Chapter 29

The Tandem 16: A Fault-Tolerant Computing System[1]

James A. Katzman

Summary A fault-tolerant computer architecture is examined that is commercially available today and installed in many industries. The hardware is examined in this paper and the software is examined in the companion paper.

Introduction

The increasing need for businesses to go on-line is stimulating a requirement for cost effective computer systems having continuous availability [Katzman, 1977; Tandem, 1976]. Certain applications such as automatic toll billing for telephone systems lose money each minute the system is down, and the losses are irrecoverable. Systems commercially available today have met a necessary requirement of multiprocessing but not the sufficient conditions for fault-tolerant computing.

The greatest dollar volumes spent on systems needing these fault-tolerant capabilities are in the commercial on-line, data base transaction, and terminal oriented applications. The design of the Tandem 16 NonStop[2] system was directed toward offering the commercial market an off-the-shelf, general purpose system with at least an order of magnitude better availability than existing off-the-shelf systems without charging a premium. This was accomplished by using a top down system design approach, thus avoiding the shortcomings of the systems currently addressing the fault-tolerant market.

Except for some very expensive special systems developed by the military, universities, and some computer manufacturers in limited quantities, no commercially available systems have been designed for continous availability. Some systems such as the ones designed by ROLM have been designed for high MTBF by "ruggedizing," but typically computers have been designed to be in a monolithic, single processor environment. As certain applications demanded continuous availability, manufacturers recognized that a multiprocessor system was necessary to meet the demands for availability. In order to preserve previous development effort and compatibility, manufacturers invented awkward devices such as I/O channel switches and interprocessor communication adapters to retrofit existing hardware. The basic flaw in this effort is that only multiprocessing was achieved. While that is necessary for continuously available systems, it is far from sufficient.

Single points of failure flourish in these past architectures (Fig. 1). A power supply failure in the I/O bus switch or a single integrated circuit (IC) package failure in any I/O controller on the I/O channel emanating from the I/O bus switch will cause the entire system to fail. Other architectures have used a common memory for interprocessor communications, creating another single point of failure. Typically such systems have not even approached the problem of on-line maintenance, redundant cooling, or a power distribution system that allows for brownout conditions. In today's marketplace, many of the applications of fault-tolerant systems do not allow any down time for repair.

Expansion of a system such as the one in Fig. 1 is prohibitively expensive. A three processor system, strongly connected in a redundant fashion, would require twelve interprocessor links on the I/O channels; five processors would need forty links; for n processors, 2n(n-1) links are required. These links often consist of 100–200 IC packages and require entire circuit boards priced between $6,000 and $10,000 each. Using the I/O channel in this manner limits the I/O capabilities as a further undesirable side effect. The resulting hardware changes for expansion, if undertaken, are typically dwarfed in magnitude by the software changes

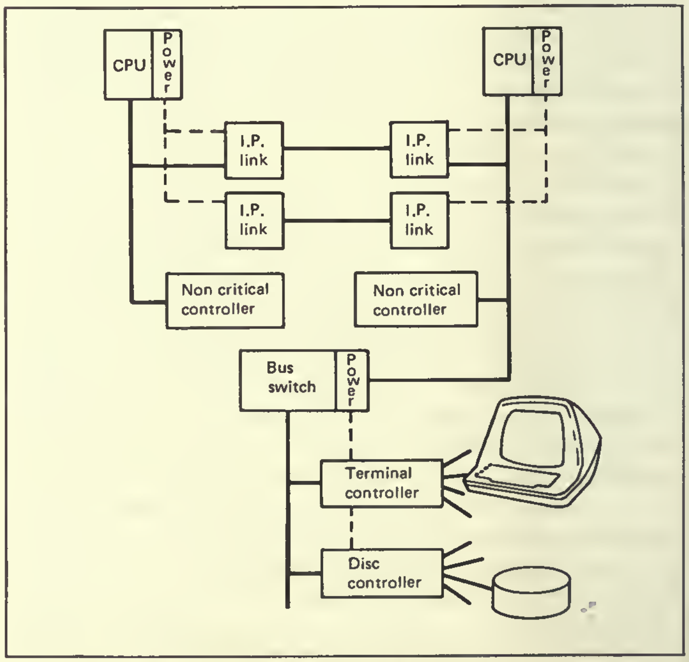

Fig. 1. Example of previous fault-tolerant systems.

[1]© 1977. Reprinted with the express permission of Tandem Computers Inc., Cupertino, Calif.
[2]NonStop is a trademark of Tandem Computers.

needed when applications are to be geographically changed or expanded.

This paper describes the Tandem 16 architecture at the lowest level (the hardware). Section 1 deals with the overall system organization and packaging. Section 2 explains the processor module organization and its attachment to the interprocessor communications system. Section 3 discusses the I/O system organization. Section 4 discusses power, packaging, and on-line maintenance aspects that are not covered elsewhere in the paper.

1. System Organization

The Tandem 16 NonStop system is organized around three basic elements: the processor module, dual-ported I/O controllers, and the DC power distribution system (Figs. 2 and 3). The processors are interconnected by a dual-interprocessor bus system: the Dynabus; the I/O controllers are each connected with two independent I/O channels, one to each port; and the power distribution system is integrated with the modular packaging of the system.

The system design goal is two-fold: (1) to continue operation of the system through any single failure, and (2) to be able to repair that failure without affecting the rest of the system. The on-line maintenance aspects were a key factor in the design of the physical packaging and the power-distribution of the system.

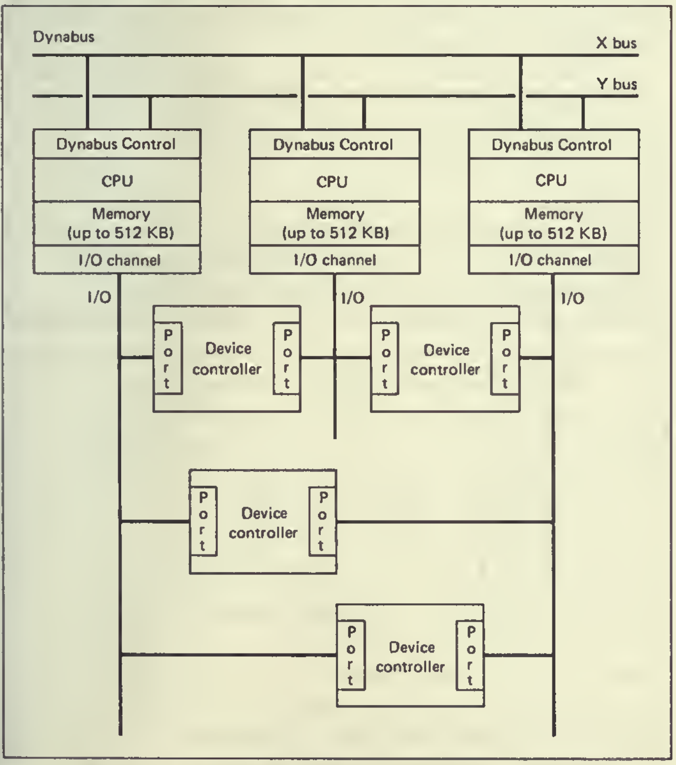

Fig. 2. Tandem 16 system architecture.

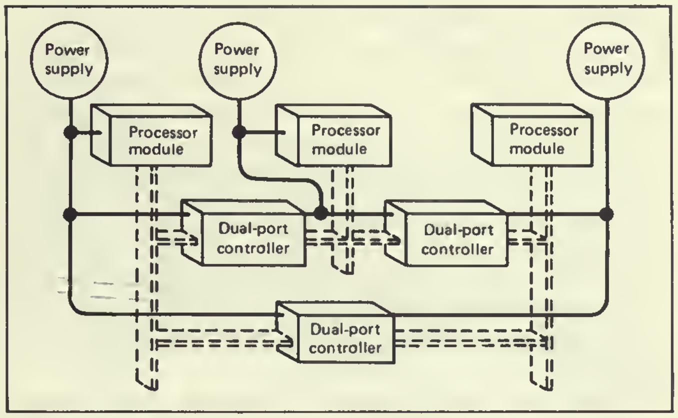

Fig. 3. Power distribution.

System Packaging

The cabinet (Fig. 4) is divided into 4 sections: the upper card cage, the lower card cage, cooling, and power supplies. The upper card cage contains up to 4 processors, each with up to 512K bytes of independent main memory. The lower card cage contains up to 32 I/O Controller printed circuit (PC) cards, where each controller consists of one to three PC cards. The cooling section consists of 4 fans and a plenum chamber that forces laminar air flow through the card cages. The power supply section contains up to 4 power supply modules. Multiple cabinets may be bolted together, and the system has the capability to accommodate a maximum of 16 processors.

