INSIDE DIGITAL RESEARCH'S FLEXOS/386 AND X/GEM

by Peter Ruber

Part 3

In the DOS world, device drivers are recognized by the operating system when you place the driver name after a DEVICE= statement in a CONFIG.SYS file, and then reboot the system. FlexOS allows you to dynamically load and unload drivers from the command line while the system is operational. In order for you to do this, the device must be physically attached to your system, and the device driver has to be located in the DRIVERS subdirectory.

Those of you who have worked solely with a single-user DOS system may not appreciate the need to load and unload devices "on-the-fly," because the configurations of your systems aren't likely to change too often. By virtue of its design principle, FlexOS operates in environments where hardware devices may need to be loaded and unloaded to accomplish different realtime operations. This is particularly advantageous in automation and process control applications that continue round the clock. You just can't shut down the system at will.

If a device is to be permanent, its driver and the operating parameters can be placed into the system's CONFIG.BAT file (to be discussed shortly). If, on the other hand, the device will only be used sporadically, its driver can be invoked from the command line with the DVRLOAD command. The syntax is:

```
DVRLOAD   device_name:
driver_name access_level
```

A number of access-level options can be assigned to a driver. They can be one or more of the following:

E Exclusive access denied. Only shared access is allowed, assuming the N option is also set.

L Lockable. Allows a single user to obtain exclusive control (lock) of a system-wide device.

M Multiple Partitions. When you create a partition with FDISK, you must use the M option with DVRLOAD and DVRUNIT when loading HD0:, DH1:, etc.

N Shared access enabled. Two or more processes can access the same device simultaneously.

P Permanent driver. Once installed, the driver cannot be removed. The P option is useful for devices that do not need to be changed (for example, a hard disk).

R Raw Read access allowed. Data can be read directly (without interpretation) as it appears on the media. You do not see filenames or the relationships of files within directories.

S Raw Set access allowed. Attributes of the driver itself (like the country code or baud rate of a console) can be changed.

V Verify Writes. Verifies all writes to a device. This option can affect performance.

W Raw Write access allowed. Data can be written directly (without interpretation) to the media.

The DVRUNIT command adds another device of an existing device type to the system. If your system has one hard disk drive and you want to add another, you must use DVRUNIT. Access levels can be attached to this command.

The DVRLINK command associates one driver with another. For example, you can link a printer driver to a port driver. This enables the printer to communicate with your computer through a specified port. In the command line syntax, the device_name is followed by the name of the sub_driver which is associated with that port.

DVRUNLK removes a driver from the FlexOS system that had been installed with the DVRLOAD and/or DVRUNIT commands.

```
-----------------------------------------------------------------------
name: process   pid : 7   wndw: 3             pcon: 5
fmly: 2         grp : 99  user: 11            stat: running      *
prty: 200       ppid: 5   (user process)      (main context)
evnt: 0000 0000 0000 0000 0000 0000 0000 0000
-----------------------------------------------------------------------
name: shell     pid : 5   wndw: 3             pcon: 5
fmly: 2         grp : 99  user: 11            stat: waiting
prty: 200       ppid: 4   (user process)      (main context)
evnt: 0000 0000 0000 0000 0000 0000 0000 0000
-----------------------------------------------------------------------
name: wmanager  pid : 4   wndw: 3             pcon: 5
fmly: 1         grp : 99  user: 11            stat: waiting
prty: 190       ppid: 3   (user process)      (main context)
evnt: 0000 0000 0000 0000 0000 0000 0000 0000
-----------------------------------------------------------------------
name: logon     pid : 3   wndw: 0             pcon: 5
fmly: 1         grp : 99  user: 11            stat: waiting
prty: 210       ppid: 0   (user process)      (main context)
evnt: 0000 0000 0000 0000 0000 0000 0000 0000    (system manager)
-----------------------------------------------------------------------
```

Fig. 1: An example of the Process View display.

The CONFIG.BAT File

There are no CONFIG.SYS files in FlexOS. System drivers and associated parameters are contained in an extensive CONFIG.BAT text file that the operating system reads during the bootstrap. It defines the system's console, its hard disk and floppy drives, the serial and parallel ports, pointing devices, etc., and the syntax for their associated drivers. Since a FlexOS system will likely have additional consoles (terminals) and multiport cards, plus hardware that controls realtime and event-driven devices, the CONFIG.BAT file can be liberally augmented by REM statements that define the purpose and use of each driver. This provides a reference to future system managers when they are first introduced to a FlexOS system.

