Software Industry Trend To Incorporate System Applications Into OS

Systems applications extend capabilities of an operating system and allow richer I/O and file management, as well as a richer applications environment.

by **Eric W. Wasiolek**, Excelan

Computer operating systems are becoming increasingly sophisticated. Many have the ability to distribute data across machines, to interact with the graphics output of programs located on other computers, and even run executables designed for other operating systems. These 'extended' operating system capabilities are derived from the wide-scale incorporation of system level technologies such as graphics subsystems, networked window managers, file redirectors, co-resident operating systems, indexed record handlers, and networking. This article will discuss the technical details and industry significance of incorporating systems applications into operating system software.

To fully comprehend the impact of this trend, it's useful to understand what an operating system is and how it differs or relates to a systems or vertical application. Fundamentally, operating systems are comprised of subsystems which manage hardware resources. Systems applications extend the basic functionality of these subsystems (Fig 1). Like operating systems, systems applications are generic in nature, provide general services to vertical applications, often reside in 'privileged' or 'operating system' memory, and provide programmatic interfaces to allow invocation of their services. The fact that there are so few appreciable differences between operating systems functions and systems applications functions accounts in part for the trend of simply incorporating such applications into the operating system.

'Incorporation' is simply a tendency among operating system vendors, whether software houses or systems integrators, to bundle systems applications with the operating system software. This bundling may range from offering the system application as an operating system option to distributing the application's routines as a part of the operating system utilities or libraries, to incorporating the application as a standard set of operating system kernel routines.

Technically, systems applications function as operating system extensions. Networking, graphics subsystems, and window managers extend the basic I/O subsystem functionality. Networking allows applications to read/write bytes to/from peripheral devices on remote computers. Graphics subsystems allow sending voluminous bit patterns at high transfer rates to terminal screens. Window managers control the input and output of data to/from specified screen areas. Distributed file servers (DFS), indexed record handlers, and networked indexed sequential access methods (ISAMs) allow the file subsystem to store/retrieve remote/local files, and to read/write bytes within files in a key-indexed fashion.

Co-resident operating systems extend the overall applications execution environment so applications designed for different operating systems may run on the same computer. Jointly, these applications extend basic OS functionality from character screen I/O to graphical, windowed, and networked I/O, and from local sequential file management to distributed and indexed file management, and finally from a homogeneous to a heterogeneous applications execution environment.

Extended Operating Systems

Recent activity in the computer industry indicates the strength of the trend toward extended operating systems—Sun Microsystems is perhaps the foremost example. Sun's SunOS UNIX derived operating system has a DFS (Network File System), TCP/IP networking, a networked graphical windowing system (X/NeWS), and a, 386 version that includes a DOS co-resident operating system. Microsoft's OS/2 has a graphical window manager called Presentation Manager; its extended edition includes an SQL-based database client, 3270 terminal emulation, and a distributed file system (LAN Manager) based on popular networking protocols (TCP/IP, ISO, and NetBEUI/TokREUI). AT&T and the Santa Cruz Operation have incorporated the streams technology into their OS which provides a framework for the execution of multiple network protocols. Even DEC and IBM have announced plans to extend their operating systems with X/Windows, NFS, and VP/IX. Apple's A/UX incorporates X/Windows and TCP/IP networking as well.

I/O Subsystem Extensions

The I/O subsystem controls how data flows between applications and peripheral devices, including input devices like keyboards and mice, and output devices like terminal screens and printers. Users always interact with applications only through the I/O subsystem. Systems applications like device drivers for specialized peripherals, graphics subsystems, window managers, and network protocols are being incorporated into I/O subsystems to allow users to view and interact with applications in a greater variety of ways.

Graphics Device Support

It's increasingly popular to include graphics terminals and associated controllers with computer hardware as CPU power cheapens, and as the number of graphics-based applications increase. However, basic operating systems are designed to read/write characters, not graphic bit streams, to terminal devices.

To meet the requirements for graphics display, many operating system vendors have, or are planning to include, graphics library interface support (most according to the CGI or GKS standards) and kernel driver support for popular graphics terminals. This allows graphics-based applications to communicate with graphics controllers and terminals that also subscribe to these interface standards. Graphics support is added by bundling a user library that makes I/O system calls to a kernel graphics terminal device driver. Standard user libraries, such as CGI and GKS, provide a device-independent top portion so the same application may display on a variety of terminals. The bottom half of the library communicates with device-specific system calls.

Networking

Network software allows applications to read/write data to/from network devices (and ultimately applications on other machines on the same network). Network software thereby extends the I/O subsystem to include intermachine communication (e.g. communication with peripheral hardware physically attached to remote computers).

