is a multi-tasking system? Imagine every program that you work with acting like Sidekick: immediately available at any time, without terminating your current task. Secondly, imagine that any of those programs can be working away in the background: compiling, answering the telephone, walking the dog, and so on. That's a multi-tasking system.

Operating systems other than Tripos have been implemented on a larger number of machines, and on a larger number of different types of machines; but very few other operating systems are so *easy* to implement. A recent port from one 68000 machine to another required a change of *only two lines* in the assembler kernel. Tripos has been implemented for the following machines: Computer Automation LSI–4, Digital PDP–11, Data General Nova, IBM Series 1, Intel 8086, and a large number of 68000 and 68010 machines including the Stride, the Amiga and the Microbox III.

Tripos originated in 1976 with Martin Richards in the University of Cambridge Computer Laboratory, and was devised for single-user minicomputers of that period. With the help of a number of research students, it was demonstrably up and running by February 1979. Unix predates it, but Tripos was one of the earliest operating systems written in a high-level language to have been specifically designed with portability in mind (Unix wasn't), and the only one to have gone beyond the experimental stage. It was designed to be portable, fast and small.

Tripos is used at Cambridge University for research, and former Computer Laboratory members have taken it with them to several companies where it is currently being used for software development; companies which include Micro Concepts, Topexpress, and Metacomco.

Modularity & portability

Tripos is an extremely modular system. It consists of only a 1500-line kernel and device drivers (for keyboard, screen, serial port, and so on) written in assembler, and four utility programs written in BCPL. The assembler is, as a matter of policy, always a manufacturer's standard assembler. A BCPL compiler is required; this is almost invariably also implemented in BCPL, and produces an intermediate code which requires only a simple interpreter. This is important — a portable operating system is *not* portable unless its base language is too.