Other types of batch files are allowed on FlexOS that can perform nested operations. In this case, program control flows from the batch file containing the BATCH filespec command to the named filespec. The function and operation of FlexOS batch files is very similar to DOS batch file functions. The AUTOEXEC.BAT file, as used in FlexOS, sets the system PATH, the prompt, and, with the DEFINE statement, the paths to various subdirectories.

Process View

This is one of two commands which enable system administrators to obtain information about what is going on in a multiuser, multitasking FlexOS system. The other one is SYSTAB.

The PROCESS command lets you view or delete processes currently running on the system. A process is defined as any command, program, or application running under FlexOS. The PROCESS VIEW command, followed by an option, provides a wealth of information that includes process priority, parent ID, process type, context, event flags, memory statistics including maximum process memory, code, data, heap start, and memory status.

It also provides paged output, as opposed to streamed output, a User filter which displays only those processes that match this user's Group and User IDs; and it provides a Family filter which displays only those processes that match this user's Family ID. The PROCESS CANCEL ID=##### command can be issued to remove any running process. See Figure 1.

SYSTAB

The SYSTAB command display the current status of the system tables. The -P# option pauses the display between updates for # seconds, where # is a number from 1 to 9. The -T option displays the system clock time in the bottom right corner of the window. The -? option displays the help screen.

After you invoke SYSTAB, the SYSTAB Menu appears. Users may either move the cursor to the desired option and press Enter, or type the option letter:

A Displays all the system tables in memory. The tables appear in the same order as the options listed below. They remain on screen for the number of seconds specified by the -P option when SYSTAB was invoked. SYSTAB continuously cycles through the tables.

B Displays the Process Environment Information Table.

C Displays the Pipe Information Table.

D Displays the Physical Device Table.

E Displays the Virtual Console Table.

F Displays the verbose form of the Virtual Console Table.

G Displays the System Logical Name Definition Table.

H Displays the Process Logical Name Definition Table.

I Displays the Shared Memory Information Table.

(continued on page 389)

◆ FLEX OS/386

(continued from page 378)

J Displays the Calling Process's Environment Table.

K Displays the Console Information Table.

L Displays the System Memory Information Table.

M Displays the System Environment Information Table.

N Displays the Shared Runtime Library Table.

O Exits SYSTAB.

The UNIX Utilities

FlexOS/386 provides two groups of UNIX utilities. The first group contains some 18 UNIX utilities in the UNIX_UTL subdirectory in the Programmer's Toolkit. These are popular public-domain programs commonly used by UNIX engineers which have been ported to FlexOS by using the MetaWare High-C cross-compilers and the FlexOS software development library. When FlexOS/386 is installed on a hard disk, these utilities are automatically moved into the BIN subdirectory where the FlexOS programs are stored. They are referred to as the UNIX-Like Utilities in the FlexOS user manual.

The second group is included in the software development tools. It is expected that DRI will soon include in a forthcoming update a popular and widely used UNIX text editor called MICROEMAX, which they are porting to FlexOS with the author's permission.

The UNIX-like utilities, which are now a part of the basic FlexOS/386 operating system, are:

Command	Description
BANNER	Prints banner pages
CAT	Concatenates files
CMP	Compares two files
CUT	Cuts out specified fields from each line in a file
DIFF	Finds differences between two files
DUMP	Displays file contents in hexadecimal, octal, decimal, or signed decimal form
DUP	Prints multiple files of the same string
ECHOX	Outputs special characters
FGREP	Searches a file or files for the string-entered
GREP	Searches a file or files for the pattern-entered
LC2UC	Translates lowercase characters to uppercase
MAKE	Builds programs
PASTE	Merges lines from a file or files
PR	Prints a file or files
SPLIT	Breaks down a file into equal-size files
STRINGS	Prints all sequences of printable characters in a file
SUM	Calculates file checksum and block counts
UC2LC	Translates uppercase characters to lowercase
WC	Counts file lines, words, and characters

This concludes the overview of the FlexOS command structure.

Interprocess Communications

An important feature of FlexOS' internal structure that I have only mentioned so far in passing, and which deserves some discussion, is how FlexOS manages multiprocess communications with its devices. This is accomplished through pipes and semaphores, and...