Each processor module, consisting of CPU, memory, Dynabus control and I/O channel, is powered by an associated power

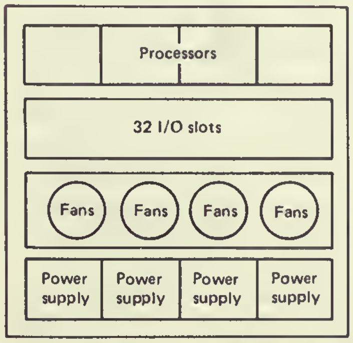

Fig. 4. Tandem 16 physical cabinet.

supply. If a failed module is to be replaced in this section its associated power supply is shut off, the module is replaced, and the power supply is turned on. Each card cage slot in the I/O card cage is powered by two different power supplies. Each of the I/O controllers is connected via its dual-port arrangement to two processors. Each of those processors has its own power supply; usually, but not necessarily, those two supplies are the ones that power the I/O controller (Fig. 3). Each slot in the I/O card cage can be powered down by a corresponding switch disconnecting power from the slot from both supplies without affecting power to the remainder of the system. Therefore, if a power supply fails, or if one is shut down to repair a processor, no I/O controllers are affected.

The dual-power sourcing to the I/O controllers was originally designed using relay switching. This plan was abandoned for several reasons: (a) to contend with relay failure modes is difficult; (b) the number of contact bounces on a switch-over is neither uniform nor predictable, making it difficult for the operating system to handle power-on interrupts from the I/O controllers; and (c) during the switch-over, controllers do lose power, and while most controllers are software-restartable, communications controllers hang up their communications lines. We therefore devised a diode current sharing scheme whereby I/O controllers are constantly drawing current from two supplies simultaneously. If a power supply fails, all the current for a given controller is supplied by the second power supply. There is also circuitry to provide for a controlled ramping of current draw on turn-on and turn-off so there are no instantaneous power demands from a given supply causing a potential momentary dip in supply voltage.

Both fans and power supplies are electrically connected using quick disconnect connectors to speed replacement upon failure. No tools are required to replace a power supply. A screwdriver is all that is needed to replace a fan. Both replacements take less than 5 minutes.

Interconnections

Physical interconnection is done both using front edge connectors and back-planes. Communication within a processor module (e.g., between the CPU and main memory) takes place over four 50 pin front edge connectors using flat ribbon cable. Interprocessor communication takes place over the Dynabus on the back-plane also utilizing ribbon cable. The I/O controllers use etch trace on the back-plane for communication among PC cards of a multicard controller. The I/O channels are back-plane ribbon cable connections between the processors and the I/O controllers.

Peripheral I/O devices are connected via shielded round cable either to a bulk-head patch panel or directly to the front edge connectors of the I/O controllers. If a patch panel is used, then there is a connection using round cables between the patch panel and the front edge connectors of the I/O controllers.

Power is distributed using a DC power distribution scheme. Physically, AC is brought in through a filtering and phase splitting distribution box. Pigtails connect the AC distribution box to one of the input connectors of a power supply. The DC power from the supply is routed through a cable harness to a laminated bus bar arrangement which distributes power on the back-planes to both processors and I/O controllers.

2. Processor Module Organization

The processor (Fig. 5) includes a 16 bit CPU, main memory, the Dynabus interface control and an I/O channel. Physically the CPU, I/O channel and Dynabus control consists of two PC boards 16 inches by 18 inches, each containing approximately 300 IC packages. Schottky TTL circuitry is used. Up to 512K bytes of main memory is available utilizing core or semiconductor technology. Core memory boards hold 32K 17-bit words, and each occupies two card slots because of the height of the core stack. Semiconductor memory is currently implemented utilizing 16 pin, 4K dynamic RAMs. These memory boards contain 48K 22-bit words per board and occupy only one card slot and are therefore three times denser than core.

The processor module is viewed by the user as a 16-bit, stack-oriented processor, with a demand paging, virtual memory system capable of supporting multiprogramming.

The CPU

The CPU is a microprogrammed processor consisting of a bank of 8 registers which can be used as general purpose registers, as a LIFO register stack, or for indexing; an ALU; a shifter; two memory stack management registers; program control registers

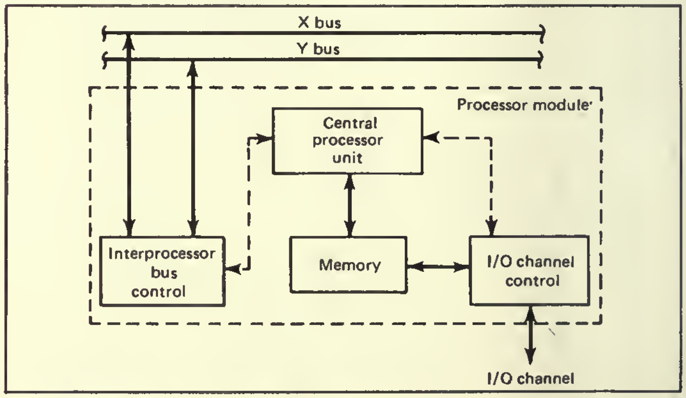

Fig. 5. Tandem 16 processor organization.

(e.g., program counter, instruction register, environment or status register, and a next instruction register for instruction prefetching); scratch pad registers available only to the microprogrammer; and several other miscellaneous flags and counters for the microprogrammer.

The microprogram is stored in read-only memory and is organized in 512-word sectors of 32-bit words. The microinstruction has different formats for branching, sequential functions, and immediate operand operations. The Tandem 16 instruction set occupies 512 words with the decimal arithmetic option occupying another 512 words. The address space for the microprogram is 2K words.

The microprocessor has a 100 ns cycle time and is a two stage pipelined microprocessor; i.e., all microinstructions take two cycles to execute but one completes each cycle. In the first stage of the pipeline any two operands are selected by two source fields in the microinstruction for loading into the ALU input registers. In the second stage of the pipeline the ALU performs a primitive operation on the operands placed in the ALU input registers during the previous cycle and performs a shift operation on the results. In parallel, a miscellaneous operation such as a condition code setting or a counter increment can be done, the result can be stored in any CPU register or dispatched to the memory system or I/O channel, and a condition test made on the results. Each of these parallel operations is controlled by a separate control field in the microinstruction.

The basic set of 123 machine instructions includes arithmetic operations (add, subtract, etc.), logical operations (and, or, exclusive or), bit deposit, block (multiple element) moves/compares/scans, procedure calls and exits, interprocessor SENDs, and I/O operations. All instructions are 16 bits in length. The decimal instruction set provides an additional 20 instructions dealing with four-word operands.

The interrupt system has 16 major interrupt levels which include interprocessor bus data received, I/O transfer completion, memory error, interval timer, page fault, privileged instruction violation, etc.

Provision is made for several events to cause microinterrupts. They are entirely handled by the CPU's microprocessor without causing an interrupt to the operating system. One event, for example, is the receipt of a 16 word packet over the Dynabus. A packet is the primitive unit of data which is transferred over the Dynabus for interprocessor communication. The microprocessor puts the information in a predetermined area of memory and does not cause a system interrupt until the entire message is received.

The register stack is used for most arithmetic operations and for holding parameters for block instructions (moves/compares/scans) which need the parameters updated dynamically so that the instructions may be interruptable and restarted. The 8-register stack is a "wraparound" stack and is not logically connected to the memory stack.

Main Memory

Main memory is organized in physical pages of 1K words of 16 bits/word. Up to 256K words of memory may be attached to a processor. In the core memory systems there is a parity bit for single error detection, and in semiconductor memory systems there are 6 check bits/word to provide single error correction and double error detection. Due to the relative reliability of these two technologies, we have found that semiconductor memory, without error correction, is much less reliable than core, and that with error correction, it is somewhat more reliable than core. Battery backup provides short term non-volatility to the semiconductor memory system for utility power outage considerations.

It might be noted that there are some memory systems using a 21 bit error correction scheme (5 check bits on a 16 bit data word instead of 6). While 5 bits are enough to correct all single bit errors, it does not detect approximately ⅓ of the possible double bit error combinations. In these conditions, this 5 check bit scheme will incorrectly deduce that some bit (neither of the bits actually in error) is incorrect and correctable. The scheme will then correct this bit (actually causing 3 bits to be in error), and deliver it to the system as "good" reporting a correctable memory error.

Memory is logically divided into 4 address spaces (Fig. 6). These are the virtual address spaces of the machine; both the system and the user have a code space and a data space. The code space is unmodifiable and the data space can be viewed either as a stack or a random access memory, depending on the addresssing mode used. Each of these virtual address spaces is 64K words long, and is addressed by a 16 bit virtual address.

The physical memory address is 18 bits with conversion from the virtual address to physical address accomplished through a mapping scheme. Four maps are provided, one for each logical address space; each map consists of 64 entries one for each page in the virtual address space. The maps are implemented in 50 ns access bipolar static RAM. The map access and main memory error correction is included in the 500 ns cycle time for semiconductor memory systems.

The unmodifiable code area provides reentrant, recursive, and sharable code. The data space (Fig. 7) can be referenced relative to address 0 (global data or G+ addressing), or relative to the memory stack management registers in the CPU.