The four cooperating utility tasks

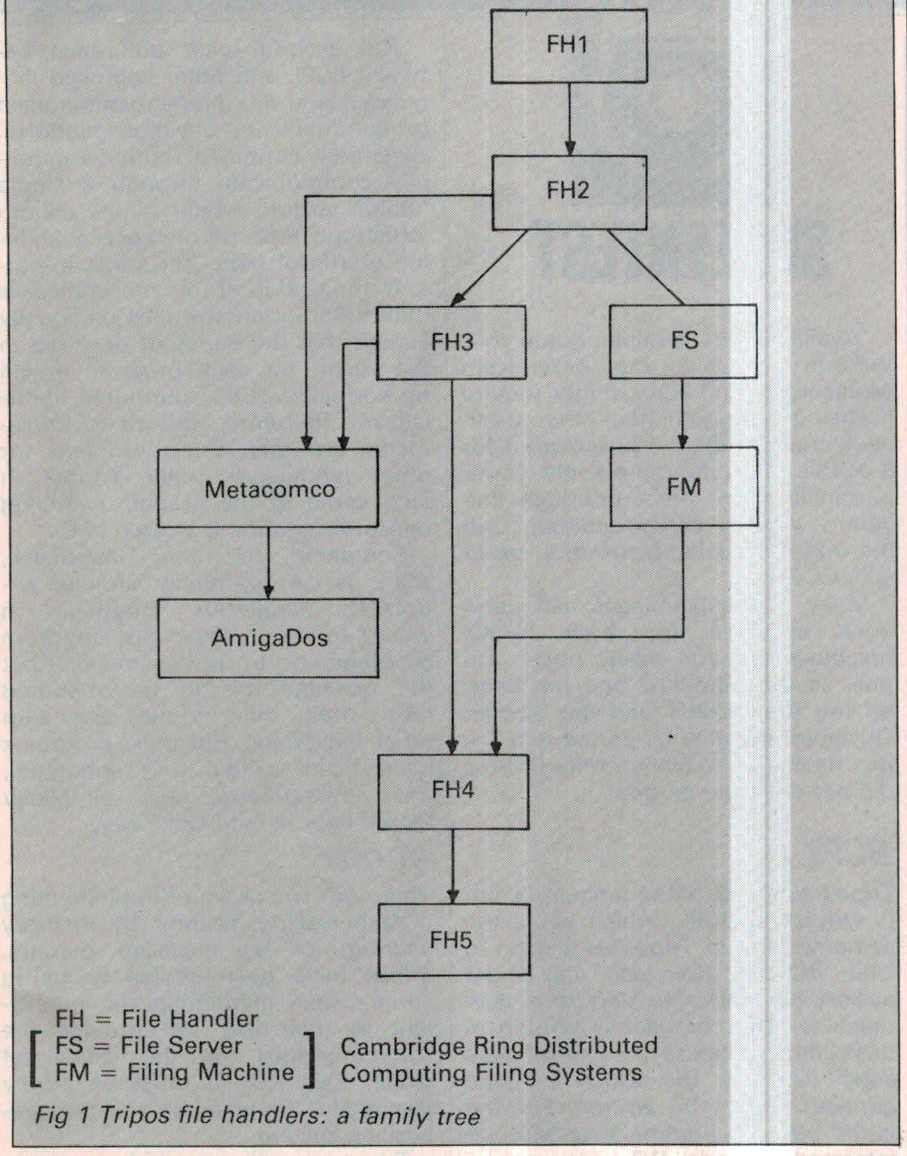


Fig 1 Tripos file handlers: a family tree

are:
1) a command line interpreter (cli);
2) an interactive debugger;
3) a console handler; and
4) a file handler

which behave as if they run concurrently. As the machine has only one processor, they are really only 'pseudo-concurrent', but the task-switching is built into the kernel and can do it in about 200 microseconds.

This concurrency must be distinguished from Digital Research's 'non pre-emptive multi-tasking' GEM-DOS (GEM on 68000 machines) where no task can be interrupted until it has finished; unlike Tripos tasks, they won't 'let go' or 'shut up'. This means that GEM-DOS operates like a task-switching system instead of a multi-tasking system: nothing happens in the background, so pauses while a work processor task is being typed in are not used for anything useful. True multi-tasking packages can be written to run under GEM-DOS if all the participating programs are written to use a sequence of extremely small tasks, but task-switching would slow things down. This can be contrasted with Tripos, where the operating system handles the multi-tasking, and any program can be written as if it had full control of the machine.

The file handler is the task which manages the filing system, and as it is a self-contained program, a variety of filing systems (with different disk formats) can be accommodated. If more than one such handler is loaded, then these filing systems can be used simultaneously. This is an even more flexible approach than that which Acorn has pursued with the BBC Micro, where the original Acorn DFS (Disk Filing System) and the Advanced System (ADFS) can be used.

The modularity of Tripos has led to confusion in some published articles, where the properties of a particular filing system have been assumed to be inherent to Tripos itself. The Metacomco AmigaDos filing system, with track cacheing and forward and backward block chaining, is more sophisticated than the 'standard' Cambridge Tripos disk system where blocks just point back to the file header (as well as forward to the next, of course). Recent research projects in distributed operating systems at Cambridge have produced FS-Tripos and FM-Tripos (File Server and Filing Machine respectively), where the file handler may not reside in the computer at all, but in another computer somewhere else, which may or may not be anywhere near the hard disk pack. Such is the flexibility of the basic system.

The Tripos kernel and device drivers form a basis for embedded operating systems for specific items of apparatus or process control, by adding suitable interface handlers. Such Tripos-based systems have been used for a wire-wrapping machine and for a variety of dedicated network file servers. It is often argued that portable programs are good programs for reasons other than their portability, and this seems to hold for Tripos, too.

Files and file size

Whatever the filing system, Tripos implementations nearly always provide a hierarchical system with directories and sub-directories, as used by Unix, Acorn ADFS and Microsoft MS-DOS.

As it has been designed as a single-user operating system, Tripos has no protection against intentional destruction, but there are some safeguards against accidental deletion. Files and directories can be labelled with Read, Write and Delete access, so valuable files can be set so that the access must be changed before they can be altered.

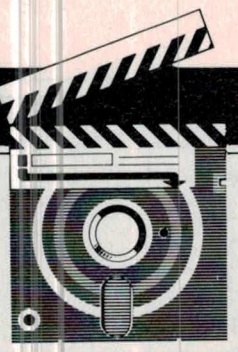
Files are assumed to be an unstructured sequence of bytes, as in Unix. There are no filemarks at the end of files; the length is written, together with the name of the creator and the date, in the file header. Fileblocks can be read by random access within each file.

As different filing systems can be mounted, the restrictions on maximum size of file or maximum size of any one disk drive, are implementation dependent. There are two limitations: the number of bits used to encode the length of a file in bytes; and the number of bits used to store the disk key, which is a fileblock address. If the disk dey is 16 bits, then there can only be a total of 64k blocks on each drive; and if each block is 512 bytes, this translates as a maximum drive size of 32Mbytes which is the same as MS-DOS. The length of the file is usually stored at 32 bits, which means that the maximum file size is 128 times the maximum drive size!