Digital Research's Concurrent

(continued on page 408)

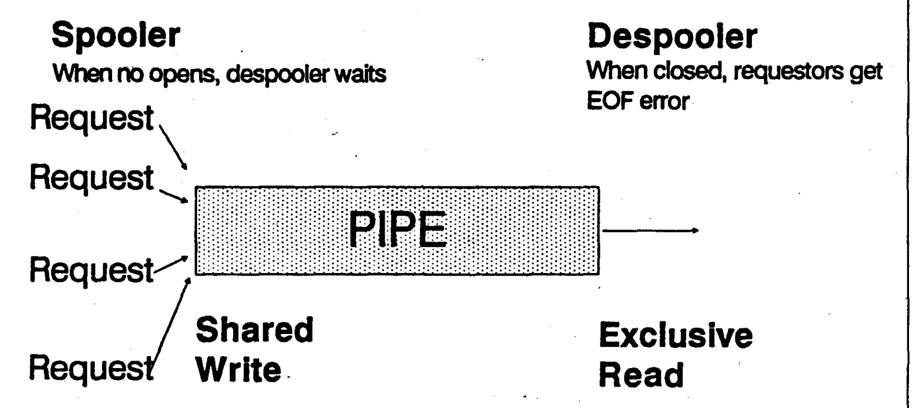


Fig. 2: The Spooler Pipe

◆ FLEX OS/386

(continued from page 389)

DOS operating systems contain the hooks and the necessary system calls for interprocess communications in its native programming mode. Although they allow for multiple threading of processes in background operations specific to control and data acquisition applications (Heuristics, Inc., (916) 369-6175, is the chief implementer of this tech nology), it doesn't have the sophistication of named pipes and semaphores, nor the power that is available under the FlexOS technology.

FlexOS supports interprocess communications with a device designated pi:. Processes can use an in-memory file called a pipe, created on such a device to establish a buffer used for depositing and withdrawing messages. Pipe files have two ends: one to write to and one to read from. Processes deposit and withdraw messages on a first-in, first-out (FIFO) basis. The number of messages a process can store at one time is limited only by the pipe's length.

SVCs used for managing pipes are CLOSE (close a pipe), CREATE (create and open a pipe), DELETE (remove a pipe), GET (retrieve a pipe table), LOOKUP (scan and retrieve pipe tables), OPEN (open a pipe), READ (read from a pipe), SEEK (set or retrieve file pointer), and WRITE (write to a pipe). The READ, WRITE, or DELETE privileges have the same meaning for pipes as for disk files. The name field contains the address of the pipe name. The record-size parameter regulates the message blocks. A record size of four means all pipe I/O is conducted in 4-byte blocks. The File Security Word sets the Owner, Group, and World access privileges. The size is set to the pipe buffer length.

The Pipe Resource Manager maintains a directory of all existing pipes. Each directory entry includes the pipe creator's User and Group IDs and the File Security Word. The Pipe Resource Manager also makes a PIPE Table for each pipe, indicating the values set by the CREATE call. No special access privilege is required to use LOOKUP to retrieve PIPE Tables, but the calling process must have opened the pipe to use GET. The DELETE SVC removes a pipe. A CREATE option can be selected that automatically deletes a pipe on the last close. If the pipe is being used solely to communicate between two or more processes for the life of the processes, the pipe is deleted automatically when the processes terminate.

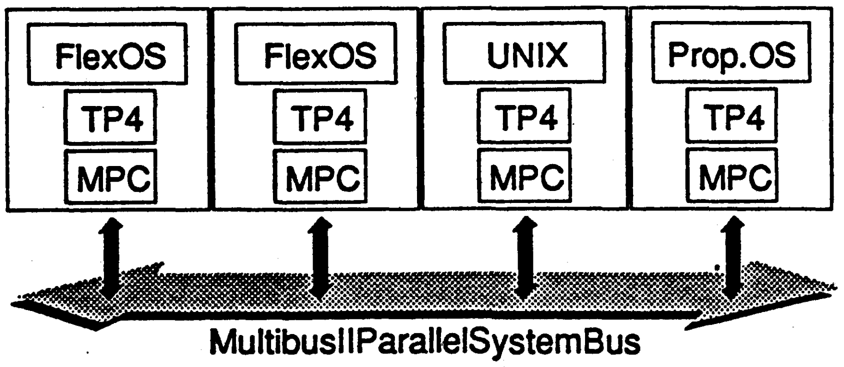


Fig. 3: The TP4 Transporter (ISO 8073 standard) provides virtual circuit services for interprocess communications over a MULTIBUS II Parallel System Bus.*

Accessing Pipes

Processes must open a pipe before they can read from or write to it. When the OPEN call is made, the Pipe Resource Manager compares the User and Group IDs of the calling process with those in the pipe's directory entry to determine which access privileges are available. Should none of the privileges match, the OPEN fails.

