Technical articles

Smart link comes to the rescue of software-development managers

Resource-management hardware and software join existing development systems into an Ethernet-based network that eases software creation and control

by James P. Schwabe, *Intel Corp., Santa Clara, Calif.*

☐ A strong lifeline in a sea of complexity, the new NDS II network development system will help manage the writing of complex software for tomorrow's powerful microsystems. It builds on existing Intellec development systems and the specifications of the Ethernet protocol to create a local network for distributed software development.

Considerable intelligence is contained within the NDS II system, linking programmers' work stations and managing the interactive flow of software development that results. Communications control, via Ethernet or an even simpler alternative, is split between the central manager and the work stations.

At the heart of the system is the network resource manager, which both controls the net of work stations and lets the user configure it to suit the development task under way. The NRM will also manage a powerful system memory of Winchester-technology disk drives.

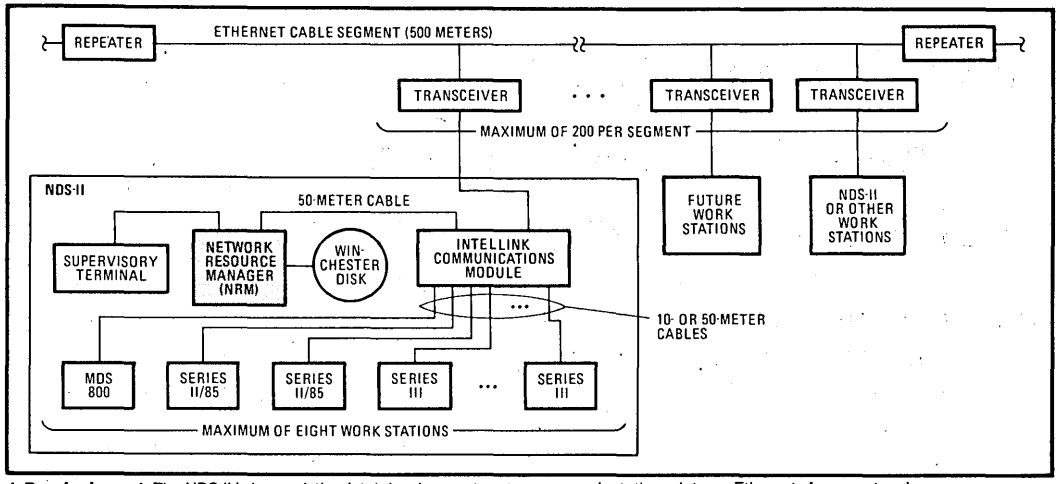
The manager itself is an example of the boons of well-thought-out and complex software, for it contains powerful system tools. Among these features are a hierarchical file structure that is also distributed and a file-protection setup that offers the maximum flexibility in access to files while guaranteeing their integrity.

Important program-management tools include a routine that oversees the rewriting of software during development and another that automates the generation of a complete program from the most current modules.

The NDS II is the second step in the evolution of Intel's network architecture, iLNA [*Electronics*, Aug. 25, 1981, p. 120]. It connects Intellec development systems together so they can share large-capacity Winchester disk drives and a line printer located at the NRM. It will also serve as the basis for a whole new line of modular development system tools such as remote emulators, logic analyzers, and more.

Both the NRM and each work station can be connected directly to the Ethernet coaxial cable by a transceiver or by the Intellink communications module (Fig. 1). By itself, the Intellink module provides nine ports for interconnection, creating a local network of nine systems (eight work stations and one NRM). To another controller, the Intellink represents a segment of Ethernet cable that has nine transceivers already in place and working.

For networks with a radius of 50 meters or less, Intellink is a simple, low-cost alternative to installing Ethernet cabling and transceivers. Any work station can



1. Developing net. The NDS II brings existing Intel development systems, or work stations, into an Ethernet. A new network resource manager and the Intellink communications manager make management of distributed software development possible.

be installed by simply plugging a 50-m transceiver cable directly into the Intellink—a 5-second operation.

