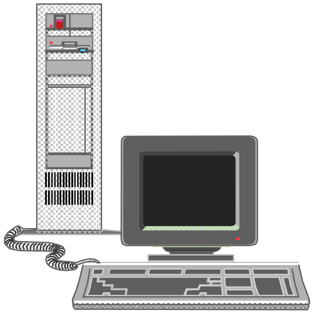
A NonStop Kernel



Luke Levesque
COP6614

Outline

- Hardware and Fault Models
- NonStop Kernel
- Messages
- System bus
- Process pairs
- Additional Issues
- Recent Improvements

Introduction

- Timeframe: late 70s early 80s.
- Fault-tolerant hardware and operating systems had been around for a while, but they were special-purpose (telephone equipment, etc.)
- Needed an expandable, general purpose fault-tolerant system for many different commercial needs (banks, airlines, etc.).
  - Primarily transaction processing
- Enter Tandem Computers and the NonStop kernel.
  - Specialized hardware coupled with the kernel
  - Designed to be online and operational 24/7 even "in the presence of a single fault".

Hardware

- System is a network of nodes
  - Redundancy is the key here
  - Each node contains two or more processors
    - Each has memory, power supplies, etc..
    - 16-bit processors
  - Redundant busses, disk I/O, and disks
    - Busses are faster than CPU to keep data moving across all nodes