AmigaDos permits files to be split over several disks, but probably not over several drives. It all depends on how the file handler is written.

Simultaneous access

Tripos locks files against simul-



SCREENTEST

taneous access by means of a token-passing mechanism. Locks may be shared or exclusive, so a local area network of Tripos machines with a file server is a suitable vehicle for multi-user database or document processing applications, or developing any multi-user network application. Protection has to be built-in at the application level (OSI level seven), however, and is not provided by the operating system.

All current versions of Tripos implement locks on directories and on files, but not smaller-scale locking on blocks which is really required for safe database update applications, although the locks are perfectly adequate for shared-document access. This omission would be a disaster on any other non-modular operating system, but with Tripos, suitable file handlers can be built to almost any specification of that type.

A family tree of Cambridge Tripos file handlers is shown in Fig 1. FS and FM are the file server and filing machine versions for use with the Cambridge Ring, and FH4 is a local disk version which inherits their extra facilities (such as filename aliases).

The latest Cambridge version, FH5, is the one with the 32Mbyte drive size limitation because a 16-bit key saves disk space (FH1, 2, 3 and 4 use a 32-bit disk key and have a drive size limitation of 4Gbytes). Metacomco's several versions of Tripos for the Stride and other machines, and AmigaDos, derive from FH3 and FH2, and follow an entirely separate, parallel line of development.

The 'standard' Tripos filing system is called FH3 and is shown in Fig 2. The more recent FH5 (used by Tripos III and developed by Micro Concepts) uses a slot/key elaboration which makes simple directory accesses much faster. Eleven file entries (slots) fit into each key fileblock, so a simple listing of filenames could be about 10 times faster. The file headers only need to be read if the size, time stamp or the name of the file owner is required.

Software

As Tripos has been used for so long in a research environment, there is a great deal of software available which has been designed for writing different versions of Tripos, but this is not what most people want. For 'generic Tripos' on 68000 machines, there are compilers 'available' for BCPL, Algol 68C, C, Pascal, Fortran 77, Cambridge Lisp, Prolog (NSW) and Ponder, and also an assembler and a disassembler. (Ponder is the result of Alvey-sponsored research into functional languages, and it contains a rather unusual mouse-driven spreadsheet.

'Availability' is variable. Some software is for sale by mail order from Metacomco, and some might require licensing arrangements; other packages could probably be procured for a couple of p ints in the Eagle. Other possibilities are the Cambridge line editor, various screen editors, and the text processing/typesetting package, GCAL.

Many of the languages and packages available for basic 68000 machines can run easily under Tripos, so the Modula-2 and the Basic on the Atari 520ST and the Sinclair QL could probably be ported to a Tripos machine relatively easily (if they did not originate on one).

BCPL

Tripos supports many languages but is written in BCPL, which bears the same relation to Tripos as C does to Unix. BCPL is, after Lisp, one of the easiest languages to port to a new machine. The compiler is written in BCPL and produces an intermediate code (O-code) for which a code generator must be written. For the initial port, a different, even simpler intermediate code, INT-code, can be produced by the compiler; this also requires a code generator. By comparison, C (and hence Unix) is much harder to bring up on a new machine. BCPL also benefits from its limited circulation in that there are very few different versions of the language, so although it has no ISO standard, it is much more portable than Pascal, which has.

BCPL is commonly described as an Algol-like language, which gives some idea of its age. C is a development of BCPL, and nearly all the differences between the two languages can be traced to the simple fact that C can access bits, bytes, words and longwords, whereas to BCPL, the world consists only of an expanse of uniform-length words (of at least 16 bits, but otherwise of any length) which are addressed as words, not bytes. BCPL, like C, has no strict 'typing' — that is, variables can be assigned to characters, reals, Booleans, and so on, and then to another type without restriction. Unlike C, BCPL makes no distinction whatsoever: types simply don't exist — there is only 'the word'.

The second main difference between BCPL and other compiled languages is in the data structures used to communicate between modules. Separately-compiled modules generally communicate through a single 'global vector', which is just an unstructured area of memory containing words of data. The most important implication of this mechanism is that the programmer must personally ensure that the variables declared in the vector for each program match up correctly. BCPL subroutine libraries are, therefore, difficult to implement, but this is no problem for other languages under Tripos; in fact, some of the graphics-handling part of AmigaDos is written in C.