The lowest level language provided on the Tandem 16 system is T/TAL, a high-level, block-structured, ALGOL-like language which provides structures to get at the more efficient machine instructions. The basic program unit in T/TAL is the PROCEDURE. Unlike ALGOL, there is no outer block, but rather a main

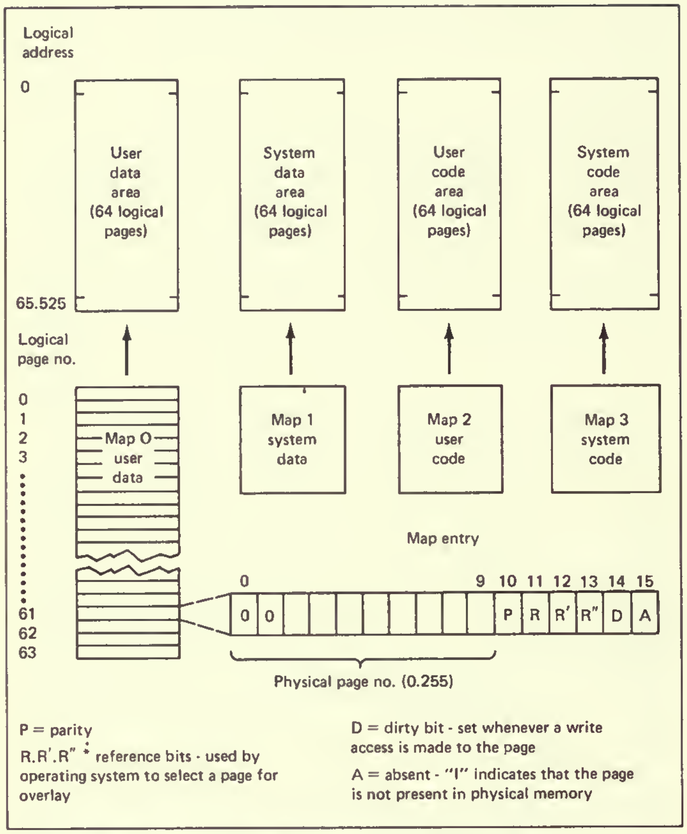
Logical address
0
65.525
User data area (64 logical pages)
System data area (64 logical pages)
User code area (64 logical pages)
System code area (64 logical pages)
Logical page no.
0 1 2 3 61 62 63
Map 0 user data
Map 1 system data
Map 2 user code
Map 3 system code
Map entry
0 9 10 11 12 13 14 15
0 0 P R R' R" D A
Physical page no. (0.255)
P = parity
R.R'.R" = reference bits - used by operating system to select a page for overlay
D = dirty bit - set whenever a write access is made to the page
A = absent - "I" indicates that the page is not present in physical memory

Fig. 6. Tandem 16 logical memory address spaces.

PROCEDURE. T/TAL has the ability to declare certain variables as global. PROCEDURES cannot be nested in T/TAL, but a SUBPROCEDURE can be nested in a PROCEDURE and only in a PROCEDURE. A SUBPROCEDURE is limited in local variable access capabilities.

The memory stack, defined by two registers in the CPU, is used for efficient linkage to and from procedures, parameter passing, and dynamic storage allocation and deallocation for variables local to the procedure.

The L register (Local variables) points to the last stack marker placed on the stack. This marker contains return information about the caller such as the return address and the previous location of the L register. The contents of the L register are primarily changed by the procedure call and exit instructions.

Addressing relative to the L register provides access to parameters passed to a procedure (L−) and local variables of the procedure (L+). Parameters may be passed either by value (using direct addressing) or by reference (using indirect addressing).

The S register (stack top pointer) points to the last element placed on the stack. It is used for a SUBPROCEDURE's sublocal data area when S relative addressing (S−) is used.

There is a special mode of addressing used by the operating system, called System Global (SG+) addressing. It is used by the operating system while it is working in a user's virtual data space (on his behalf) and needs to address the system data space. The system data space contains many resource tables and buffers and the need to access them quickly justifies the existence of this addressing mode.

There are three tables known to the operating system, the microprogram and the hardware: the system interrupt vector (SIV), the I/O Control (IOC) table, and the Bus Receive Table (BRT). These tables will be explained in later sections as appropriate.

The Dynabus

The Dynabus is a set of two independent interprocessor buses. Bus access is determined by two independent interprocessor bus controllers. Each of these controllers is dual-powered, in the same manner as an I/O controller. The Dynabus controllers are very small, approximately 30 IC packages, and are not associated with, nor physically a part of, any processor. Each bus has a two byte data path and control lines associated with it. There are two sets of radial connections from each interprocessor bus controller to each processor module. They distribute clocks for synchronous transmission over the bus and for transmission enable. Therefore, no failed processor can independently dominate Dynabus utilization upon failure since in order to electrically transmit onto the bus, the bus controller must agree that a given processor has the right to transmit. Each bus has a clock associated with it, running independently of the processor clocks and located on the associated bus controller. The clock rate is 150 ns on two to eight processor systems. The clock does need to be slowed down for the longer interprocessor buses of greater than eight processors. Therefore each bus on small systems transfers at the rate of 13.3M bytes/second and on the larger systems at 10M bytes/second. Performance measurements have shown that under worst case test conditions the Dynabus is only 15% utilized in a ten processor system.

Each processor in the system attaches to both interprocessor buses. The Dynabus interface control section (Fig. 8) consists of 3 high speed caches: an incoming queue associated with each interprocessor bus, and a single outgoing queue that can be switched to either of the buses. All caches are 16 words in length and all bus transfers are cache to cache. All components that attach to either of the buses are kept physically distinct, so that no single component failure can contaminate both buses simultaneously. Also in this section are clock synchronization and interlock circuitry. All processors communicate in a point to point manner

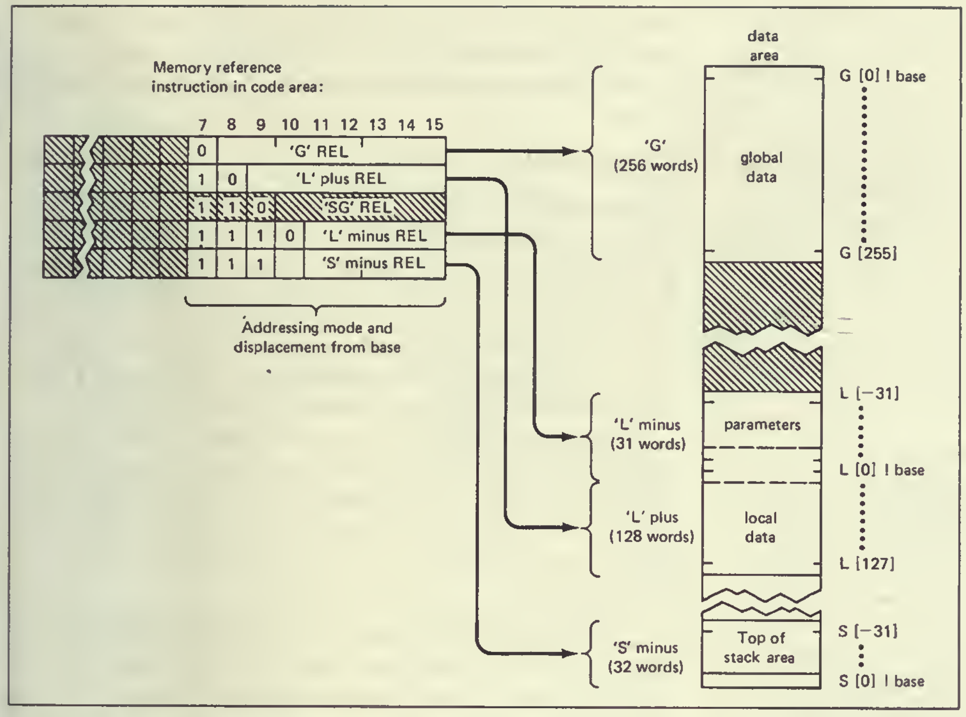

Fig. 7. Tandem 16 data space.

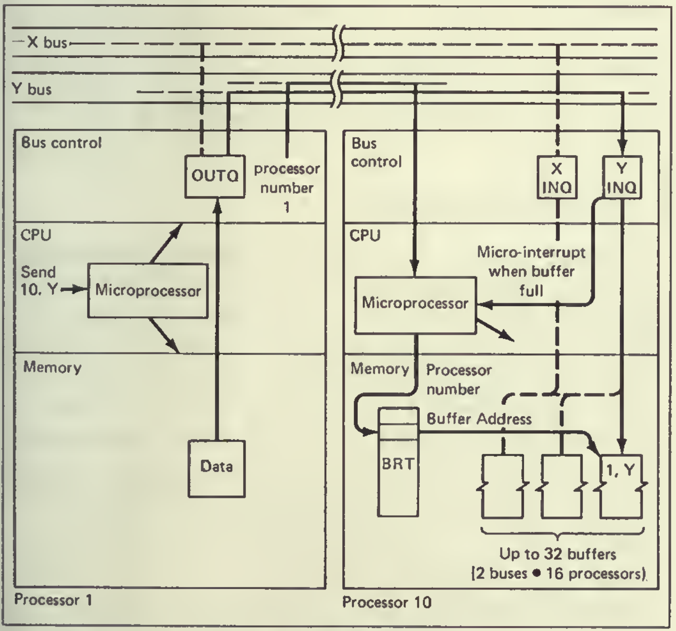

Fig. 8. Tandem 16 dynabus interface and control.

using this redundant direct shared bus (DSB) configuration [Anderson and Jensen, 1975].