A pipe acts differently depending on whether an end is opened in Exclusive or Shared mode. If one end of a pipe is opened in Exclusive mode and then closed, a process attempting to read or write on the other end receives an end-of-file

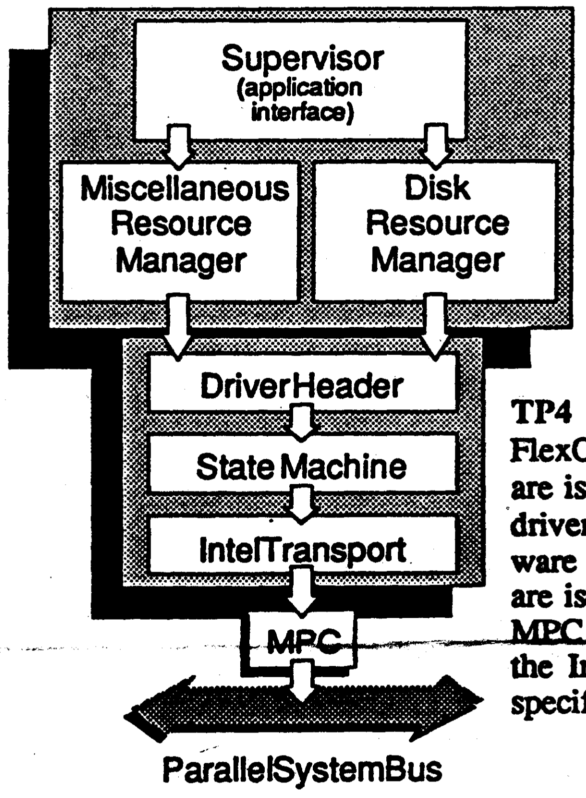
(EOF) error. This happens independently of how the other end was opened, whether it's currently open, or if the last open was in Shared mode; the process accessing data through the opposite end waits until the operation is complete.

If one end of a pipe file is opened in Shared mode and then closed, FlexOS treats the pipe as if it were still open on the other end. Therefore, any process accessing it waits until the operation is complete.

Note the distinction between Shared mode and Shared File Pointer mode. A pipe opened in Shared File Pointer mode is shared only by those processes with the same Family ID (FID). After a pipe end that was opened in Shared File Pointer is closed by all the processes, any process accessing the other end receives an EOF error. This notifies the calling process that a process on the other end of the pipe has either closed the file or terminated.

Using modes to restrict access is consistent with spooler-type applications. For example, a spooler process could create a pipe and reserve for itself Exclusive access to the read end, leaving the write end available for Shared access by any other process (see Figure 2).

Processes open and close the write end when they are sending files to the spooler. When the write end is closed, the spooler waits until it is opened by another process. If the spooler closes the read end, processes attempting to write to the other end receive an EOF error that indicates to the writing process there is no process at the other end to read its file.



FlexOS Hardware-independent modules

TP4 Transporter FlexOS dependencies are isolated to the driver header. Hardware dependencies are isolated to the MPC interface (here the Intel Transport specification).

Fig. 4

Interprocess communication takes place when processes use the READ SVC to get data in the buffer. The READ and WRITE flags and parameters are the same for pipes as they are for disk files, and the file pointer is maintained. There is no limit to the number of processes that can participate in the exchange.

The amount of data written to and read from the pipe is independent of the pipe's size. When the amount of data exceeds the pipe size, the following procedures are observed:

* On write operations, when the pipe is full, the process waits for another process to read data from the other end. The event completes when the process reading the data removes enough to compensate for the difference.

* On read operations when the

pipe is empty, the process waits for another process to write data to the other end. The event completes when enough data has been written to compensate for the difference.