For expansion beyond nine systems or to a distance greater than a 50-m radius, the Intellink provides a built-in port for connecting the local cluster to Ethernet cable by means of a transceiver. Connection to the Ethernet allows communication with other work stations, NDS II networks, or other Ethernet-compatible devices that use the iLNA network architecture.

No matter which physical setup is chosen, each work station has independent access to, and can be directly accessed from, the Ethernet and the NDS II network. Each has a unique work-station identifier, distinguishing it from every other terminal in the world and ensuring correct communication between stations on the various local networks.

For multiple-net environments, each network can have a unique network identifier to allow their coexistence on one Ethernet. In a single net, the network identifier is not used, but its assignment ensures an orderly progression to a multi-net environment.

All current Intellec development systems can be upgraded to NDS II work stations. An upgrade consists of a communication-controller board set, software, and either 10- or 50-m cables.

The communication controller, a two-board set that plugs into any Intel Multibus chassis, provides many of the data- and physical-link functions of the six-layer standard reference model for open-systems interconnection (Fig. 2). The data-link functions performed are framing, link management, and error detection. Physical-link functions include preamble generation and decoding and bit encoding and decoding.

One board contains a 5-megahertz 8086 microprocessor with local random-access and read-only memory and interval timers, as well as direct-memory-access channels for sending and receiving data at 10 megabits per second. The second board contains bit-serial send-and-receive logic, packet address-recognition logic, and error-detection logic. The boards ensure that bad packets resulting from a collision are ignored.

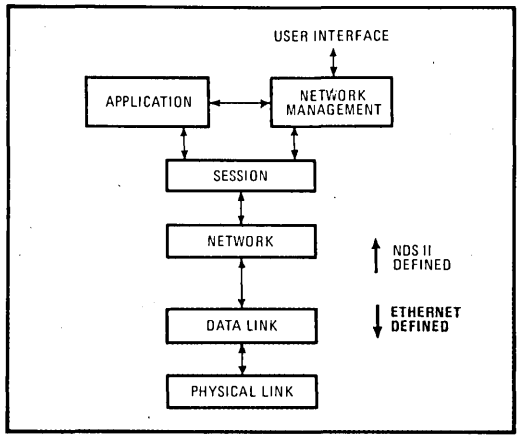
The NRM coordinates all the work stations' activities and manages file access to the shared disks. Initially, it will support one 8-inch 35-megabyte Winchester disk subsystem, as well as Intel cartridge-module disks. Multiple-disk support is in the wings, along with a larger 84-megabyte disk. It will be possible to attach six disks to one NRM, providing more than enough on-line shared storage for large program development and archiving. In addition, each work station can contain 2.5 megabytes of floppy-disk storage as a local resource.

Control contingent

The NRM (Fig. 3) comprises 13 Multibus slots, power supply, 8086-based system-processor board, input/output board based on the 8088 and 8089, 512-K-byte memory board with error checking and correction, two communication boards, and one 5¼-in. floppy-disk drive. The cabinet also has space for a cartridge-tape unit, expected to be delivered in mid-1982, which will give full intelligent archival backup for the Winchester disks housed in the attached cabinet.

To protect the integrity of the network, access to the NRM is restricted: a special supervisory terminal connected to the unit's serial port provides an interface with its commands and utilities. These facilities include system generation, intelligent archiving, and normal network maintenance such as the creation of any necessary user identifications.

The most important utility for system configuration is called Sysgen, an interactive routine designed to assist the supervisor, or project manager, in creating the NRM operating system. Sysgen makes it possible to create, modify, or delete system parameters, peripheral-devices configuration, and network configuration. It allows the project manager to tailor the network configuration on the fly in order to fit the changing needs of microprocessor development projects.



2. New layers. To the hardware layers of Ethernet, NDS II adds software layers that permit up to eight users to work together. The network layer need not be present if NDS II is not linked to the Ethernet, simplifying the operating system.