For any given interprocessor data transfer, one processor is the sender and the other the receiver. Before a processor can receive data over an interprocessor bus, the operating system must configure an entry in a table (Fig. 9) known as the Bus Receive Table (BRT). Each BRT entry contains the address where the incoming data is to be stored and the number of words expected. To transfer data over a bus, a SEND instruction is executed in the sending processor, which specifies the bus to be used, the intended receiver, and the number of words to be sent. The sending processor's CPU stays in the SEND instruction until the data transfer is completed. Up to 65,535 words can be sent in a single SEND instruction. While the sending processor is executing the SEND instruction, the Dynabus interface control logic in the receiving processor is storing the data away according to the appropriate BRT entry. In the receiving processor this occurs simultaneously with program execution.

The message is divided into packets of 15 information words and an LRC check word. The sending processor first fills its outgoing queue with these packets, requests a bus transfer, and transmits upon grant of the bus by the interprocessor bus controller. The receiving processor fills the incoming queue associated with the

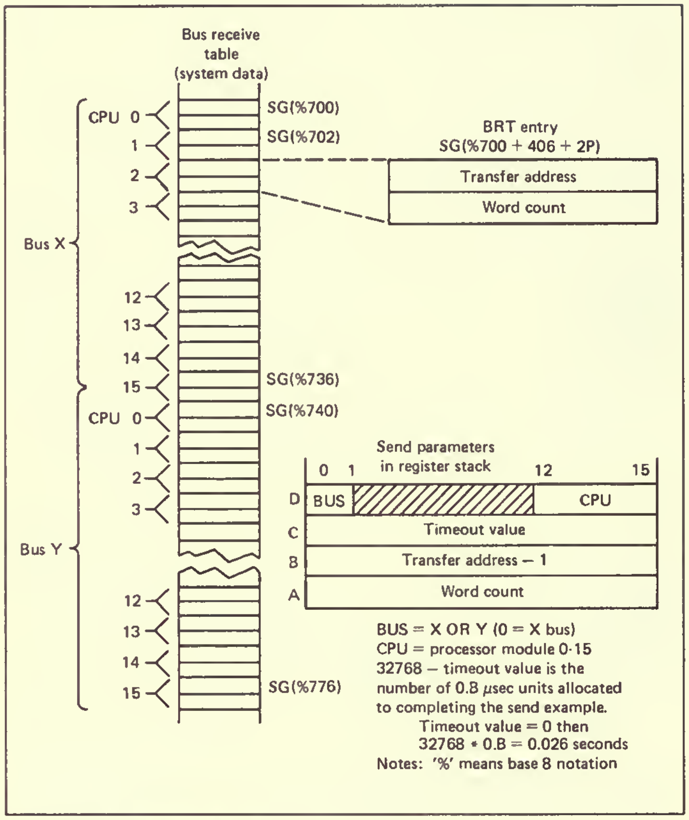
Bus receive table (system data)
CPU 0
1
2
3
12
13
14
15
CPU 0
1
2
3
12
13
14
15
Bus X
Bus Y
SG(%700)
SG(%702)
SG(%736)
SG(%740)
SG(%776)
BRT entry SG(%700 + 406 + 2P)
Transfer address
Word count
Send parameters in register stack
0 1 12 15
D BUS CPU
C Timeout value
B Transfer address − 1
A Word count
BUS = X OR Y (0 = X bus)
CPU = processor module 0·15
32768 – timeout value is the number of 0.8 μsec units allocated to completing the send example.
Timeout value = 0 then 32768 * 0.8 = 0.026 seconds
Notes: '%' means base 8 notation

Fig. 9. Bus receive table.

bus over which the packet is received, and issues a microinterrupt to its own CPU. The microprocessor of the CPU checks the BRT entry, stores the packet away, verifies the LRC check word, and updates the BRT entry accordingly. If the count is exhausted the currently executing program is interrupted; otherwise program execution continues.

The BRT entries are two words that include a transfer count and buffer address. The SEND instruction has as parameters the designation of the bus to be used, the intended receiver, the data buffer address in the system data space, the word count to be transferred, and a timeout value. Error recovery action is to be taken in case the transfer is not completed within the timeout interval. These parameters are placed on the register stack and are dynamically updated so that the SEND instruction is interruptible on packet boundaries.

There are several levels of protocol, beyond the scope of this paper, dealing with the interprocessor bus that exist in software [Bartlett, 1978], to assure that valid data is transferred. The philosophy for the hardware/software partitioning was to leave the more esoteric decisions to the software, e.g., alternate path routing, and error recovery procedures, with fault detection and reporting implemented in the hardware. Fault detection was designed in those areas having the highest anticipated probability of error.

The Input/Output Channel

The heart of the Tandem 16 I/O System is the I/O channel. All I/O is done on a direct memory access (DMA) basis. The channel is a microprogrammed, block multiplexed channel with the block size determined by the individual controllers. All the controllers are buffered to some degree so that all transfers over the I/O channel are at memory speed (4M bytes/second) and never wait for mechanical motion since the transfers always come from a buffer in the controller rather than from the actual I/O device.

There exists a table in the system data space of each processor called the IOC (I/O Control) table that contains a two word entry (Fig. 10) for each of the 256 possible I/O devices attached to the I/O channel. These entries contain a byte count and virtual address in the system data space for data transfers from the I/O system.

The I/O channel moves the IOC entry to active registers during connection of an I/O controller and restores the updated values to the IOC upon a disconnection. The I/O channel alerts the I/O controller when that count has been exhausted and that causes the controller to interrupt the processor.

The channel does not execute channel programs as on many systems but it does do data transfer in parallel with program execution. The memory system priority always permits I/O accesses to be handled before CPU or Dynabus accesses (in an on-line, transaction oriented environment, it is rare that a system is not I/O bound). The maximum I/O transfer is 4K bytes.

3. I/O System Organization

The I/O system had a design goal of being very efficient in a transaction, on-line oriented environment. This environment has constraints different from those of a batch environment. The figure of merit in an on-line system is the number of transactions/second/dollar that can be handled by the system. We also wanted an I/O system that had low overhead, fast transfer rates, no overruns, and no interrupts to the system until a logical entity of work was completed (i.e., no character by character interrupts from the terminals). The resulting design satisfied these goals by implementing an I/O system that was extremely simple.

I/O controllers reconnect to the channel when their buffers are stressed past a configurable threshold, transfer data in a burst mode until their buffer stress is zero (buffer empty on input operations, full on output operations), and disconnect from the channel. When the transfer terminates, the I/O controller interrupts the processor. Controllers may interrupt for other reasons

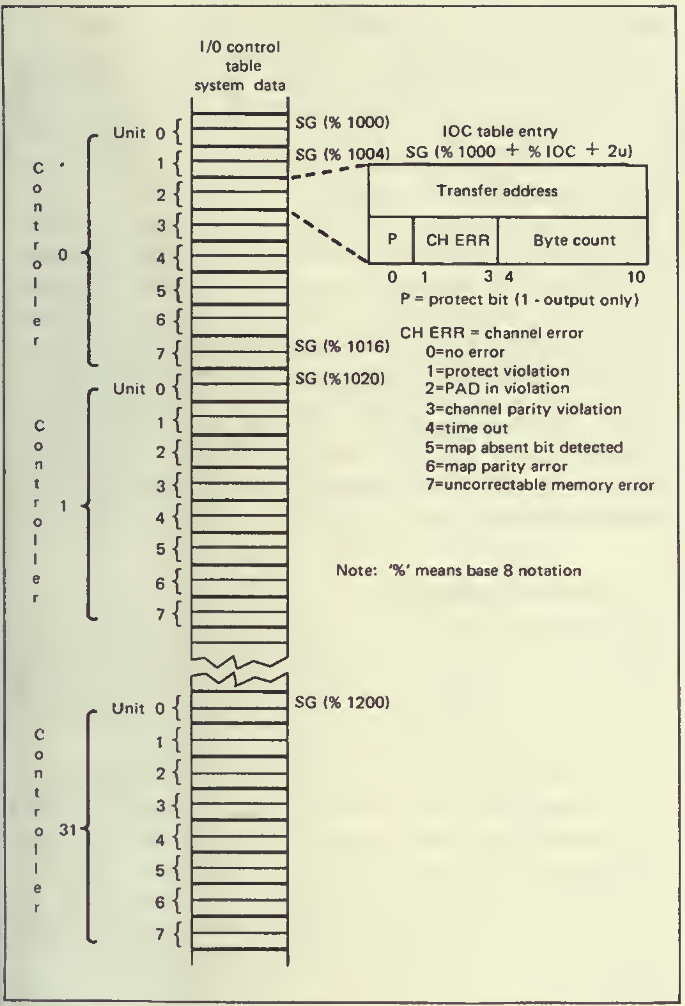

Fig. 10. I/O control table.

than an exhausted byte count, e.g., a terminal controller receiving an end-of-page character from a page mode terminal, or I/O channel error condition, or a disc pack being mounted.

