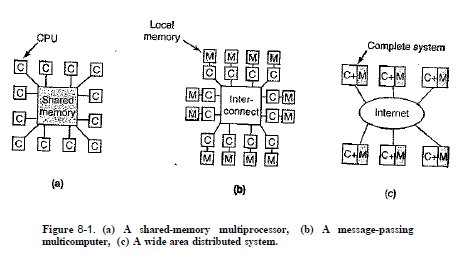
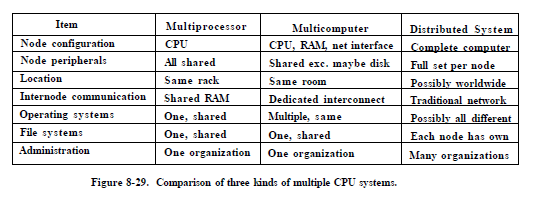
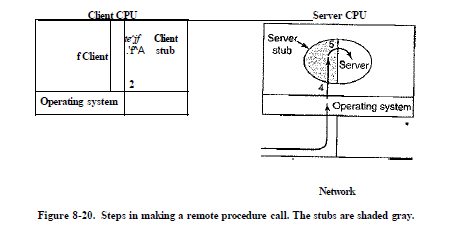
**Chapter 8 (Multiple Processor Systems)**

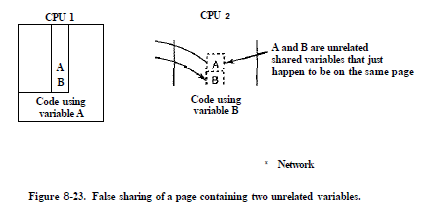




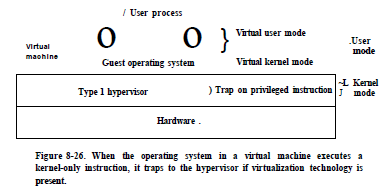
* Shared-memory multiprocessor is a computer system in which two or more CPUs share full access to a common RAM.
  + Various Models:
    - Each CPU Has Its Own Operating System
    - Master-Slave Multiprocessors
    - Symmetric Multiprocessors (There is one copy of the operating system in memory, but any CPU can run it.)
  + Scheduling has two dimensions. The scheduler has to decide which thread to run and which CPU to run it on.
* In message-passing multicomputer, each memory is local to a single CPU and can be accessed only by that CPU. CPUs communicate by sending messages over the interconnect.
* The only real difference between message-passing and Distributed system is that in the latter, complete computers are used and message times are often longer than message-passing.
* Message-passing multicomputer:
  + Send and Receive messages
  + Blocking (synchronous) versus Nonblocking (asynchronous) Calls
* Remote Procedure Call (RPC)
  + Message-passing model suffers from one incurable flaw: the basic paradigm around which all communication is built is input/output, and many people believe that I/O is the wrong programming model.
  + Solution by allowing programs to call procedures located on other CPUs.
  + When a process on machine 1 calls a procedure on machine 2, the calling process on 1 is suspended, and execution of the called procedure takes place on 2.
  + Information can be transported from the caller to the callee in the parameters and can come back in the procedure result. No message passing or I/O at all is visible to the programmer.
  + To call a remote procedure, the client program must be bound with a small library procedure called the client stub that represents the server procedure in the client's address space, the server is bound with a procedure called the server stub.
  + **Step 1** is the client calling the client stub. This call is a local procedure call, with the parameters pushed onto the stack in the normal way.
  + **Step 2** is the client stub packing the parameters into a message and making a system call to send the message. Packing the parameters is called **marshaling**.
  + **Step 3** is the kernel sending the message from the client machine to the server machine.
  + **Step 4** is the kernel passing the incoming packet to the server stub.
  + **Step 5** is the server stub calling the server procedure. The reply traces the same path in the other direction.



* + Implementation Issues:
    - Passing pointers is impossible because the client and server are in different address spaces, solution is to wrap or marshal values and send them instead of pointers.
    - Printf, for example, which may have any number of parameters (at least one), and they can be an arbitrary mixture of integers, shorts, longs, characters, strings, floating-point numbers of various lengths, and other types. Trying to call printf as a remote procedure would be practically impossible.
    - Use of global variables
* Distributed Shared Memory (DSM):
  + In DMS, each page is located in one of the memories, and each machine has its own virtual memory and its own page tables.
  + When CPU does a LOAD or STORE on a page it does not have, a trap to the operating system occurs. The operating system then locates the page and asks the CPU currently holding it to unmap the page and send it over the interconnection network. When it arrives, the page is mapped in and the faulting instruction is restarted.
  + In a DSM system, the address space is divided up into pages, with the pages being spread over all the nodes in the system.
  + **Replication**: One improvement to the basic system that can improve performance considerably is to replicate pages that are read only.
  + Advantages to a larger page size for DSM:
    - The startup time for a network transfer is fairly substantial, by transferring data in large units, when a large piece of address space has to be moved, the number of transfers may often be reduced.
  + Disadvantages to a larger page size for DSM:
    - The network will be tied up longer with a larger transfer, blocking other faults caused by other processes.
    - False sharing: a page containing two unrelated shared variables, A and B. Processor 1 makes heavy use of A, reading and writing it. Similarly, process 2 uses B frequently, the page containing both variables will constantly be traveling back and forth between the two machines because they appear by accident on the same page. when a process uses one of them, it also gets the other.
      * The larger the effective page size, the more often false sharing will occur, and conversely, the smaller the effective page size, the less often it will occur.



