CMPSCI 677 Operating Systems **Spring 2016**

Lecture 7: February 10

Lecturer: Prashant Shenoy *Scribe: Tao Sun*

7.1 Server Design Issues

7.1.1 Server Design

There are two types of server design choices:

- **Iterative server**: It's a single threaded server which processes the requests in a queue. While processing the current request it adds incoming requests in a wait queue, and once the processing is done, it takes in the next request in the queue. Essentially, requests are not executed in parallel at any time on the server.
- **Concurrent server**: In this case when a new request comes in, the server spawns a new thread to service the request. Thus all processes are executed in parallel in this scenario. This is the *thread-per-request* model. There is also one more flavor to the multi-threaded server. Unlike the previous case where new thread is spawned every time a new request comes in, there is a pool of pre-spawned threads in the server which are ready to serve the requests. A thread dispatcher or scheduler handles this pool of pre-spawned threads.

7.1.2 How to locate an end-point(port number) ?

The client can determine the server it need to communicates using one of the two techniques.

- Hard code the port number of the server in the client. If the server moves then the client will need to be recompiled.
- A directory service is used to register a service with a port number and IP address. The client connects to the directory service to determine the location of the server with a particular service, and then sends to it the request. In this case, the client only needs to know the location of the directory service.

7.1.3 Stateful or Stateless ?

- **Stateful servers** are those, which keep a state of their connected clients. For example, push servers are stateful servers since the server need to know the list of clients it needs to send messages to.
- **Stateless servers** on the other hand don't keep any information on the state of its connected clients, and can change its own state without having to inform any client. In this case, the client will keep its own session information. Pull servers are examples of stateless servers where the clients send requests to the servers as and when required. The advantage of stateless servers are that the server crash will not impact clients – more resilient to failures, it's also more scalable, since clients take some workloads.

- **Soft state servers** maintain the state of the client for a limited period of time on the server and when the session timeouts it discards the information.

7.1.4 Server clusters

In a cluster, the servers are segregated according to their functionality into tiers and each tier is replicated so as to enhance serviceability. For example, the database tier of the system can be replicated and distributed on several different machines, thus when a request comes in, the dispatcher needs to identify which machine should the request be directed to. These dispatchers are typically called *load balancers*. It keeps track of load on each server in the cluster and is also responsible for maintaining session details of clients, so that it can send the request to the correct server upon concurrent requests by the same client. Maintaining consistency among replicated data is important to ensure correct operation. Once the dispatcher gets the request from the client, it uses the following techniques to transfer the request to the correct sever.

- **TCP Splicing**: The client connects to the dispatcher through TCP connection but there is no direct connection between the client and server. The dispatcher sets up a separate connection between the server and itself and then splices the client connection and server connection together so that data transmission is possible between server and client.
- **TCP Handoff**: The dispatcher hands off the request from the client directly to the server at TCP Level or using HTTP redirect.

7.1.5 Server Architecture

7.1.5.1 Different types of server architectures:

- **Sequential**: In this architecture, a single-threaded process is used to serve requests one at a time and no concurrency is achieved. It can service multiple requests by employing events and asynchronous communication.
- **Concurrent**: two types:
  - Multi-threaded server: In this design, a new thread is spawned each time a request is received. This is the thread-per-request mode. A pre-spawned pool of threads could also be used to serve incoming requests.
  - Multi-process server: Similar to the multi-threaded server design, a process-per-request model or a pre-spawned pool of processes could be used to serve incoming requests.
- **Event-based server**: In this technique, asynchronous non-blocking calls are used in a single-threaded server. For example when asynchronous I/O is used, file reads which are usually blocking calls are no longer blocked, and are instead handed over to the OS. In the meantime, the next request in the queue could be processed. When the I/O operation is complete, an interrupt is received and the original request resumes execution. This way, we can use a single thread of execution but still achieve some degree of concurrency since non-blocking calls are used.

7.1.5.2 Which architecture is most efficient?

In the uni-core scenario, event-based implementation $>$ thread-based implementation $>$ process-based implementation $>$ pure sequential based implementation.

Event-based implementation though sequential offers concurrency because asynchronous non-blocking calls are used. Hence when a request being executed issues a non-blocking I/O call, it is passed onto the OS, and the next request in the queue is executed until the response for the I/O is received via an interrupt. Moreover the event-based implementation is a sequence of function calls rather than context-switches (among threads or processes) that are present in both the thread-based or process-based implementations making it an efficient design. However, it is more difficult to develop event-based programs when compared to multi-threaded or multi-process programs. Threads are more light-weight (less context switch and less memory consumption) to create than processes. Hence thread-based implementation is more efficient than process-based implementation. In general, concurrent implementation is more efficient than pure sequential implementation since more requests could be served at once in concurrent servers. In sequential servers, only one request can be served at a time.