Dual-Port Controllers

The dual-ported I/O device controllers provide the interface between the Tandem 16 standard I/O channel and a variety of peripheral devices using distinct interfaces. While the I/O controllers are vastly different, there is a commonality among them that folds them into the Tandem 16 NonStop architecture.

Each controller contains two independent I/O channel ports implemented by IC packages which are physically separate from each other so that no interface chip can simultaneously cause failure of both ports. Each port of each controller has a 5-bit configurable controller number, and interrupt priority setting. These settings can be different on each port. The only requirement is that each port attached to an I/O channel must be assigned a controller number and priority distinct from controller numbers and priorities of other ports attached to the same I/O channel.

Each controller has a PON (power-on) circuit which clamps its output to ground whenever the controller's DC supply voltage is not within regulation. The PON circuit has hysteresis in it so that it will not oscillate if the power should hover near the limit of regulation. When the power is within regulation, the output of the PON circuit is at a TTL "I" level. A power-on condition causes a controller reset and also gives an interrupt to one of the two processors to which it is attached. The output of the PON circuit is also used to enable all the I/O channel bus transceivers so that a controller being powered down will not cause interference on the I/O channels during the power transient. This is possible because the PON circuit operates with the supply voltage as low as .2 volts and special transceivers are used which correctly stay in a high impedance state as long as the control enable is at a logical "0."

Logically only one of the two ports of an I/O controller is active and the other port is utilized only in the event of a path failure to the primary port. There is an "ownership" bit (Fig. 11) indicating to each port if it is the primary port or the alternate. Ownership is changed only by the operating system issuing a TAKE OWNERSHIP I/O command. Executing this special command causes the I/O controller to swap its primary and alternate port designation and to do a controller reset. Any attempt to use a controller which is not owned by a given processor will result in an ownership violation. If a processor determines that a given controller is malfunctioning on its I/O channel, it can issue a DISABLE PORT command that logically disconnects the port from that I/O controller. This does not affect the ownership status. That way, if the problem is within the port, the alternate path can be used, but if the problem is in the common portion of the controller, ownership is not forced upon the other processor.

A controller signals an interrupt on the I/O channel if the channel has indicated an exhausted transfer count, if the controller terminates the transfer prematurely, or for attention purposes.

When simultaneous interrupts occur on an I/O channel, a priority scheme determines which interrupt is handled first. There are two levels of priorities, designated "rank 0" and "rank I." Each rank has up to 16 controllers assigned to it. Jumper wires on each controller determine the rank and position within the rank (positions 0 to 15). The I/O channel issues a rank 0 interrupt poll cycle and each controller assigned to rank 0 can place an interrupt request, if it needs service, on a dedicated data bit of the I/O channel determined by the jumper wires. If there are no controllers on rank 0 requiring service, the I/O channel issues the interrupt poll cycle for rank I. Note, only 32 controllers can be assigned to a given channel and each one has a unique rank and

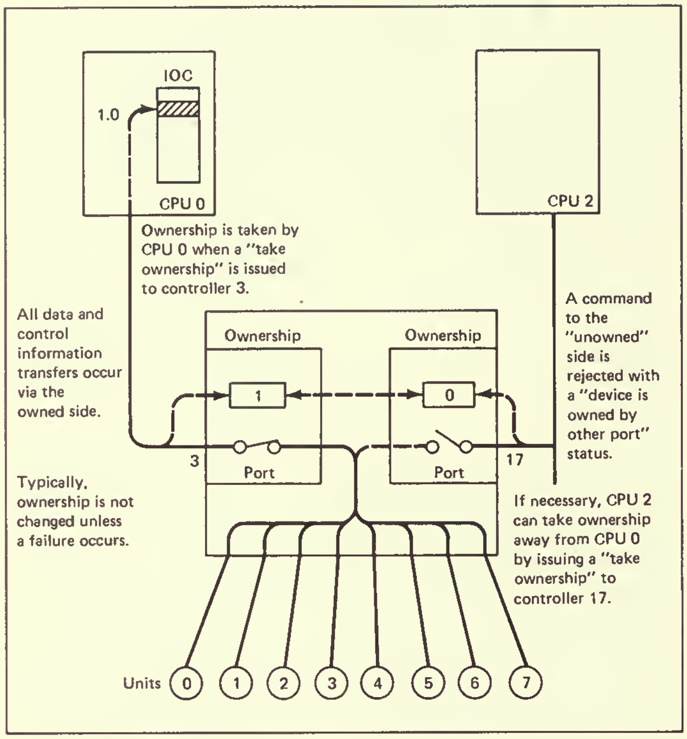
IOC
1.0
CPU 0
CPU 2
Ownership is taken by CPU 0 when a "take ownership" is issued to controller 3.
All data and control information transfers occur via the owned side.
Ownership
Ownership
1
0
3
17
Port
Port
A command to the "unowned" side is rejected with a "device is owned by other port" status.
Typically, ownership is not changed unless a failure occurs.
If necessary, CPU 2 can take ownership away from CPU 0 by issuing a "take ownership" to controller 17.
Units
0
1
2
3
4
5
6
7

Fig. 11. Ownership circuitry.

position designation. The highest priority controller is granted access to the interrupt system. Thus a radial polling technique allows the processor to resolve 32 different controller priorities in just two poll cycles. Each port of a controller has a separate set of configuration jumpers so that a controller can have different priorities on its primary and alternate path.

Controller Buffer Considerations

In the design of the Tandem 16 I/O system, a lot of attention was paid to the overrun problem. While overruns are possible on this system, they have been made a rare occurrence. Each I/O controller has 3 configurable settings: the I/O controller number, the interrupt priority, and buffer stress threshold reconnect setting.

Each I/O controller is buffered to some extent. The asynchronous terminal controller has 2 bytes of buffering, while the disc controller has 4K bytes of buffering. Considerations of device transfer rate, channel transfer rate, the individual controller's buffer depth, the controller's reconnect priority, and a given channel's I/O complement can be used to determine the buffer's depth (stress threshold) at which a reconnect request should be made to the channel to minimize the chance of overrun. Each controller with significant buffering (more than 32 bytes) has a configurable stress threshold. Buffer stress is defined as the number of cells full on an input operation, and the number of cells empty on output operations. In general, the I/O channel relieves stress while the I/O device generates more stress. Therefore the higher the stress, the more the buffer needs relief from the I/O channel, regardless of the direction of data transfer.

Tandem has developed a program which takes a system configuration and determines the appropriate stress threshold settings needed to guarantee no data overruns. Since reconnect overhead time is known, and all transfers on the I/O bus take place at memory speed, and the upper bound of the block length is known for each type of controller, it is a deterministic function as to whether or not an overrun is possible. If it is impossible to generate a no-overrun configuration, the program will output a minimum-overrun threshold setting. Most times, however, it is possible to iterate on the configuration until threshold settings can be determined that prevent overruns.

Disc Controller Considerations

The greatest fear that an on-line system user has is that "the data base is down" [Dolotta et al., 1976]. Many of these users are willing to pay the premium of having duplicated or "mirrored" data bases in case a disc drive fails. To meet this requirement, Tandem provides automatic mirroring of data bases.

A disc volume is a set of data contained on one spindle or one removable disc pack. A user may declare any of the disc volumes as mirrored pairs at system generation time (Fig. 12). The system then maintains these pairs so they always contain identical data. Thus protection is achieved for a single drive failure. Each disc drive in the system may be dual-ported. Each port of a disc drive is connected to an independent disc controller. Each of the disc controllers is also dual-ported and connected between two processors. A string of up to 8 drives (4 mirrored pairs) can be supported by a pair of controllers in this manner.

Note that in this configuration there are many paths to any given data and that data can be retrieved regardless of any single disc drive failure, disc controller failure, power supply failure, processor failure, or I/O channel failure.

The disc controller is buffered for a maximum length record which provides several features important in an on-line system. First, the disc controller is absolutely immune to overruns. Second, data to be written on two drives need be transferred over the I/O channel only once. The data may then be posted twice from the controller's internal buffer. Thus the channel's data transfer capacity is little impaired by mirrored volumes.