This extension involves a protocol suite and networking card device driver so an application can send/receive bytes to/from remote computers through the local I/O system. User applications make network calls through a user library that allows the specification of logical machine-name destinations and transmission modes. The network library is implemented on top of the system call interface.

The application, for example, sends information to a remote system by issuing a *send n bytes to machine X* network library call. It then sends *n* bytes to a buffer and invokes the write system call to send the bytes to a kernel buffer and invoke the network driver routine. The network driver reads the bytes, processes them, and writes the doctored bytes to the local network controller card that converts them into a physical format that may be propagated across the network medium. Likewise, kernel-based applications may write to the network media, though the driver routines are invoked directly, and not through the user library.

Window Managers

A recent and sophisticated I/O subsystem extension is window management. Window managers are systems applications that allow users to view and interact with several appli cations in different areas of the same physical screen concurrently. Moreover, some window managers allow such a capability with graphics applications as well (e.g. Microsoft's Presentation Manager). Some window managers, in addition, allow interactic with remote character-based and graphics terminals, as well as local a plications, and even applications running under dissimilar operating

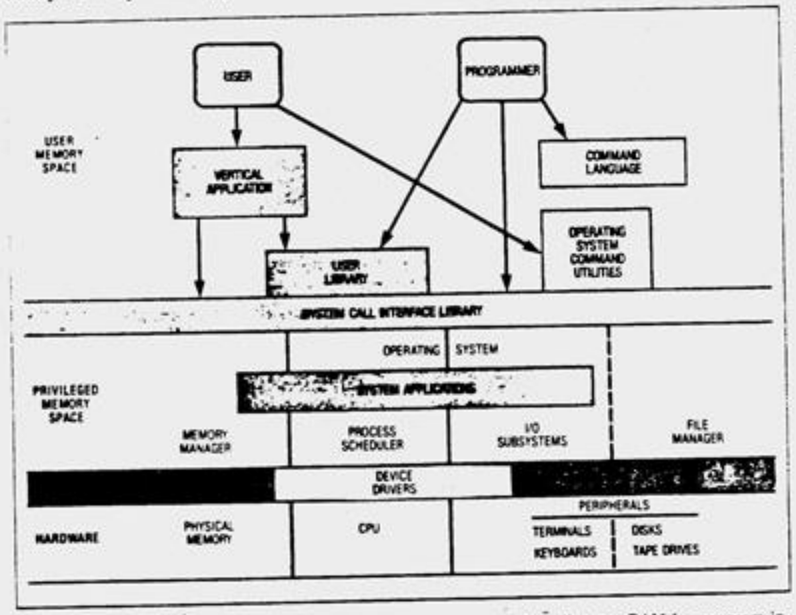

Fig 1 In general, operating systems include a memory manager that allocates RAM for program instructions and data, a process scheduler that determines when the CPU will execute specific routines, a peripheral I/O system that controls information flow, and a file subsystem that determines the organization of data on disk and tape drives. Vertical applications are shielded from these routines through a system call interface. Systems applications may reside in user or kernel memory but, like operating system routines, they provide general services to vertical applications and users. Operating system routines, in turn, communicate with hardware through device driver routines.

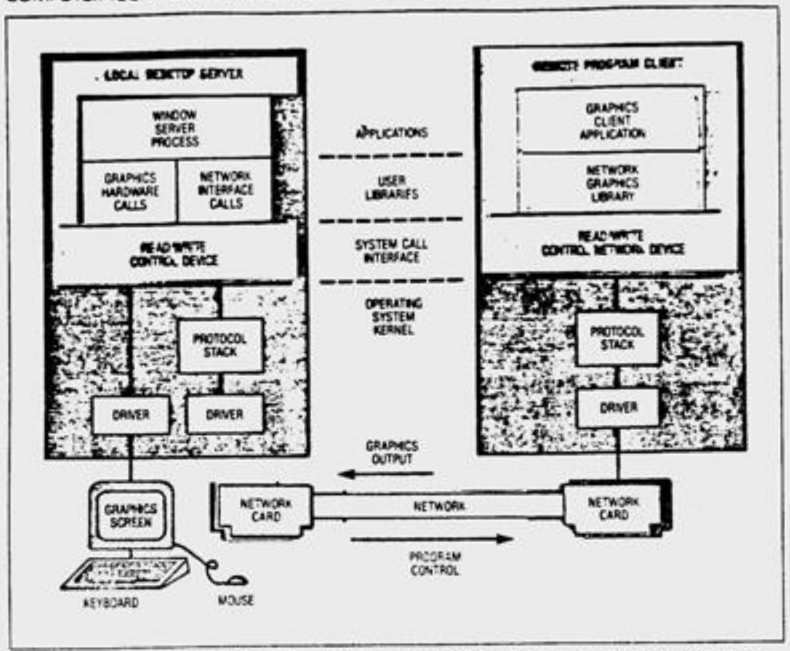

Fig 2 In a networked graphical window management system, a remote window-client application sends graphics output to a local window server process over a network link.

systems (e.g. MIT's X Windows, and Sun's NeWS).