Compared to other languages, BCPL is an extremely simple, extremely dangerous language in which to write programs. Program development in BCPL implies that the machine has to be re-booted more often than is the case with other languages. But once programs have been written and debugged, their transparency and simplicity makes later maintenance easy.

In use

There are two ways of implementing a multi-tasking system: by memory sharing; or by message passing. These have been shown to be in some sense mathematically equivalent in that they can provide the same facilities, so the choice of which type to use must be purely pragmatic. Tripos is a message-passing system.

Tripos permits any task to create new tasks which then continue to run concurrently. Each new task is set up with:

— an identifying integer (for example, 5);
— a run-time stack;
— a global vector;
— an empty list of messages (the work queue);
— the task status (waiting, running, held, and so on); and
— a priority (an integer).

The priority can be changed after creation, but this is slightly expensive and not usually useful. Tasks in AmigaDos with the same priority are time-sliced by the kernel; in other systems, tasks are defined always to have different priorities.

Each task is allocated two areas of memory — a global vector and its run-time stack — and is told about certain system structure addresses, and can 'see' its queue of incoming messages. It usually does not know, and need not know, anything about other tasks in the system, but as all Tripos programs have access to the whole memory map, a pathological task can corrupt others. A task can

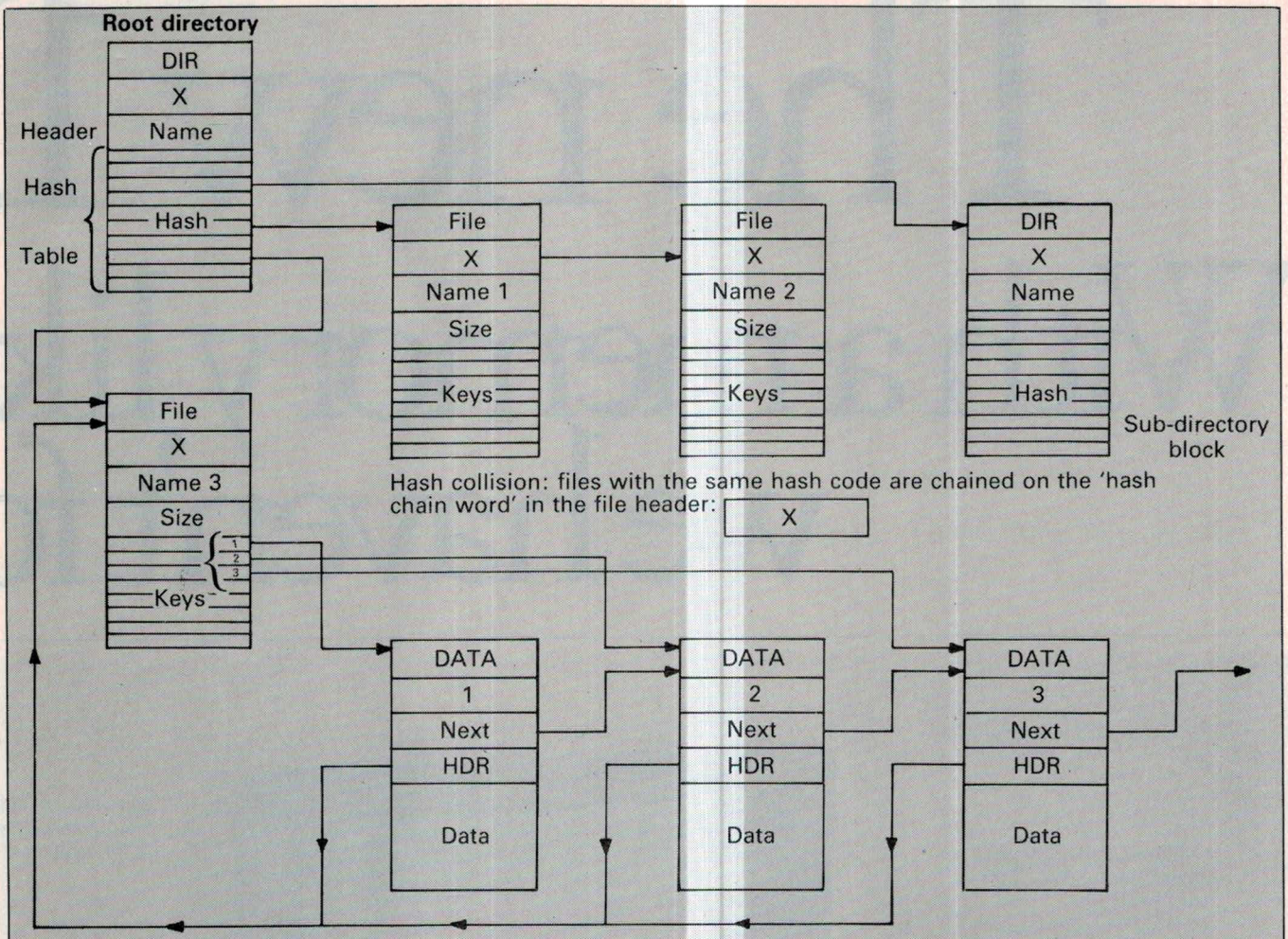


Fig 2 A 'standard' Tripos file handler: FH3

safely request more memory for its stack, and de-allocate memory when it has finished with it by using routines in the kernel. At the end of a task, all memory is de-allocated.

Within each task there can be a number of co-routines which bear the same relationship to tasks as tasks do to the operating system. They are pieces of code which do not hand over control to another routine within the task until they decide to, but as they are just parts of a task, the whole task can be swapped in and out without the co-routines 'realising'. A co-routine is allocated its own run-time stack, but shares the global vector of its parent task. Co-routine switching is approximately 10 times faster than task-switching, and co-routines offer most of the advantages of tasks but without synchronisation and time penalties. They are widely used in the Tripos system software.

The command line interpreter (CLI) is used to interpret strings typed at the keyboard (or sent over a network) as commands to run utility programs. The utilities supplied with Tripos have exactly the same status as user-written programs, and are run as co-routines of the CLI task. One of these utilities is the Run command, which creates a new CLI task and takes the rest of the input line as a program to run in that task. For example, 'Run BCPL from t-s to t' creates a background task which goes away and compiles the BCPL source file, t-s. When it finishes the compilation, the task commits suicide. If automatic suicide is not wanted, typing 'NEWCLI' creates a permanent, new CLI task which writes to the screen and reads from the keyboard. Keyboard input is directed towards the intended task by means of a typed escape sequence, or, with Intuition and AmigaDos, by moving the mouse to the required CLI window.