From the work-station perspective, the NRM is a remote file system. Each station functions as a stand-alone development system for all tasks not requiring NRM resources. When access to these resources is required, the user simply logs onto the network. The work station's resident operating system formats the appropriate file request, which the NRM processes interactively with other stations' demands.

The NRM operating system is multitasking, allowing a work station to access a file on the shared disk while other stations concurrently access other disk files. The interleaving of disk accesses, as well as the high-speed packet transmissions on the Ethernet, enables each work station to share equally the large file store—its being accessed by one user does not prevent other work stations from gaining access.

In an eight-station environment, the performance degradation due to network contention and the NRM operating system will be no more than 10%. This performance is one of the major reasons why distributed development systems provide a more cost-effective method for microprocessor development than time-shared systems; the former are much less susceptible to saturation under concurrent loading than are the latter.

Managing the work

To ensure efficient software development, high performance must be combined with tools to manage software complexity. For example, large software projects are often broken down into small tasks, and efficient file sharing becomes essential to project coordination. The shared-file system on NDS II is built on the RMX-86 volume-based hierarchy in which each user directory represents a node on a hierarchy of directories, commonly referred to as a hierarchial file system (Fig. 4).

Hierarchical file systems can contain a multitude of directories and data files. At the apex is the root volume, a conceptual file from which all directories emanate. The root volume contains all the volumes of the directories. Each volume can contain as many directories or files as available disk space will allow, and any directory may contain other directory files or data files. Each file (directory or data) can be traced through the hierarchy by its own path name. The NDS II hierarchical file system goes one step further by extending from the NRM to include the directories at the user's work station. When the user logs off the network, the only directories available are those on the work-station disks. When the user logs on, he or she gains access to the NRM system directories.

Thus each programmer has access to a common data base without the confusion of sifting through one massive directory. What's more, the structure keeps other users' files out of the way. In addition, it permits logically separate types of software within a user's directory. A programmer can create subdirectories to separate source files from object files, from backup files, and so on.

As a project's size increases, the number of directories and the complexity of path names in the system also increases. To simplify the task of accessing any particular directory, the user can assign a less cumbersome name—what amounts to a macroinstruction. Then, the user simply types in this macroname. Maximum flexibility is maintained, as each programmer can assign macronames to any directory.

An added benefit from macroname assignment is device transparency: the user concerns himself only with directories, irrespective of physical location. Physical devices are fixed in size and location, as opposed to directories, which can be adjusted to organize the contents in an optimal fashion.

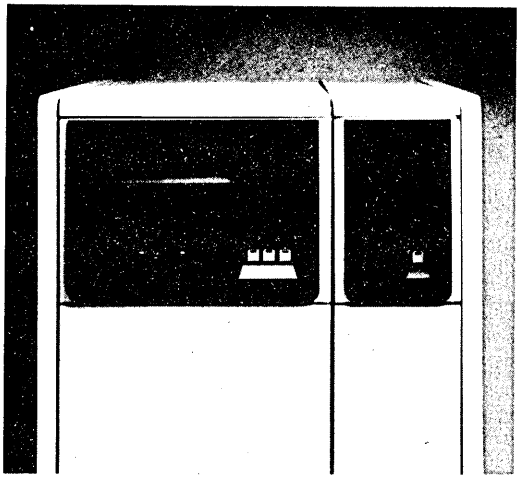
File protection

Before accessing the network, each user must be identified to the NRM through a log-on procedure. This setup establishes a unique user identification that is subsequently used to control access to files and directories in the hierarchical file system. Each directory and data file has specific "owner" and "world" access rights, which protect against accidental modification or deletion.

A file has three possible access rights for both the owner and the world: read, write, and delete. A directory also has three similar access rights for both the owner and the world: list a directory, add a directory entry, and delete a directory entry.

The access rights in file systems improve coordination during software development by allowing complete modules that have been tested and debugged in a user's work space to be converted into read status for the world. Then these modules can be integrated and tested with other independently developed software modules. Thus modules declared as read-only are guaranteed to be the most current debugged versions, and a common data base of completed modules is ensured.