X Windows and NeWS are examples of networked window managers. These products are systems applications that allow a computer program on one system to send its output to the graphics terminal screen of another computer. Such systems applications are distributed according to the client-server model. Programs make window library calls like *open a window, draw a circle*, or *fill this area with this color.* This user library then sends the graphical description formed by window commands through a network transport subsystem to a window server on the system where the destination screen resides. The window server accepts the command and writes the graphical output to the window buffer. The window server process is device-specific. The server then invokes I/O system calls that open the device driver for the graphics terminal screen and write the screen image to the graphics device (Fig 2).

File Subsystem Extensions

The file subsystem controls how applications store/retrieve data between long-term storage devices. Two related technologies, distributed file systems (DFS) and ISAMs, are increasingly being incorporated into OS distributions. DFS allow applications to access files located on remote and local machines. ISAMs allow applications indexed access to data records through specified search values. Moreover, distributed ISAMs may be incorporated by combining ISAM and network technology.

File subsystems are composed of a file system call interface, a file manager, a collection of file tables, and a long-term storage device driver. Applications make file system calls, like *make a file system, create a file*, or *read a file*. Such calls invoke OS routines, collectively called the 'file manager,' which establish and interact with storage hardware device drivers through file tables.

The file manager may be functionally divided into categories of file management routines: administration, access authorization and lock management, allocation, and access control. File tables may be functionally divided into four separate areas:

- A 'per process state table' that maintains information indicating which processes have which files open with which permissions.
- A 'file system table' that maintains a list of used and free disk blocks, and maintains blocks that describe the entire file system and its catalogs (directories).
- A 'file attribute table' and a 'file location table.' They essentially maintain a list of what attributes each file has, and which disk blocks are associated with which files.

File manager routines search these tables to perform their functions. Device drivers, located through an index table, allow the file manager to invoke routines to communicate with particular long-term storage devices such as disks and tape drives.

ISAM File Subsystem Extensions

ISAMs allow an application to store, associate, and retrieve data according to specified search values. The most popular example of an ISAM's use is a database application. Through an ISAM, such an application may store thousands of records in such a way that subsets thereof may be subsequently retrieved by search values. ISAMs achieve this capability by internally organizing the bytes with a file into an index structure and controlling the access to and from data on disks through such index structures. The code that controls indexed access is typically called the 'record handler.'

ISAMs are typically marketed as user libraries. Because they provide basic file services, they're excellent candidates for incorporation into file subsystems, with the ISAM libary calls becoming a standard part of the file system call library (adding 'isamcreat,' 'isamread,' 'isamaddindex' ...), and by record handler routines becoming additional file manager routines. AT&T, for example, has indicated their intention to incorporate Informix Corp's C-ISAM into the UNIX System V kernel. Many mainframe operating systems, especially of the IBM variety, include record types and ISAM access as standard services.

A record handler creates a file through the file system as normal. However, the record handler subsequently controls exactly what type of information is written to which bytes and blocks of the file, thereby internally structuring the file so that indexed access to data may be achieved. The record handler typically organizes the first portion of a file into index field names followed by the block number or byte offset of valid search values for that field. Search values, in turn, point to addresses of data records.

Record handler algorithms adjust byte offsets and block numbers to continue to correctly address data items as new index fields, search values, or records are inserted or deleted. Record handler implementations determine which portions of the index structure stay in core memory, how many index fields, search values, and records of what lengths may be inserted, and whether data records are even contained within the same file as their index structure.

Networked ISAMs

It's technically possible for indexed access to occur across machine boundaries for both 'cooked' and 'raw' ISAM implementations. In the cooked case, a local application makes a local ISAM user library call which is converted into the appropriate local file system calls with the specified byte offsets. Rather than being executed locally, the file system calls are then routed through a network driver to the destination machine and read by a listening ISAM subprocess that executes the file calls and returns the requested bytes to the local ISAM application buffer back over the network.