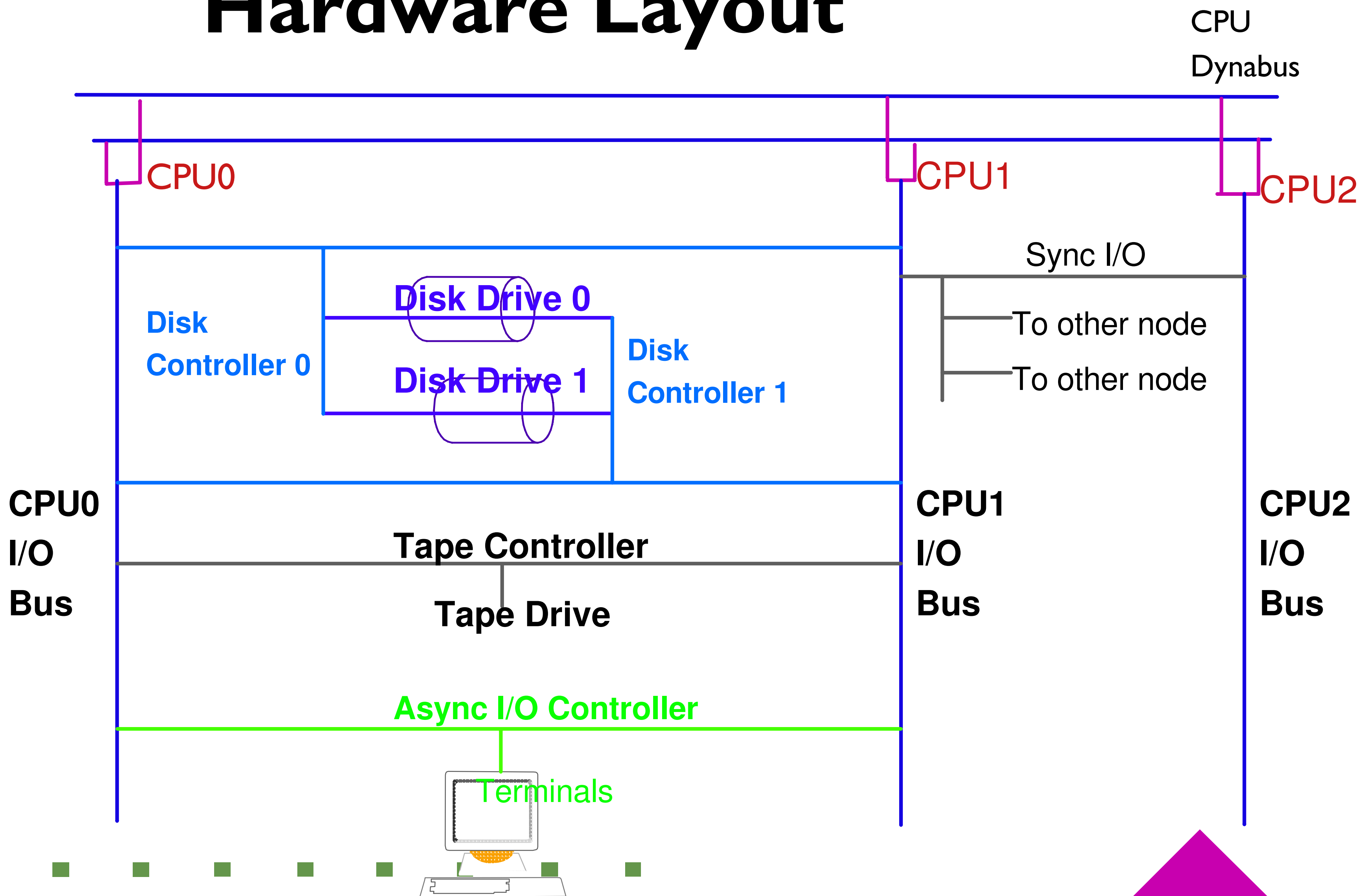
Hardware

- CPUs are stack-oriented
  - 1K words (512KB)
  - ECC RAM with double detection
  - Battery backup
  - Separate code and data areas
    - Data can be viewed as stack or flat
  - All I/O was with DMA
- Instruction set has built in support for sending messages (microcoded)

Hardware

- Received messages are placed in buffer and an interrupt given (memory access vs. instruction)
- Entire system may be located locally or at remote locations
  - Up to 255 processors
- CPU was microcoded

Hardware Layout



Hardware Fault Model

- Single fault should not bring down system
  - Redundancy helps with this
- Must be able to repair and reintegrate while online
- Types of faults:
  - Power supply, processor, memory: Disables that processor
  - Interprocessor bus or I/O channel: Disables that bus
  - I/O Controller: Disables the controller

Hardware Fault Model

- Physical events that create a fault:
  - Permanent hardware failure. Uses recovery algorithm to prevent data loss.
  - Intermittent hardware failures are difficult to detect. May corrupt data. Fail-fast.
  - External factors (A/C breaks, 'user error', etc.)
- Corruption-free recovery is important in all cases!

NonStop Kernel Overview

- System provides many advanced feature:
  - Multiprogramming
  - 1 gig of virtual memory per processor
  - Reliable Communication (inter-process and interprocessor)
  - Fault tolerant
  - Modular and Expandable
  - Everything is a file (abstraction)
  - SQL/Database integrated with file system

NonStop Kernel - Inspiration

- Such a kernel had never been done before, so no existing work to study
- Authors instead took inspiration from:
  - Brinch Hansen's Nucleus: Detailed reliable message passing between processes and devices and the API needed to support it.
  - Dijkstra's THE system: Rings of protection with levels of abstraction.

NonStop Kernel - Processes

- Processors support 256 processes each
- Code sharing allowed
- Cost of context switch is mostly because of memory mapping
- All processors have both a monitor and memory manager process
  - Monitor function creates/removes processes, controls the message system, responds to information queries, and fault recovery/management
- Processid composed of: node number, processor number, process number, and a timestamp or symbolic name

NonStop Kernel - Processes

- Process synchronization primitives
  - Counting semaphores
    - Used mostly within the kernel for access to shared I/O, etc..
  - Local event flags
    - Signals the process that a specific event has occurred
      - Disk I/O complete
      - Incoming message from another process
      - Message has been completed elsewhere
- Primitives are not exhaustive

Messages

- Any sort of communication outside a process is done with messages.
  - Even system calls, such as creating a new process
  - Shared memory does not exist in system, even among local processes
- Messages can be queued FIFO or by priority
- Message interface hides details of sending and receiving
  - Includes location of process and any sort of transmission errors
- Messages consist of a request and a response

Messages

- This methology is very similar to making a function call and getting a result
  - Each request is acknowledged, to improve fault tolerance
  - Also allows for background processing of requests
  - This is NOT RPC
- Application is not allowed direct access to messages at all
  - Accomplished using standard user/supervisor mode of processor
  - Gives OS complete control over handling messages and error conditions

Messages

- Messages are sent by value, NOT address
- System status messages can be sent without expecting a reply
  - No buffer can be used
- Messages may be queued up internally (no need to service right away)
- Message cancellation provides a way to signal other hardware or software failures not necessarily caused by the two processes
  - Might trigger recovery algorithms in processes as needed
  - Cleans up waiting messages if process or processor fails

Message Resource Control

- Risk using up lots of resources when sending messages
- Need to keep resources available for system messages and to avoid monopolization
- Resources also need to be available to receive incoming messages
- Solution
  - LCBs (Link Control Block) for sending and receiving can be reserved in advance by OS and processes
  - An additional pool of the remaining LCBs can be utilized if all reserved LCBs are in use

Message Resource Control

- Solution
  - If an LCB cannot be obtained after 10 seconds (hardcoded value), the call will fail and no message can be sent until an LCB is freed
  - Message buffers are allocated as needed and from different pools depending on the type of request
  - Server processes have some buffer pools permanently allocated so they can always service requests