This disc controller uses a Fire code [Peterson, 1961] for burst error correction and detection. It can correct 11 bit bursts in the controller's buffer before transmission to the channel. Since overlapped seeks are allowed by the controller, when data is to be

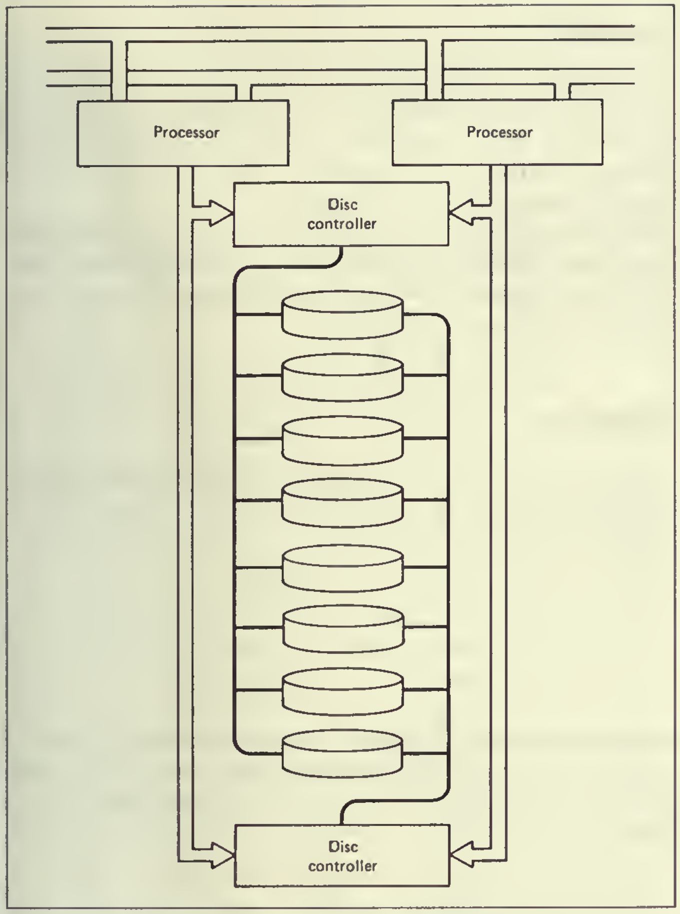

Fig. 12. Tandem 16 disc subsystem organization.

read from a mirrored pair it can be read from the drive which has its arm closest to the data cylinder. It is interesting to note that since the majority of transactions in an on-line system are reads, mirrored volumes actually can increase performance.

NonStop I/O System Considerations

The I/O channel interface consists of a two byte data bus and control signals. All data transferred over the bus is parity checked in both directions, and errors are reported via the interrupt system. A watchdog timer in the I/O channel detects if a non-existent I/O controller has been addressed, or if a controller stops responding during an I/O sequence.

The data transfer byte count word in the IOC entry contains four status bits including a protect bit. When this bit is set to "1" only output transfers are permitted to this device.

Because I/O controllers are connected between two independent I/O channels, it is very important that word count, buffer address, and direction of transfer are controlled by the processor instead of within the controller. If that information were to be kept in the controller, a single failure could cause both processors to which it was attached to fail. Consider what would happen if a byte count register was located in the controller and was stuck in such a situation that the count could not decrement to zero on an input transfer. It would be possible to overwrite the buffer and cause system tables to become meaningless. The error would propagate to the other processor upon discovery that the first processor was no longer operating.

Other error conditions that the channel checks for are violations of I/O protocol, attempts to transfer to absent pages (it is the operating system's responsibility to "tack down" the virtual pages used for I/O buffering), uncorrectable memory errors, and map parity errors.

4. Power, Packaging, On-Line Maintenance

The Tandem 16 power supply has 3 sections: a 5 volt interruptible section, a 5 volt uninterruptible section, and a 12–15 volt uninterruptible section. The interruptible section will stop supplying DC power when AC is lost while the uninterruptible sections will continue to supply DC power. The interruptible section powers I/O controllers and that portion of a processor which is not related to memory refresh operation. The uninterruptible sections provide power for the memory array and refresh circuitry. The 5 volt sections are switching regulated supplies while the 12–15 volt section is linearly regulated. The uninterruptible sections have a provision for a battery attachment so that in case of utility power failure, memory contents are kept for 1.5 to 4 hours, depending on the amount of memory attached to the supply.

The power supply accepts AC input of 110 or 220 volts ±20% to provide brownout insensitivity. At nominal line conditions, over 30 msec of ridethrough is provided by storage capacitors. A power-fail warning signal is provided when there is at least 5 msec of regulated power remaining so that the processor can go through an orderly shut down. Some users must remain operational through utility power failure and have generator systems which provide continuous AC power for the entire system, including peripheral devices.

The power-fail warning scheme in the Tandem 16 power supply monitors charge in the storage capacitors rather than monitoring loss of AC peaks as is conventionally done. This has the advantage that the 5 msec to do a power shutdown sequence in the processor is guaranteed even if it occurs after a brownout period.

The power supply provides all other prudent features required in a computer system, such as over voltage and over current protection, and over temperature protection.

The power-up sequencing on disc drives has been implemented with independent rather than daisy chained circuits. In the daisy chained approach, one bad sequencer circuit can cause the remaining drives in the chain not to sequence up after a power failure.

Further Packaging and On-Line Maintenance Considerations

Modularity is a key concept in the Tandem 16 system. The maintenance philosophy is to make all repairs by module replacement at the user site without making the system unavailable to the user. Therefore the back-planes, power supplies, fans, and I/O channels, as well as the PC cards, are modular and easily replaceable. Thumb screws are used when they can be so that a minimum of tools are needed for repair. The package is designed so that there is easy access to all modules.

Processors and I/O controllers not only can be replaced on-line, but added on-line without system interruption if expansion is planned, all without application software being changed.

Conclusion

The contribution of the Tandem 16 system lies in the synthesis of a system to directly address the need of the NonStop application marketplace. By avoiding the "onus of compatibility" to any previous system, an architecture could be designed from "scratch" that was "clean" and efficient.

The system goals have been met to a large degree. Systems have been shipped containing two to ten processors. Many application programs are on-line and running. They recover from failures, and stay up continuously.

References

Anderson and Jensen [1975]; Bartlett [1978]; Dolotta et al. [1976]; Katzman [1977]; Locks [1973]; Mil 217 [1965]; Peterson [1961]; Tandem [1976].

The Tandem 16: A "NonStop" Operating System[1]

Joel F. Bartlett

Summary The Tandem/16 computer system is an attempt at providing a general-purpose, multiple-computer system which is at least one order of magnitude more reliable than conventional commercial offerings. Through software abstractions a multiple-computer structure, desirable for failure tolerance, is transformed into something approaching a symmetric multiprocessor, desirable for programming ease. Section 1 of this paper provides an overview of the hardware structure. In Sec. 2 are found the design goals for the operating system, "Guardian." Section 3 provides a bottom-up view of Guardian.

1. Introduction

1.1 Background

On-line computer processing has become a way of life for many businesses. As they make the transition from manual or batch methods to on-line systems, they become increasingly vulnerable to computer failures. Whereas in a batch system the direct costs of a failure might simply be increased overtime for the operations staff, a failure of an on-line system results in immediate business losses.

1.2 System Overview

The Tandem/16 [Katzman, 1977; Tandem, 1976] was designed to provide a system for on-line applications that would be significantly more reliable than currently available commercial computer systems. The hardware structure consists of multiple processor modules interconnected by redundant interprocessor buses. A PMS [Bell and Newell, 1971] definition of the hardware is found in Fig. 1.

Each processor has its own power supply, memory, and I/O channel and is connected to all other processors by redundant interprocessor buses. Each I/O controller is redundantly powered and connected to two different I/O channels. As a result, any interprocessor bus failure does not affect the ability of a processor to communicate with any other processor. The failure of an I/O channel or of a processor does not cause the loss of an I/O device. Likewise, the failure of a module (processor or I/O controller) does not disable any other module or disable any inter-module communication. Finally, certain I/O devices such as disc drives may be connected to two different I/O controllers, and disc drives may in turn be duplicated such that the failure of an I/O controller or disc drive will not result in loss of data.

[1]Reprinted with the express permission of Tandem Computers Inc. "NonStop" is a trademark of Tandem Computers Inc.

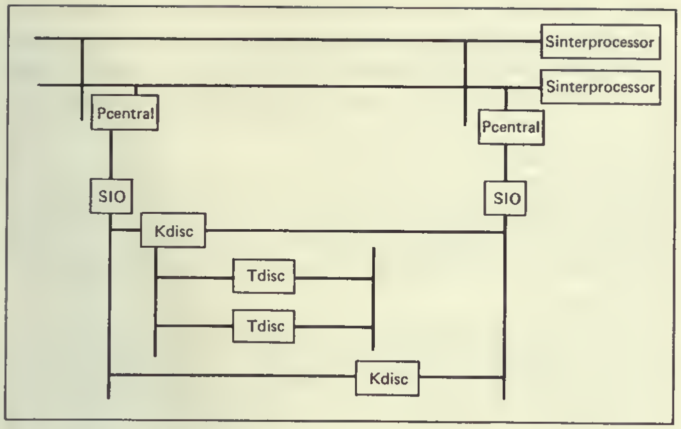

Fig. 1. Hardware structure.