One of the ways in which the system can achieve efficiency in a small system is by running all tasks in the same address space, so when running, it can take up less space than it requires for disk storage (which is about 68k for Tripos III). In any case, common pieces of code, such as the entire run-time library, are shared between tasks.

Comparison with Unix

Tripos is not a virtual memory system, whereas Unix is. On a Unix machine, programs 'see' an expanse of RAM which is equal in size to the maximum address space, rather than the real RAM size, because the operating system swaps programs and data in and out between disk and RAM 'behind the back' of any executing program. The most obvious consequence of this to the user is that on a machine of similar size, Tripos's response to input is much faster.

At a slightly more sophisticated level, because Unix files have a kind of ghostly existence, it is possible to 'pipe' files from the output stream of one program into the input stream of another without pausing for breath. MS-DOS also has pipes, but they are really hidden temporary files; all the output from one program has to finish before the next can begin (unlike Unix, where it happens on the fly).

Tripos does not have virtual file handling, but since 1981 Tripos has

SCREENTEST

included a pipe handler utility which sets up a concurrent task to pass data like a Unix pipe. More recently, this has been reimplemented in Tripos III so that the pipe task can be treated almost exactly like any other file, and so provides the same level of utility as does Unix.

Tripos and Unix have a kernel written in assembler. The Unix kernel is much smaller than the Tripos kernel, but a greater proportion of Unix is written in C than the proportion of Tripos written in BCPL. The speed of operation of Unix is much more dependent on the quality of the C compiler than the speed of Tripos depends on the BCPL compiler. Thus, even discounting the virtual memory overhead, Tripos is smaller and faster. With Unix, the hand-coding effort in porting goes into the C compiler, but with Tripos it goes directly into the kernel.

Comparison with Panos

Panos is another new, small operating system written in a high-level language. It is supplied by Acorn for its Cambridge Workstation series, which is based on NatSemi 32000 processors. Panos (the name is reputed to come from a Greek restaurant) is extremely recent. It was first seen by beta-testers of the Acorn 256k NS32016 co-processor in about March 1985, and a debugged version (1.1) was released in October of that year. It is nearly nine years younger than Tripos.