Pipes are often used to make one program's output become the input of another. To do this, a parent process (often the FlexOS shell) performs the following steps:

1. Use CREATE to create a pipe.
2. Use DEFINE to redefine the STDIN file to be the name of the pipe.
3. Use COMMAND to create the receiving (child) process. The child process inherits the parent's PROCDEF Table, including the STDIN prefix.
4. Use DEFINE to reset the STDIN file to the original name.
5. Use DEFINE to redefine the STDOUT to the pipe name.
6. Use COMMAND to create the source process. This child process inherits the redefined STDOUT, but, unlike the receiving child, has the original STDOUT.

When the two processes terminate, the parent process closes the pipe. If the parent terminates before the children, the pipe is automatically removed when the children terminate.

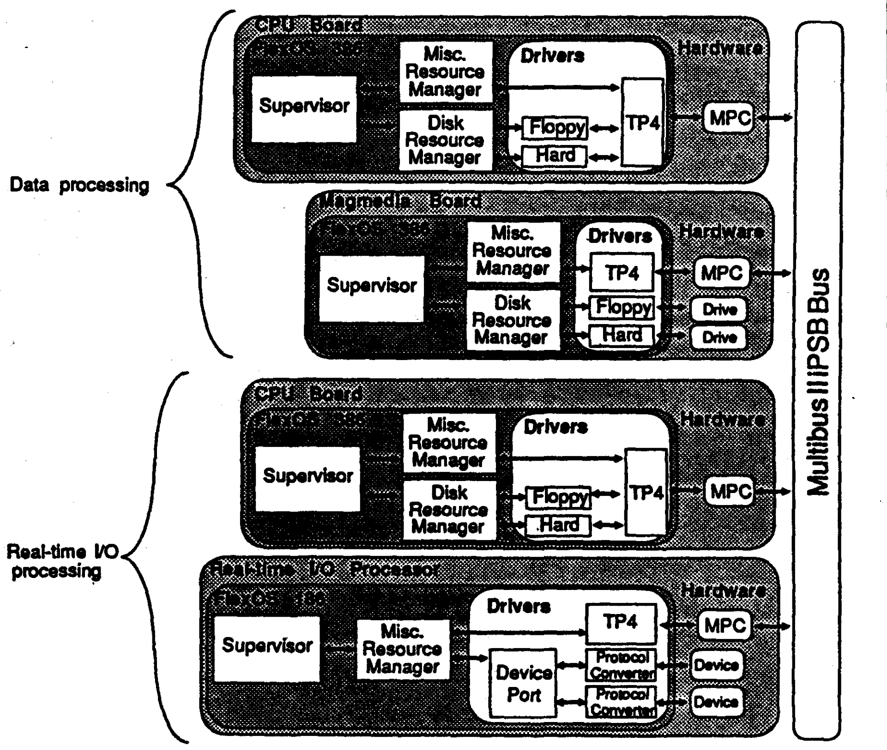


Fig. 5: *A cell controller features FlexOS 386 in real time and data processing CPU boards and on the Magmedia board, with FlexOS 186 in real-time I/O handlers.*

Synchronization and Exclusion

The Pipe Resource Manager allows a process to create a pipe with a zero-length buffer size for use as a simple semaphore. A READ operation obtains the semaphore pipe and a WRITE operation releases it. If another process has obtained the semaphore pipe previously, the calling process waits until a WRITE operation to that pipe has been performed. WRITE operations, on the other hand, never wait; if the semaphore pipe was released previously, the call returns without error.

The process creating the semaphore pipe automatically owns it from the start. Although the Pipe Resource Manager keeps a record of who read the pipe, a WRITE by any process releases it. The Process ID is maintained for two other reasons: first, so that a process can call READ multiple times on a semaphore pipe it already owns, and second, so the Pipe Resource Manager can release semaphore pipes owned by a process that has terminated.

A semaphore pipe can regulate access to a resource not managed by FlexOS. Whenever a process wants ~~to use the resource, it reads the~~ semaphore pipe. If the semaphore pipe is already owned by another process, the calling process waits until that process releases it with WRITE. Upon return from READ, the process is free to use the resource. Upon completion, the process writes to the semaphore pipe, which releases the resource so that other processes can use it.