The system is not a true multiprocessor [Enslow, 1977], but rather a "multiple computer" system. The multiple computer approach is preferable for several reasons. First, since no module is shared by the entire system, it increases the system's reliability. Second, a multiple computer system does not require the complex hardware needed to handle multiple access paths to a common memory. In smaller systems, the cost of such a multiported memory is undesirable; and in larger systems, performance suffers because of memory access interference.

On-line repair is as necessary as reliability in assuring system availability. The modular structure of the Tandem/16 system allows processors, I/O controllers, or buses to be repaired or replaced while the rest of the system continues to operate. Once repaired, they may then be reintegrated into the system.

The system structure allows a wide range of system sizes to be supported. As many as sixteen processors, each with up to 512K bytes of memory, may be connected into one system. Each processor may also have up to 256 I/O devices connected to it. This provides for tremendous growth of application programs and processing loads without the requirement that the application be reimplemented on a larger system with a different architecture.

Finally, the system is meant to provide a general solution to the problem of providing a failure-tolerant, on-line environment suitable for commercial use. As such, the system supports conventional programming languages and peripherals and is oriented toward providing large numbers of terminals with access to large data bases.

2. System Design Goals

2.1 Integrated Hardware/Software Design

The Tandem/16 system was designed to solve a specific problem. This problem was not stated in terms of hardware and software requirements, but rather in terms of system requirements. The hardware and software designs then proceeded in tandem to provide a unified solution. The hardware design concerned itself with the contents of each module, their interconnections to the common buses, and error detection and correction within modules and on the communication paths. The software design was given the problem of control; that is, selection of which modules to use and which buses to use to communicate with them. Furthermore, as errors are detected, it was the responsibility of the software to control recovery actions.

2.2 Operating System Design Goals

The first and foremost goal of the operating system, Guardian, was to provide a failure-tolerant system. This translated into the following design "axioms":

- The operating system should be able to remain operational after any single detected module or bus failure.
- The operating system should allow any module or bus to be repaired on-line and then reintegrated into the system.
- The operating system should be implemented in a reliable manner. Increased reliability provided by the hardware architecture must not be negated by software problems.

A second set of requirements came from the great numbers and sizes of hardware configurations that are possible:

- The operating system should support all possible hardware configurations, ranging from a two-processor, discless system through a sixteen-processor system with billions of bytes of disc storage.
- The operating system should hide the physical configuration as much as possible so that applications can be written to run on a great variety of system configurations.

3. Operating System Structure

To satisfy these requirements, the operating system was designed to have the appearance of a true multiprocessor at the user level. The design of the system was strongly influenced by Dijkstra's work on the "THE" system [Dijkstra, 1968a], and Brinch Hansen's implementation of an operating system nucleus for a single-processor system [Brinch Hansen, 1970]. The primary abstractions are processes, which do work, and messages, which allow interprocess communication.

3.1 Processes

At the lowest level of the system is the basic hardware as earlier described. It provides the capability for redundant modules, i.e., I/O controllers, I/O devices, and processor modules consisting of a processor, memory, and a power supply. These redundant modules are in turn interconnected by redundant buses. Error detection is provided on all communication paths and error correction is provided within each processor's memory. The hardware does not concern itself with the selection of communication paths or the assignment of tasks to specific modules.

The first abstraction provided is that of the process. Each processor module may have one or more processes residing in it. A process is initially created in a specific processor and may not execute in another processor. Each process has an execution priority assigned to it. Processor time is allocated on a strict priority basis to the highest priority ready process.

Process synchronization primitives include "counting semaphores" and process local "event" flags. Semaphore operations are performed via the functions PSEM and VSEM, corresponding to Dijkstra's P and V operations. Semaphores may only be used for synchronization between processes within the same processor. They are typically used to control access to resources such as resident memory buffers, message control blocks, and I/O controllers.

When certain low-level actions such as device interrupts, processor power-on, message completion or message arrival occur, they result in "event" flags being set for the appropriate process. A process may wait for one or more events to occur via the function WAIT. The process is activated as soon as the first WAITed for event occurs. Events are signaled via the function AWAKE. Event signals are queued using a "wake up waiting" mechanism so that they are not lost if the event is signaled when the process is not waiting on it. Like semaphores, event signals may not be passed between processors. Event flags are predefined for eight different events and may not be redefined.

When a process blocks itself to wait for some event to occur or for a semaphore to be allocated to it, it may specify a maximum time to block. If the time limit expires and the event has not occurred or the resource has not been obtained, then the process will continue execution but an error condition will be returned to it. This timeout allows "watch dog" timers to be easily placed on device interrupts or on resource allocations where a failure may occur.

Each process in the system has a unique identifier or "processid" in the form: <cpu#,process #>, which allows it to be referenced on a system-wide basis. This leads to the next abstraction, the message system, which provides a processor-independent, failure-tolerant method for interprocess communication.

3.2 Messages

The message system provides five primitive operations which can be illustrated in the context of a process making a request to some server process (Fig. 2). The process' request for service will send a message to the appropriate server process via the procedure LINK. The message will consist of parameters denoting the type of request and any needed data. The message will be queued for the server process, setting an event flag, and then the requestor process may continue executing.

When the server process wishes to check for any messages, it calls LISTEN. LISTEN returns the first message queued or an indication that no messages are queued. The server process will then obtain a copy of the requestor's data by calling the procedure READLINK.

Next, the server process will process the request. The status of the operation and any result will then be returned by the WRITELINK procedure, which will signal the requestor process via another event flag. Finally, the requestor process will complete its end of the transaction by calling BREAKLINK.

A communications protocol was defined for the interprocessor buses that would tolerate any single bus error during the execution of any message system primitive. This design assures that a communications failure will occur if and only if the sender or receiver processes or their processors fail. Any bus errors which occur during a message system operation will be automatically corrected in a manner transparent to the communicating processes and logged on the system console. The interprocessor buses are not used for communication between processes in the same processor, which can be done faster in memory. However, the processes involved in the message transfer are unable to detect this difference.

The message system is designed such that resources needed for message transmission (control blocks) are obtained at the start of a message transfer request. Once LINK has been successfully completed, both processes are assured that sufficient resources are in hand to be able to complete the message transfer. Furthermore, a process may reserve control blocks to guarantee that it will always be able to send messages to process a request

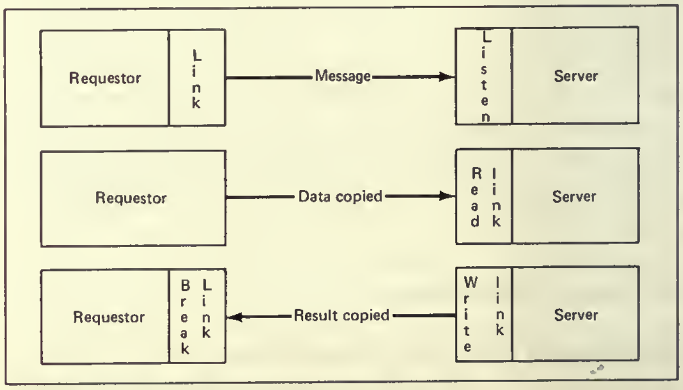

Fig. 2. Message system primitive operations.

that it picks up from its message queue. Such resource controls assure that deadlocks can be prevented in complex producer/consumer interactions, if the programmer correctly analyzes and anticipates potential deadlocks within the application.

3.3 Process-Pairs

With the implementation of processes and messages, the system is no longer seen as separate modules. Instead, the system can be viewed as a set of processes which may interact via messages in any arbitrary manner, as shown in Fig. 3.

By defining messages as the only legitimate method for process-to-process interaction, interprocess communication is not limited by the multiple-computer organization of the system. The system then starts to take on the appearance of a true multiprocessor. Processor boundaries have been blurred, but I/O devices are still not accessible to all processes.

System-wide access to I/O devices is provided by the mechanism of "process-pairs." An I/O process-pair consists of two cooperating processes located in two different processors that control a particular I/O device. One of the processes will be considered the "primary" and one will be considered the "backup." The primary process handles requests sent to it and controls the I/O device. When a request for an operation such as a file open or close occurs, the primary will send this information to the backup process via the message system. These "checkpoints" assure that the backup process will have all information needed to take over control of the device in the event of an I/O channel error or a failure of the primary process' processor. A process-pair for a redundantly-recorded disc volume is illustrated in Fig. 4.