Panos is written in Modula-2+. The '+' denotes a version of Modula-2 which has been enhanced by the DEC Palo Alto Research Centre and Acorn to include handling exceptions, finalisation, garbage collection, controlling concurrency, and providing a type-safe discipline for managing storage. It is apparently not (currently) for sale, but is probably quite portable. Modula-2 has great advantages for the writing of most system software, as it has strict type checking and yet allows easy access to the fundamentals of the machine (by means of the type 'word', for example). Type checking between tasks is not possible in Tripos because messages are passed only by a pointer to an address, so there would be no great advantage in re-writing Tripos in Modula-2 (or C), even without the consideration that BCPL is the more portable language.

Panos is not a multi-tasking system, but it inherits a great many other features from Tripos. This is a reflection of the fact that for a number of things fundamental to small operating systems, Tripos has 'got it right'. The single-tasking nature of Panos is a reflection of its origins as an operating system running on a second processor attached to a 6502 host processor via the Acorn Tube interface. Apparently, the requirement that the new system be upwards-compatible with the host processor hardware mitigated against a message-passing system.

The command line interpreter (CLI) utility in Panos uses almost exactly the same keyword handling mechanisms to pass parameters to applications programs as does Tripos. Similarly, there are special keywords such as 'help' which do not run the utility, but tell you what parameters you should have given. Parameters are set-up with options such as: /A, meaning a compulsory parameter — a value alone may be given and is identified by position; /K, meaning a keyword must be quoted; and /S, meaning a switch, no value required, only the presence or absence of the parameter (for example, list or nolist).

Panos implements the same ideas as Tripos in that the system utilities have exactly the same status as user-written programs. As with Tripos, users can rewrite the directory listing utility, or a Type or List command, or anything really, with impunity. Panos also permits any program to initiate the execution of any other program, thereby supplying some of the same facilities to the user that multi-tasking does for Tripos, but no background tasks are possible.

Within Tripos, it is not usually (or safely) possible to call any program from within any other; use is made of co-routines and tasks instead. There are no fundamental reasons why such a system (essentially, just enabling the execution of NEWCLI) could not be implemented with Tripos, but it would add little to the facilities currently available.

The Cambridge Ring

It would be impossible to describe Tripos without a mention of the parallel development which has shaped its growth — the Cambridge Ring. This is a slotted-ring (*not* token-ring) local area network (LAN) with continuously circulating packets. When a station wishes to communicte it waits for a packet, and writes its message, and the address of the recipient, in the appropriate fields.

There are approximately three packets at any one time on a 600m ring, travelling at some appreciable fraction of the speed of light and passing through shift registers at each station. The original ring ran at 10Mbit/s; the Cambridge Fast Ring (CFR) will run at 50Mbit/s, and slotted rings of that type are probably the only local area network which can guarantee LAN access for a station at intervals short enough for it to be used to carry speech.

Tripos and the Ring enable computers to be attached to a LAN with no peripherals at all — no keyboard, no screen, no disks. A user with a simple terminal and a ring station can attach to any free computer and use it as if it were sitting on the desk. Tripos's fast handling of messages and tasks, its assembler-written kernel, and its modular nature, are all intimately connected with this distributed computing project. Nearly all the servers on the Ring (plotters, fileservers, ring bridges, and so on) also run with a Tripos kernel and a different set of utilities.

Recent developments

The most significant recent development is the release of the Amiga, running Metacomco's Tripos (AmigaDos) with a new filing system, a kernel and a graphical user interface (Intuition) which permits separate windows for concurrent tasks. More fundamentally but less visibly, Micro Concepts' Tripos III includes two new basic utilities to add to the usual four: an Object Manager; and a Volume Manager.

The Tripos III Object manager automatically keeps the most recently-used commands and utilities in any spare RAM so that if used again, they will run immediately. In MS-DOS terms, all programs become resident as soon as they are used, but are 'ducked out' (invisibly) if the space is required. This overcomes part of the speed disadvantage of a disk-based as opposed to a ROM-based copy of the system and utilities. The Object Manager is a task, but the utilities it swaps in and out are not tasks as they are not running anything.

The Volume Manager can be used with any mass storage system with removable media (usually floppy disks), and simulates multiple drives if only one drive is connected. It does this by prompting for the right disk to be inserted whenever another drive is required.