Extended to multiple-project environments, the file system can provide logically separate work spaces for each project group. Specific directories can be set aside for complete modules for various projects. Each user can develop portions of the program in a private work space with guaranteed file protection and can use the public files (or directories) for integration and testing of the



3. Manager. The network resource manager (NRM) in the cabinet's left side governs access to the 35-megabyte Winchester drive on the right. Access to network-managing software is gained only through a supervisory terminal attached directly to the NRM.

module under development. Commonly used utilities and compilers can be accessible in a specific directory as public files (read-only for world access) to eliminate the necessity of redundant files at each work station. As a result, all programmers can proceed without fear of inadvertent modification of private files either by others or by themselves.

As well as managing communications between shared disks and work stations, the NRM maximizes the use of all network resources with distributed job control. DJC allows the user of any work station to export a batch job to the NRM for remote execution.

To accomplish this, the NRM classifies each work station into one of two groups—private and public. It keeps track of the public work stations and uses them to execute the queue of batch-type jobs. A user can declare any work station as public: available for use by the NRM for remote execution. Also, a programmer can send a job to a specific queue at the NRM by using the export command. The NRM executes the job on a public work station and return the results to the user directory.

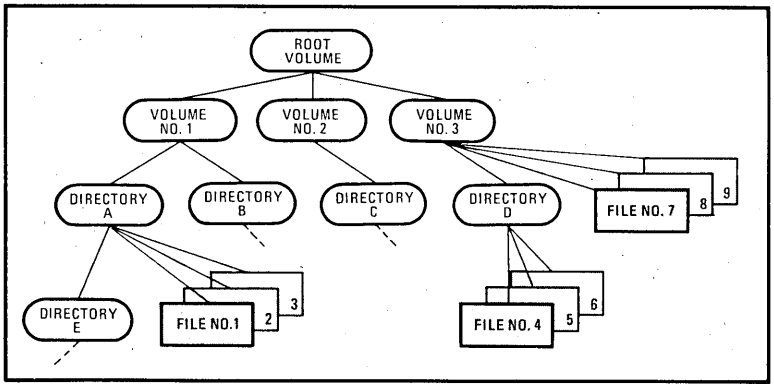
With DJC, the resources of the entire network can be shared to maximum advantage. A typical project involves program-module editing and debugging at Intellec series II or model 800 work stations, while a 8086-based Intellec series III unit can provide a host execution environment to compile completed modules quickly. DJC allows the user to export the compilation process to the high-performance series III work station, then return immediately to other tasks while the NRM oversees the compilation. At any time, the users can check on job status or queue status by typing a command from their work stations.

New work stations

Currently, Intellec development systems provide a single-task environment and therefore can be declared public to the NRM as users finish on-line work. Later this year, Intel will introduce high-performance work stations with foreground-background capability to allow a user to run a job in the foreground while making the background public so that jobs exported by other programmers can be executed through DJC. Foreground-background capability with DJC will effectively double the usefulness of the work station and substantially cut the cost of development time.

In-house benchmark tests indicate that the performance of each work station connected to the NRM is much improved. For example, a compilation executed with all file requests from the NRM hard disk is twice as fast as requesting files from the work station's floppy disk. Each station enjoys hard-disk performance during compilation, assembly, and any file manipulation—at a fraction of the cost of a dedicated disk system.

User's tools also speed program development, as well as make management easier. The most important programmer tools on NDS II are SVCS (software-version control system) and MAKE, an automatic software-generation tool. They provide a superset of the functions offered by the SVCS and MAKE found in the Unix programmers workbench.



4. Climbing an inverted tree. To find a file in the NDS II, the user first goes to the root volume of this hierarchical file structure. From that volume, he or she can go to the project volume assigned by the project manager and access other directories or files that have been declared accessible.

SVCS controls and documents changes to software products, handling both source and object files. It contains facilities for storing and retrieving different versions of a given program module, for controlling update privileges, and for recording who made what changes, when, and why.