System Bus Protocol

- Messages between processes on the same processor is easy
- Messages to other processors and nodes need to detect and handle errors seamlessly
- Types of errors
  - Process does not exist
  - Other processor is off-line
  - No free LCB
- Recovery mechanism was desired to be as simple as possible yet fulfill all requirements

System Bus Protocol

- Message packets sent with sequence numbers and checksum
- After transmission, sender puts packet info on Wait ACKnowledge (WACK) list
  - When packet is acknowledged, it is taken off the list
  - If packet not acknowledged in 1 second, packet is resent on secondary bus
- Repeat failures mark that route as down

System Bus Protocol

- When packets arrive at a processor (signaled by an interrupt), it checks:
  - Sequence numbers are as expected
  - Checksum is good
  - Correct Routing
- On error, packet is flushed. Sender must do error handling on its own.
- Info about each processor and their packets is maintained in the Bus Receive Table (BRT)
- ACKs are sent unsequenced or piggybacked as needed

System Bus Protocol

- How do we know CPUs are alive and well?
  - Every second, each processor sends an unsequenced ACK over both busses to each processor
    - Also serves to clean up from lost message ACKs
  - If processor is not heard from within two seconds, it is marked as down and all messages destined to it are canceled

Process Pairs

- Hardware is redundant, but what about applications or I/O device drivers?
- Solution is process-pairs
  - Pair of processes and symbolic name make up an I/O device driver or application process
  - Primary process (of the pair) handles requests and sends checkpoint data to backup process so that it is up to date
  - If the primary process fails, the backup process can take over without delay and without service interruption

Process Pairs

- A symbolic name is associated with BOTH primary and backup processes in a name table on each node
- When a message is sent to the named process, the table directs the message to the primary process
  - If the process was down, the table entries are swapped and the message resent to the backup process

Process Pairs Error Recovery

- More advanced error recovery needed during non-retryable requests (database modifications, voting tabulation, etc..)
- A system of tracking such requests must be used to prevent certain requests from being processed more than once
- Such requests are assigned sequence numbers to aid in synchronization between processes

Process Pairs Error Recovery

- Example: R and R' are primary/backup requester. S and S' are primary/backup server.
- 1. R=0 R'=0 --> S=0 S'=0
  - 2. IF req seq < my seq, THEN return saved status
- 3. R=0 R'=0 S=0 --> S'=0
- 4. R=0 R'=0 S=1 --> S'=1
- 5. R=1 R'=0 <-- S=1 S'=1
- 6. R=1 --> R'=1 S=1 S'=1

Process Pairs Error Recovery

- If R' or S' fail, the operation is not affected
- If R fails just after making the request, R' repeats the request. S will just send the saved status.
- If S fails during the operation, S' becomes S and either does nothing or completes the operation.
  - R may resend the request, and the request will either be done or the saved status returned
  - Could be a small window for operations to be physically repeated

Additional Issues

- Some performance loss due to message passing
  - Offset against fault tolerance
- Can be difficult to develop process pairs
  - High level languages (COBOL) can do some of this automatically
- Designing a good online application is the most difficult part
  - Monolithic
  - Too many processes

Additional Issues

- In transaction systems, the CPU is used to move data around, with very little processing
- Improving memory access will therefore improve performance
  - More processors are often added just for this reason - more memory!

Recent NonStop Improvements

- Interprocessor communication was converted into fiber optics (FOX system). Reduced electrical noise issues.
  - Store and forward with multiple paths
- Before going to MIPS processors in the 90s, certain models had microcode in RAM to allow processor updates
- More I/O channels
- Special dual diagnostic processors

Recent NonStop Improvements

- Spare RAM
- Newer versions of the hardware, such as in the S-series, use lockstep processors
  - Two processors perform the same operations
  - Output from both is compared to detect failure
- Moved from MIPS processors to Itanium in 2003
  - Loose lockstepping
  - Mostly common components
- Processors still use share-nothing architecture!

Conclusion

- No previous work on continuous uptime
- Tandem created a system capable of continuous operation even during a hardware or software fault
  - Everything is redundant in hardware and software
- Message-oriented system
- Corruption of data is avoided during fault

References

- Bartlett, Joel: A NonStop Kernel, Tandem Computers, 1981.
- Bartlet, Joel, et al. Fault Tolerance in Tandem Computer Systems. Tandem publication 90-5. May 1990.
- Dijkstra, E. W., The Structure of the "THE" Multiprogramming System. Comm. ACM 11, May 1968, pp. 341-346.
- Brinch Hansen, P., The Nucleus of a Multi-Programming System, Comm. ACM 13, April 1970, pp. 238-241, 250.
- HP NonStop Advanced Architecture FAQ. http://www.hp.com/go/nonstop. Feb 2004.
- HP NonStop Kernel Product Description. http://www.hp.com/go/nonstop. Sept 2002.