* Visualization:
  + It allows a single computer to host multiple virtual machines, each potentially running a different operating system.
  + Advantages
    - A failure in one virtual machine does not automatically bring down any others. thus maintaining the partial failure model that a multicomputer has, but at a much lower cost and with easier maintainability. (**strong isolation**)
    - Having fewer physical machines saves money on hardware and electricity and takes up less office space
    - Checkpointing and migrating virtual machines is much easier than migrating processes running on a normal operating system.
    - Run legacy applications on operating systems no longer supported or which do not work on current hardware.
    - A programmer who wants to make sure his software works on different operating systems can install vertual machine with different operating system.
  + The problem is if the server running all the virtual machines fails, the result is even more catastrophic than a single dedicated server crashing.
  + **Sensitive** Instructions: instructions that may only be executed in kernel mode.
  + **Privileged** instructions: instructions that cause a trap if executed in user mode.
  + Two Approaches:
    - Type 1 hypervisor (virtual machine monitor): It is the operating system, since it is the only program running in kernel mode, with job to support multiple copies of the actual hardware, called virtual machines, similar to the processes a normal operating system supports.
    - Type 2 hypervisor: It is a user program running on host OS that interprets the machine's instruction set, which also creates a virtual machine.
  + OS running on top of the hypervisor in both approaches is called **guest operating system**. In the case of a type 2 hypervisor, the operating system running on the hardware is called the **host operating system**.
  + Type 1 Hypervisors:
    - The virtual machine runs as a user process in user mode, and as such, is not allowed to execute sensitive instructions.
    - When the guest operating system executes a sensitive instruction, a trap to the kernel occurs
      * The hypervisor can then inspect the instruction to see if it was issued by the guest operating system in the virtual machine or by a user program in the virtual machine.



* + Type 2 Hypervisors:
    - Such as VMware, that runs as an ordinary user program on top of a host operating system such as Windows or Linux. It then installs the guest operating system to its virtual disk.
    - It scans the code first looking for basic blocks, The basic block cached first and is inspected to see if it contains any sensitive instructions, If so, each one is replaced with a call to a VMware procedure that handles it. (**binary translation**)
      * A basic block not containing any sensitive instructions will execute exactly as fast under VMware as it will on the bare machine.
      * Sensitive instructions are caught this way and emulated.
  + Paravirtualization:
    - A different approach is to modify the source code of the guest operating system so that instead of executing sensitive instructions at all, it makes hypervisor calls. In effect the guest operating system is acting like a user program making system calls to the operating system (the hypervisor).
      * API for use by guest operating systems.
      * Hypervisor becomes a microkernel.
    - Difference between true virtualization and paravirtualization:
      * On true virtualization, when a sensitive instruction is executed, the hardware causes a trap to the hypervisor, which then emulates it and returns.
      * On paravirtualization, when it needs to do I/O or change critical internal registers, it makes a hypervisor call to get the work done, just like an application program making a system call in standard Linux.
    - Issues: Solution → **VMI** (Virtual Machine Interface) form a low-level layer that interfaces with the hardware or hypervisor.
      * if the sensitive instructions are replaced with calls to the hypervisor, how can the operating system run on the native hardware?
      * what if there are multiple hypervisors available in the marketplace with different hypervisor APIs
* Distributed Systems:
  + Distributed System vs multicomputer
    - Each node in a distributed system is a complete computer, the nodes of a multicomputer are normally in a single room.
    - nodes in multicomputer communicate by a dedicated high-speed network, whereas the nodes of a distributed system may be spread around the world
    - All the nodes of a multicomputer run the same operating system, share a single file system, and are under a common administration, whereas the nodes of a distributed system may each run a different operating system.
  + loose coupling in a distributed system is both a strength and a weakness. It is a strength because the computers can be used for a wide variety of applications, but it is also a weakness, because programming these applications is difficult.
  + Distributed system has a layer of software on top of the operating system (**Middleware**).
  + Document-Based Middleware
    - Example, World Wide Web (WWW), every computer can hold one or more documents (Web pages).



* + File-System-Based Middleware
    - Using a file system model for a distributed system means that there is a single global file system, with users all over the world able to read and write files for which they have authorization.
    - Issues:
      * Transfer Model:
        + Choice between the upload/download model (download/copy before access, upload when done) and the remote access model (client sends commands to server).
        + upload/download model is simple and more efficient to transfer the whole file, but local storage should be enough to hold files, and there’s a consistency problem.
      * The Directory Hierarchy:
        + whether all clients have the same view of the directory hierarchy.
      * Naming Transparency:
        + **location transparency (location independence)**, means that the path name gives no hint as to where the file is located.
      * Semantics of File Sharing:
        + The system enforces an ordering on all system calls, and all processors see the same ordering. (sequential consistency)
        + Sequential consistency can be made in Distributed systems if there’s no caching at clients (not practical).
        + Another solution is to propagate all changes to cached files back to the server immediately (inefficient).
        + Alternative solution is to relax the semantics of file sharing. Instead of requiring a read to see the effects of all previous writes, one can have a new rule "Changes to an open file are initially visible only to the process that made them. Only when the file is closed are the changes visible to other processes.” (**session semantics**)
        + Alternatively, using upload/download model, with automatically lock a file.
  + Object-Based Middleware
    - An object is a collection of variables that are bundled together with a set of access methods.
    - CORBA (Common Object Request Broker Architecture) is a client-server system based on run-time objects.
      * ORBs (Object Request Brokers) are used to make it possible for a client on one platform to invoke a server on a different platform, through an interface (IDL “Interface Definition Language”)
  + Grids
    - A grid is a large, geographically dispersed, and usually heterogeneous independent collection of machines connected by a private network or the Internet, and which offers a a set of services to its users.
    - compared to a virtual supercomputer.
    - The original motivation for building a grid was sharing CPU cycles. When an organization did not need all of its computing power another organization could harvest those cycles.