The Object and Volume Managers are particularly useful in: (a) reducing the length of time spent reading disks; (b) making it possible to use non-system disks because many utilities have been loaded into RAM; and (c) making do (cheaply) with only

one drive. Their development was probably hastened by the fact that Micro Concepts' prototype machine had only one 200k, single-sided, single-density floppy drive, and all the initial Tripos development was done using this (try doing that with Unix). Tripos is very small: the kernel device drivers, system utilities and BCPL compiler (both source and binaries) will fit on one 720k, 3.5in floppy disk.

Wild-character expression, such as is available with MS-DOS where a '*' means any sequence of characters in a filename, is not currently part of the Tripos or AmigaDos CLI, and has to be handled by each applications program. Micro Concepts' Tripos III will include this facility in the CLI and as an operating system function call.

Future implementations

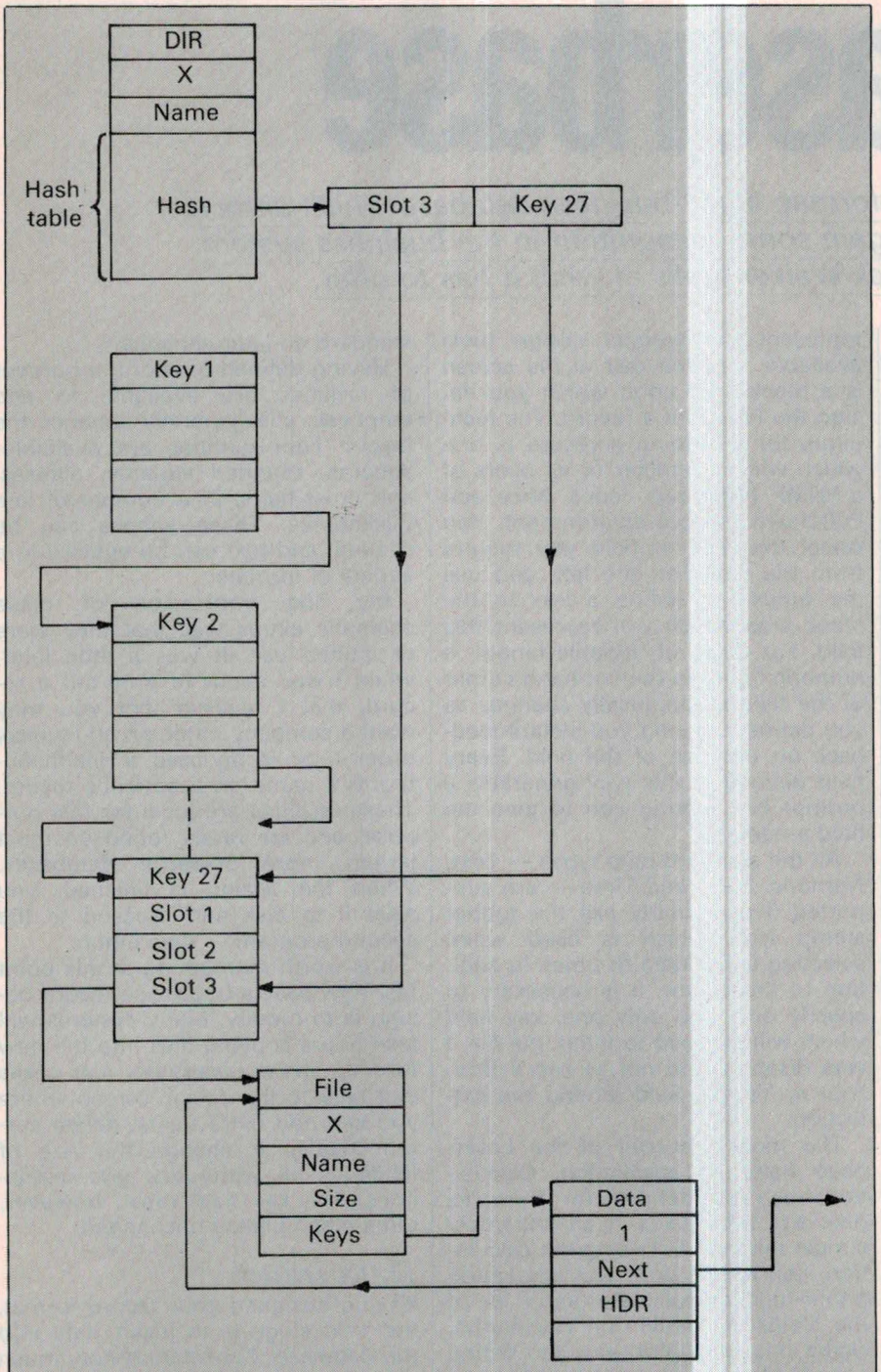
Despite the apparent predominance of 68000 and 68010 systems, Tripos is not limited to those machines and will run very happily on any system with a 'flat' address space; more than about 128k of word- (not byte-) addressed, unsegmented RAM; a floppy disk; and a 16-bit or a 32-bit microprocessor. In practice this means a 32-bit processor, as the common 16-bit micros do not implement word addressing (for example, the 8086 and 80286).