In the 'raw' case, the local application makes an ISAM call that invokes a record handler routine which consults a global location file to determine the byte, block, and machine location of the requested information. If the information is on another machine, the record handler negotiates the byte transfer through the network subsystem. It's possible in a raw implementation for a single database file to span multiple machines. Networked ISAMs, especially the cooked variety, are excellent candidates to be incorporated as standard 'networked' operating systems services.

Distributed File System Extensions

Distributed file systems (DFS) are systems applications that can store and retrieve files located on long-term storage peripherals of remote computers. Depending on the implementation, remote file access may be partially or completely transparent to the user. DFS typically provide a programmatic interface so applications can be developed to store and retrieve remote files. Often the remote interface is identical to the local interface because the DFS technology is buried within the kernel.

Examples include Sun Microsystems who bundles, Network File System (NFS) with SUNOS its distribution of UNIX. AT&T includes the basis of their UNIX System V DFS called Remote File Sharing (RFS)—namely the 'streams' technology and 'file system switch' (FSS)—with all Release 3 and

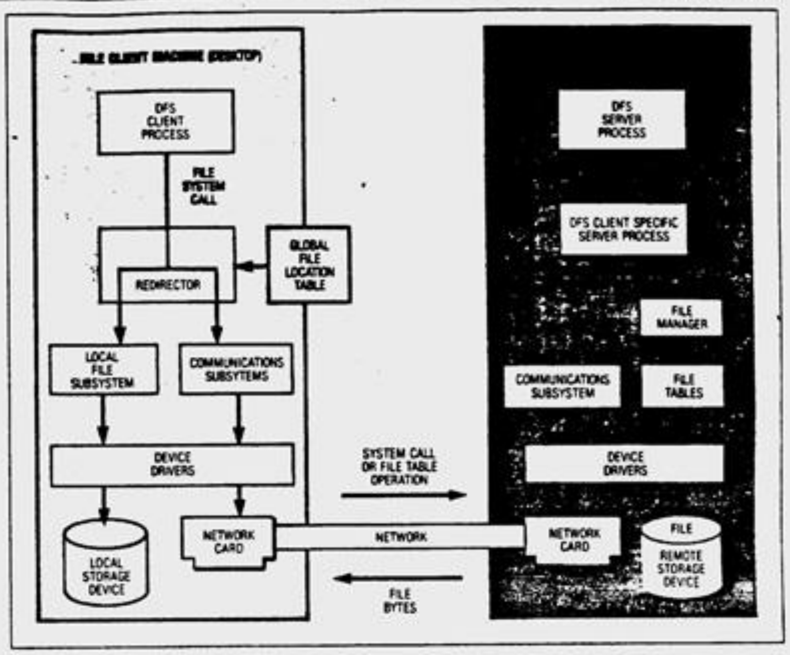

Fig 3 Whether implemented in user or operating system memory, DFS employ a distributed client-server architecture. The client (or redirector) redirects local file system calls or their file table operations across the network to the remote machine where they are executed, returning information across the network which is normally returned to the application through the local file subsystem.

later versions. Microsoft is releasing its OS/2 LAN manager operating system with a DFS client and server (MS-NET).

DFS are distributed systems applications which employ a client-server architecture. A local client process called the 'redirector' intercepts file system calls from the local application and consults a table to determine the location of the file (whether it's on a local or remote storage device). If the file is remote the client process redirects the local system call via a communications subsystem to the DFS server process of the destination machine. The server then executes the system call on the remote file system on behalf of the local client.

Although all DFS follow this general model, implementations differ. The redirector may execute within user or kernel memory. DFS servers may be implemented within user or kernel memory space as well. The DFS server listens for system calls or file operations from the client. Upon receipt the server interacts with the remote file manager to execute the system call or file table operation. Typically, the server creates a client-specific server subprocess to handle all subsequent interactions with the client. Communication is streamlined by avoiding the server bottlenecks associated with concurrent communications with multiple clients. If the server or server agent process is user memory based, the server executes the system call as a normal user application. If it's kernel- based, the server intercepts the entry-point file subsystem routine, which is the system call or file table operation.

The DFS communications subsystem includes network services. No DFS is possible without networking. The DFS client and server pass system call or file operations to one another by writing to and reading from the communication subsystem's application interface. In some instances, this interface may involve a remote procedure call (RPC) and external (machine- independent) data representation interface, as in SUN's NFS use of RPC and XDR (Fig 3). ■

Eric W. Wasiolek, *product manager of UNIX products at Excelan Corp. (San Jose, CA), has authored numerous articles on heterogeneous and distributed computing systems and applications technologies. He graduated from the Univ. of California at Berkeley.*