FlexOS on a Multibus II Parallel System Bus

Earlier in this series, I made a number of bold statements about the ability of FlexOS to move to many different hardware platforms. In addition, I said that FlexOS can coreside and function concurrently with other operating systems in large-scale computing environments; that multiple versions of FlexOS can exist on the same system; and because of its support of industry standards such as TCP/IP and POSIX (among others), that it offers intercommunication capabilities to DOS, UNIX, and OS/2 systems, to minicomputer and mainframes, as well as to proprietary operating systems such as Intel's iRMX, which actually is a realtime kernel for embedded control applications, rather than a full-fledged realtime operating system like FlexOS.

Harnessing Real Power

At present, only Intel's 386 Multibus II hardware platform can harness FlexOS' real power, because it provides a parallel systems bus which has the extended logic and architecture necessary to support multiple FlexOS coprocessors. Within the next few years, when IBM's MicroChannel Architecture and the emerging EISA (Extended Industry Standard Architecture) buses move to the next level of development, we will see multiprocessor systems becoming more commonplace. When these buses incorporate the Intel 80486 processor (and ultimately the 80586 processor, which is slated for release sometime in 1993 or 1994), we will have mainframe power sitting on our desktops. However, in terms of present-day realities, FlexOS already provides greater-than-minicomputer power through an interesting new product called the TP4 Transporter.

In the fall of 1988, American Manufacturing Systems released the TP4 Transporter (ISO 8073 standard) for Multibus II systems which it had codeveloped with Digital Research. The TP4 is a software module that provides full transporter services for the media-transparent transfer of information between bus agents through a message pipe, which is a type of file optimized for interprocess communication and synchronization. Bus agents are individual physical CPU boards. The value of separate bus agents is that they allow integrators to distribute the processing load among independent, intelligent agents interconnected through a backplane. With their combined functionality, the data processing and I/O-handling horsepower will exceed the high performance of most minicomputers. Furthermore, these multiprocessor systems also offer a high level of configuration flexibility and performance characteristics that are immune to I/O load, even in I/O-intensive realtime applications.

Intercommunications between agents over a bus parallels intermode communications over a network. Applying the International Standards Organization's (ISO) model of Open Systems Interconnect (OSI) illustrates the similarities and highlights the issues that need to be resolved to ensure reliable communications. Figure 3 illustrates how the Intel Multibus II Parallel System Bus (iPSB) and the Message Passing Coprocessor (MPC) provide services that are equivalent to the OSI-defined physical and datalink layers for managing the passing of messages between nodes.

Next is the transport layer. It provides media-transparent transfer of all data between users. It also assures the reliable transfer of information. On an operational level, this means the application simply opens, reads, and writes to a file to effect interagent transfers, as I described in the previous section dealing with FlexOS' interprocess communications.

Establishing Connections

Without a transporter like the TP4, applications must deal directly with the media and its message-passing mechanisms to establish connections between software entities. There is a considerable code burden associated with these functions, which can detract from a programmer's ability to focus on the application. The TP4 Transporter handles all of the media control functions. This relieves the application writer from the complex tasks of message passing, because the transporter transparently establishes the connection, breaks down the message into packets, arbitrates bus access to send each packet, reassembles the packets into the message, and reports any errors.

Connections are defined by agent and socket IDs so that multiple connections between the same agents are supported. This allows processes on each agent to have multiple and independent channels of communication between each other. The TP4 Transporter has been implemented first for use with FlexOS on the Multibus II platform. The software is modular and designed so that the core components are portable to other system software platforms. This allows a Multibus II developer to use the TP4 Transporter for intercommunication between FlexOS, UNIX, and/or proprietary software agents in the same system as illustrated in Figure 3. Additionally, it also allows for the integration of Intel's iRMX/iRMK system software to be integrated into a FlexOS/UNIX multiprocessor system. Applications can be developed either on the Multibus II platform, or the standard, lower-cost FlexOS/286 or FlexOS/386 development environments. Interagent communications are easily simulated using standard FlexOS named message pipes. As an example, the processes on both agents can be loaded and run on a single FlexOS system with a named pipe used to test message transfers. Moving it to the Multibus II environment is simply a matter of replacing the create-pipe function with the make-connection function and loading the processes in their respective agents.

Taking Care of Details

The reason why the TP4 Transporter supports the ISO 8073 standard results from the close similarities between interagent communications on a Multibus II system and internode communications on a network. It provides a portable and functional layer that takes care of the media-specific details of establishing virtual circuits and the transmitting of solicited and unsolicited messages. ISO 8073 is also the standard transporter for MAP/TOP installations, allowing for easy integration with these systems. MAP/TOP is one of the major protocols used in large factory automation installations by General Motors and other manufacturing firms.