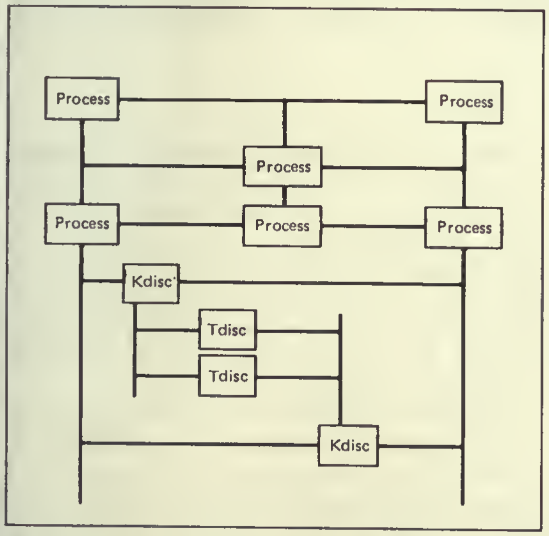

Fig. 3. System structure after the addition of processes and messages.

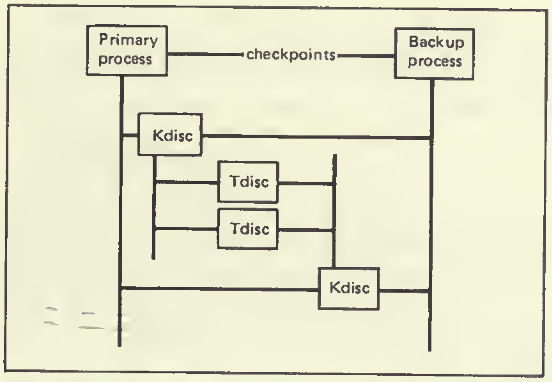

Fig. 4. Process-pair for a redundantly recorded disc volume.

Because of the distributed nature of the system, it is not possible to provide a block of "driver" code that could be called directly to access the device. While potentially more efficient, such an approach would preclude access to every device in the system by every process in the system.

The I/O process-pair and associated I/O device(s) are known by a logical device name such as "$DISC1" or by a logical device number rather than by the processid of either process. I/O device names are mapped to the appropriate processes via the logical device table (LDT) in every processor, which supplies two processids for each device. A message request made on the basis of a device name or number results in the message being sent to the first process in the table. If the message cannot be sent or if the message is sent to the backup process, an error indication will be returned. The processid entries in the LDT will then be reversed and the message resent. Note two things: first, the error recovery can be done in an automatic manner; and second, the requestor is not concerned with what process actually handled the request. Error recovery cannot always be done automatically. For example, the primary process of a pair controlling a line printer fails while handling a request to print a line on a check. The application process would prefer to see the process failure as an error rather than have the request automatically retried, which might result in two checks being printed.

The two primitives, processes and messages, blur the boundaries between processors and provide a failure-tolerant method for interprocess communication. By defining a method of grouping processes (process-pairs), a mechanism for uniform access to an I/O device or other system-wide resource is provided. This access method is independent of the functions performed within the processes, their locations, or their implementations. Within the process-pair, the message system is used to checkpoint state changes so that the backup process may take over in the event of a failure. This checkpoint mechanism is in turn independent of all other processes and messages in the system.

The system structure can be summarized as follows. Guardian is constructed of processes which communicate using messages. Fault tolerance is provided by duplication of components in both the hardware and the software. Access to I/O devices is provided by process-pairs consisting of a primary process and a backup process. The primary process must checkpoint state information to the backup process so that the backup may take over on a failure. Requests to these devices are routed using the logical device name or number so that the request is always routed to the current primary process. The result is a set of primitives and protocols which allow recovery and continued processing in spite of bus, processor, I/O controller, or I/O device failures. Furthermore, these primitives provide access to all system resources from every process in the system.

3.4 System Processes

The next step in structuring the system comes in assigning functions to processes. As previously shown, I/O devices are controlled by process-pairs. Another process-pair known as the "operator" is present in the system. This pair is responsible for formatting and printing error messages on the system console. Here is an example of where Guardian has not followed a strict level structure. The operator makes requests to a terminal process to print the messages, yet the terminal process wishes to send messages to the operator to report I/O channel errors. An infinite cycle is prevented by having the terminal process not send messages for errors on the operator terminal and having I/O processes never wait for message completions when sending errors to the operator. While it may be preferable to prevent cycles of any type in system design, they have been allowed in Guardian when it can be shown that they will terminate. The ability to reserve message control blocks assures that no cycle will be blocked because of resource problems.

Each processor has a "system monitor" process which handles such functions as process creation and deletion, setting time of day, and processor failure and reload cleanup operations.

A memory management process is also resident in each processor. This process is responsible for allocating a page of physical memory and then sending messages to the appropriate disc processes to do the actual disc I/O. Pages are brought in on a demand basis and pages to overlay are selected on a "least recently used" basis over the entire memory of the processor.

The choice of relatively unsophisticated algorithms for scheduling and memory management was a result of the fact that the system was not intended to be a general-purpose timeshare system. Rather, it was to be a system which supported multiple processes and terminals in an extremely flexible manner.

3.5 Application Process Interface

Above the process and communication structure there exists a library of procedures which are used to access system resources. These procedures run in the calling process' environment and may or may not send messages to other processes in the system. For example, the file system procedures do not do the actual I/O operations. Instead, they check the caller's parameters, and if all is in order a message is sent to the appropriate I/O process-pair. Likewise, process creation is seen as a procedure call to NEWPROCESS, which does nothing but check the caller's parameters and then send a message to the system monitor process in the processor where the process is to be created. On the other hand, a procedure such as TIME which returns the current time of day does not send any messages. In either case, the access to system resources appears simply as procedure calls, effectively hiding the process structure, message system, hardware organization, and associated failure recovery mechanisms.

3.6 Initialization and Processor Reload

System initialization starts with one processor being cold loaded from some disc on the system. The load file contains a memory image of the operating system resident code and data, with all system processes in existence and at their initial states. The system monitor process then creates a command interpreter process.

Guardian may be brought up even though a processor or peripheral device is down. This is possible because operating system disc images may be kept on multiple disc drives, I/O controllers may be accessed by two different processors, and the terminal that has the initial command interpreter on it is selected by using the processor's switch register.

After a cold load, the system logically consists of one processor and any peripherals attached to it. More processors and peripherals may be added to the system via the command interpreter command:

```
:RELOAD 1,$DISC
```

This command will read the disc image for processor 1 from the disc $DISC and send it over either interprocessor bus to processor 1. Once it is loaded, all processes residing in other processors in the system will be notified that processor 1 is up.

This command is also used to reload a processor after it has been repaired. Guardian does not differentiate between an initial load of a processor and a later reload. In each case, resources are being logically added to the system and processes must be notified so that they may make use of them.

The previous example of a reload message being sent to all processes is an example of how functions are split in Guardian. A mechanism is provided for informing a process of a system status change. It may then take some unspecified action (including doing nothing). Similarly, a system power-on simply sets the PON event flag for all processes. The operating system kernel must only insure that the process structure and message system are correctly saved and restored. It is then the responsibility of individual processes to do such things as reinitialize their I/O controllers.

3.7 Operating System Error Detection

Besides the hardware-provided single error detection and correction on memory, and single error detection on the interprocessor and I/O buses, additional software error checks are provided. The first of these is the detection of a down processor. Every second, each processor in the system sends a special "I'm alive" message over each bus to all processors in the system. Every two seconds, each processor checks to see that it has received one of these messages from each processor. If a message has not been received, then it assumes that that processor is down.

Additionally, the operating system makes checks on the correctness of data structures such as linked lists when operations are done on them. Any processor detecting such an error will halt.

All I/O interrupts are bracketed by a "watch dog" timer such that the system will not hang up if an I/O operation does not complete with the expected interrupt. If an I/O bus error occurs then the backup process will take over control of the device using the second I/O bus.

As previously noted, the interprocessor bus protocol is designed to correct single bus errors. In addition to this, extensive checks are made on the control information received over the buses to verify that it is consistent with the state of the receiving processor.

Power-fail/automatic restart is provided within each processor. A power-failure is detected independently by each processor module and as a result is not a system-wide, synchronous event. The system was designed to recover from either a complete system power-fail, or a transient which will cause some of the processors to power-fail and then immediately restart.

4. Conclusions

The innovative aspects of Guardian lie not in any new concepts introduced, but rather in the synthesis of pre-existing ideas. Of particular note are the low-level abstractions, process and message. By using these, all processor boundaries can be hidden from both the application programs and most of the operating system. These initial abstractions are the key to the system's ability to tolerate failures. They also provide the configuration independence that is necessary in order for the system and applications to run over a wide range of system sizes.

Guardian provides the application programmer with extremely general approaches to process structuring, interprocess communication, and failure tolerance. Much has been said about structuring programs using multiple communicating processes, but few operating systems are able to support such structures.

Finally, the design goals of the system have been met to a large degree. Systems, with between two and ten processors, have been installed and are running on-line applications. They are recovering from failures and failures are being repaired on-line.

References

Bell and Newell [1971]; Brinch Hansen [1970]; Dijkstra [1968a]; Enslow [1977]; Katzman [1977]; Tandem [1976].