Documentation of module status and of the levels, or versions, involved is the key factor determining the success of program development by group effort. Valu-

MAKEing it easy to revise programs

NDS-II's MAKE facility is a development tool for both generation and documentation of a software system. Suppose, for example, a software system called PGM.86 consists of three separate programs linked together, and, for simplicity, that each program consists of only one compiled source file, rather than a subsystem of multiple files. This relationship forms a dependency that would be graphed by the user as in the figure below.

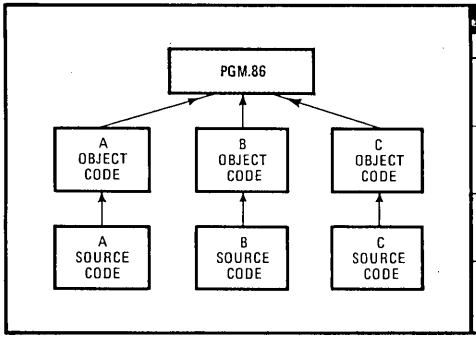
With the MAKE facility, a user can create an automated-generation procedure for the system PGM.86 that checks the currency of each subprogram. A MAKE command file that does so is illustrated in the accompanying table.

When the command file is invoked, the commands it contains are executed in top-down fashion. In step 1 of the table, the facility first checks if the PGM.86 is older (represented by the greater-than sign) than any of its dependent object-code modules. The facility checks and compares the date and time stamp of each module with that of PGM.86. Date and time stamps are updated automatically whenever a file is modified.

If any of the object modules are newer versions, then MAKE is instructed to link together the latest versions of the object modules to form the latest version of the software system. Before executing the link routine, the MAKE facility must first check to see if any of the object files are older than the related source files given in the dependency graph, as shown in steps 2, 3, and 4.

The MAKE facility goes through each step and executes the specified task only if the specified condition is true. Once the dependency graph is created, the MAKE facility can quickly and automatically generate the latest version of a software system under development even when source files change frequently.

The MAKE facility removes much of the guesswork surrounding software-system generation by ensuring the latest versions of source code is incorporated into the final software system. The dependency graph in its current form can also be printed by NDS II to document the software-system construction without having to keep an out-of-date sketch taped to the laboratory wall.



MAKE PROGRAM FOR PGM.86

Steps	Statements
1	IF PGM.86 > A.OBJ, B.OBJ, C.OBJ THEN RUN LINK86 A.OBJ, B.OBJ, C.OBJ TO PGM.86 END
2	IF A.OBJ > A.SRC THEN RUN PLM86 A.SRC END
3	IF B.OBJ > B.SRC THEN RUN PLM86 B.SRC END
4	IF C.OBJ > C.SRC THEN RUN ASM86 C.SRC END

able development time can be lost trying to work someone else's modified modules if documentation specifying what, where, when, and why changes were made is not available. In fact, as programs become more complicated, even the module writer may not exactly remember the history of the module.

Automatic documentation

SVCS provides a tool for automatic documentation of these facts. When a new module is created, it is set to level 1. All subsequent versions of the module are maintained with in a single file. Changes to the module are stored as "deltas" to the original. SVCS automatically records what changes were made and when they were made, and it requires the modifier to specify a reason for the change. The project manager may create a software checkpoint at any time by declaring the module as the next release level; subsequent deltas will then be applied to only this new release level.

Other capabilities in SVCS also increase project control. Restrictions may be placed on who is allowed to make changes to which modules and at which levels. An identification facility is also included, allowing the system to stamp modules containing object code with version information. From this information alone, a user can determine the level of source code used to generate the object module and thereby determine exactly which level of software is current and which level is being executed. To aid support groups in future maintenance of the program, any level of a software system can be regenerated from the original modules.

The second important program management tool on NDS-II is called MAKE, (see "MAKEing it easy to revise programs," above). When MAKE is invoked, a software system is automatically generated from the most current version of specific modules delineated by a dependency graph. MAKE ensures that the software generation is current and correct, while recompiling only program modules that need to be updated. To coincide with the concept of modular program development, any component of a MAKE could invoke another MAKE to generate a lower-level component such as a library. □