The internal structure of TP4 is made up of three discrete sections (see Figure 4). There is the driver header, the transporter state machine, and the low-level interface. All TP4 Transorter code is written in C. The transporter state machine is operating system- and backplane-independent. This module handles the logical operations surrounding bus address translation, virtual circuit management, and message packetization and assembly.

The driver header reconciles the state machine with the operating system. FlexOS is a machine-independent operating system with all hardware dependencies isolated in device drivers. All FlexOS device drivers have a standard interface to the system for control operations such as opening and closing the file or device, passing functions, arguments, and data, flushing buffers, setting control parameters, and returning results. The TP4 Transporter driver header for FlexOS converts the information passed through this interface to a form meaningful to the state machine.

The low-level interface reconciles the state machine with the bus-control hardware. In the FlexOS version, the TP4 Transporter uses the Intel Transport specification to manage access to the Message Passing Coprocessor. The TP4 Transporter provides fast, predictable realtime response to events on other agents, independent of bus load. Two features govern bus use and message transfers. Large transactions are broken down into small packets. Although this results in a slight decrease in bus throughput because the bus arbitration cycle is repeated more often, there is an advantage, because processes can interleave their messages with lengthy transactions between other processes.

(continued on page 437)

(continued from page 413)

The second feature allows a process to bypass the current transport queue in order to get a message to another agent/process without waiting for a current transaction to complete. The TP4 Transporter preserves the buffers queued up at both sending and receiving nodes while handling the expedited message. Together, these two features provide an orderly and flexible mechanism accommodating the two important aspects of bus communications in realtime systems: transferring large messages without interfering with other processes' use of the bus; and aborting/resetting a transaction without waiting for it to complete.

A FlexOS/TP4 Realtime System

FlexOS and the TP4 Transporter are currently integral parts of the American Manufacturing Systems Orchestrator, a cell controller that combines data acquisition, process monitoring, and powerful data-processing capabilities.

The architecture of this modular system (shown in Figure 5) uses standard 80386- and 80186-based board agents. One set of CPUs handles the critical activities of realtime control; the other set handles the less time-dependent data-processing functions. Each realtime agent is individually and dynamically programmable and configurable. Disk I/O is supported through the Magmedia Board.

The TP4 Transporter is a FlexOS loadable driver delivered as a part of the FlexOS Multibus II Driver Pack. This pack contains a full FlexOS system executable on a Multibus II system with the Intel iSBC 386 single-board computer, the American Manufacturing Systems Magmedia Controller, and the Concurrent Technologies M-PC 186/010 Intelligent I/O Controller. Also included are target system driver sources, and a TP4 Transporter in loadable and linkable form.

In this system, data-processing and realtime components share use of the bus for lengthy data transactions and short data bursts. It is not unusual for the data-processing CPU board to load a file through the Magmedia board while, simultaneously, the realtime CPU board was receiving short, asynchronous control or status updates from a realtime I/O processor. Predictable, realtime response occurs despite the disk load, because the TP4 Transporter breaks down lengthy logical transactions into a series of small packets. At the end of each packet, the I/O processor can initiate a message transfer. When that packet concludes, the bus arbitration cycle is repeated so that the data-processing CPU and the Magmedia boards can pick up where they left off.

Because FlexOS' modularity allows systems to be built with any combination of core components (see Part 1 of this series for the unwanted or unnecessary resources) and overhead can be eliminated from any CPU agent. In systems that may contain dedicated agents (such as robotic assembly), FlexOS/186 can be placed into ROM.

In January

Next month, we begin our exploration of the X/GEM multitasking graphics interface for FlexOS/386. I will also explore X/GEM's portability to X Windows under UNIX and other software platforms. If you're interested in getting a jump on this section, you can refer to my article entitled "GEM Moves to OS/2's Presentations Manager and X Windows Under UNIX" in the December 1988 issue of *Computer Shopper*.

For detailed information about the FlexOS/386 and X/GEM Programmer's Toolkits and System Builder's Kits, refer to the ending paragraphs of Part 1 in this series. Or contact Digital Research, Inc., Box DRI, 70 Garden Ct., Monterey, CA 93942; phone (408) 649-3896; or you can call their North American sales office at (408) 982-0700.