Future implementations of Tripos can be expected on NatSemi 32000 series microprocessors, which are generally considered to have a cleaner architecture than the 68000, and versions for the Torch XXX and the Atari 520ST could probably be brought to market extremely quickly if there were demand. Tripos should also run very well on most RISC machines, such as those produced by Acorn (the ARM) and IBM (the PC/RT), but its message-passing system would not sit very happily on the Transputer, which passes messages as data rather than by an address pointer (Micro Concepts is considering a Transputer second processor for the Microbox III, but the Transputer itself would not run Tripos). Tripos is a high-level system and is not dependent on the physical size of the databus, as long as it is adequately multiplexed. It could run on (cheap) machines which use 8-bit bus versions of the 68000, the 32000 or the ARM.

Although there is not much to be gained by rewriting Tripos in a language other than BCPL, for reasons of business confidence it is possible that a C-based implementation may have more commercial success, even though from a standing start, a BCPL implementation is probably faster to port onto a new machine.

It is possible to envisage a virtual memory version of Tripos, where a number of virtual Tripos machines would run as tasks on a conventional Tripos system. This would slow down the response, but that is inevitable for all virtual memory operating systems (such as Unix).

What happens next with Tripos depends directly on how many systems are ported onto how many more machines. New machines are constantly appearing, and easily portable operating systems (POSs) have a distinct speed advantage in the porting procedure. By a process of natural selection, portable systems will gradually displace the CP/M-alikes and MS-DOS-alikes of the world from the next generations of small, powerful computers in business, education and the home, as well as making inroads into embedded systems and programmable controllers.



Exactly like the 'standard' FH3, but the hash table points to intermediate 'key blocks', which each contain 11 'slots' which are file header addresses. Holding several file header addresses in a single block speeds up directory access.

Fig 3 A Tripos III file handler: FH5

Acknowledgements

I wish to thank Dr Mike Richardson for his help in researching this article, and to the software engineering sub-group of the Cambridge University Caving Club for its constructive criticism. Any misapprehensions are my own.

END

The new 1M

With a memory like we haven't for

Internal power supply.

Resolution: 640×400 pixel monochrome or 320×200 with 16 colours, 640×200 with 4 colours.

TOS in ROM creates a workspace of over 700K bytes.

94-key QWERTY keyboard with numeric keypad and cursor controls.

The price! Under £1 a Kb (excluding VAT) including colour monitor.

Monitor: 12″ high-resolution monochrome or 14″ colour.

Integral 1Mb (unformatted) double-sided 3½″ disk drive.

Two-button mouse.

Port for mouse or joysticks.

An enormous 1024K Ram plus a powerful Motorola 68000 processor running at 8MHz.

Ab 1040STF

that you can be sure gotten a thing

The Atari 1040STF employs state of the art 16/32 bit technology. Yet its price is unbeatable.

The ST range of computers already has a large number of software programmes available, including word processors, spreadsheets and databases, as well as a variety of programming languages and specialist business packages.

The 1040STF will also run software written on several other popular operating systems, including CP/M.

It has a 1024K Ram, integral 1Mb (unformatted) double-sided 3½" disk drive, two-button mouse and built-in power supply.

The operating system is in Rom, leaving Ram free for applications. Basic and Logo programming languages complete the package.

With 12" monochrome monitor, we recommend it sells for £799 excluding VAT saving you at least £1600 against its nearest rival. The price of our 14" colour system is a remarkably low £999 excluding VAT.

As the American magazine 'Byte' commented, "for some time to come the 1040STF will be the clear leader in price/performance."

For the name of your nearest dealer, ring Teledata on 01-200 0200.

And that includes an unbeatable price

ATARI®

Power Without the Price™