However, in the multi-core environment, event-based servers will not achieve true parallelism since it has only one thread of execution. In this scenario, a thread-based implementation would perform better as it could run on multiple cores simultaneously. If you really need a very high performance server, what you will actually do is to have as many processes as your cores. Each process does event-based. So you actually use hybrid architecture. You create one process per core and each process is going to serve asynchronously. Those are the most efficient server you can write. But they are very complicated.

7.2 Code and Process Migration

As part of distributed scheduling, code/processes may need to be migrated from one machine to the other. In some cases, entire VMs may need to be migrated. Both these cases will be discussed.

7.2.1 Motivation

- **Performance and flexibility** are two important reasons why code/process migration is done. By moving code/processes, the load on a system can be redistributed for better performance. The program can be executed on any machine, which offers the flexibility of not being tied to just one machine.

- **Code Migration** (aka weak mobility): In this case, code is shipped from one machine to another. Code migration can only be done before the process begins, once the code starts executing, it is essentially process migration. Code migration is somewhat simpler as only the code needs to be moved and started on the receiving machine. This is much more common than process migration. Here are some examples: 1) Filling a form on a browser which then sends a query that is executed in the server. 2) A Java applet, the client downloads the applet code from sever and executes it on the local machine.

- **Process Migration** (aka strong mobility): In this case, a process that is running is moved to another machine. This is more complicated than code migration, as every process has state (like an address space and resources) that needs to be transferred, and then it needs to be resumed from where it was suspended. Examples include Condor, Sprite(Berkeley), and DQS.

- **Flexibility**: Many operating systems today let you plug in any device without having the driver software pre-installed. If the driver software is not present, the OS will offer to download it from a repository or from the internet. This is another example of code migration. Hence code migration offers more flexibility as you don't need to have every piece of software/driver pre-installed and can be downloaded on demand.

7.2.2 Migration Models

- Processes have an address space that consists of a code segment, a stack segment, and a heap segment. Apart from these, the processes could also have associated resources that it uses. If a process is migrated, all its associated code and resources need to be transferred.
- Migrating code is simpler than migrating an entire process. During weak mobility, the binary code or the script is transferred to the new system and is executed from the initial state.
- Migration can either be sender-initiated or receiver-initiated. There is a difference between whether the sender pushed the code (sender-initiated) or whether the receiver requested it (receiver-initiated). E.g. a query that is sent to the server is sender-initiated code migration. A Java applet is an example of the receiver-initiated code migration, the client requests it from the server and then downloads to its own system.

7.2.3 Who executes migrated entity?

- **Code migration**: The entire code is transferred to the new system and is executed from the initial state.
- **Process migration**: The existing process including the code and execution segments and the resources it is using, is moved to the new system from where it continues executing. This can be implemented in two ways:
  - Remote cloning: The process is cloned to the new system. Once completed, the cloned process continues executing and the former process in the old system is stopped.
  - Cloning is not involved. The actual process is moved to a new machine where it registers with the new system and continues executing.

7.2.4 Models for Code Migration

As discussed previously, we have two types of mobility, strong mobility and weak mobility, each of them can be sender or receiver initiated, the weak mobility can be executed at target process or at separate process, and the strong mobility can be remote cloning or purely process migration. In total we have 8 settings.

7.2.5 Resources Migration

Migrating a process from one machine to another requires stopping the process at the point of execution on one machine and then starting from the same point in other machine. When process migration takes place then code segment, resource segment and execution segment needs to be migrated to the new machine. This can be a tedious task as many constraints are involved. For instance, if a process on the original system is accessing a file and the process is migrated, an error would be caused when the same file is accessed in the new system as that file doesnt exist. Before we look at how to handle these resources, we discuss different types of process to resource bindings.

- Binding by identifier: This is a hard binding (accessing local files on A), such kind of resources need to be moved to system B since the process will use them as is i.e use the same file name to access the open file in System B. Other examples include specific web pages, ftp servers that will need to accessed.

- Binding by value: Suppose the process was running in JVM 2.0 environment in system A, then when it migrates to system B, it will expect JVM 2.0 environment to be present to continue execution. Thus when process migrates to B, a reference to the JVM 2.0 present on B has to be provided to the process.
- Binding by type: This is the weakest form of binding. Suppose the process was accessing hardware devices such as a printer, then when it moves to machine B, this particular process can use the device that is attached to machine B.

The above bindings show that some resources allow more flexibility when a process is moved than other resources. Now we need to discuss if we can move the resources to different machines i.e. resource to machine binding.

- Unattached resources: These are generally data files that have been opened by the process for read / write operation. These resources can be moved easily and quickly to another machine but they need be referenced properly in the new system so that process can access the path of the files correctly.
- Fastened resources: These are generally local databases on machine A that are used by the process. Moving these resources to new machine B is possible but can be expensive. Thus, it is preferable to create global references for such resources that can be accessed over network.
- Fixed Resources: These are resources like printer or other devices which cant be moved. For these generally global referencing technique is used. For example, open socket connections can be accessed through tunneling. Another option is rebinding the process to locally available resources.

Different combinations of the bindings produce different type of complexity and each has to be handled in a different way. For example, if we have a resource bound by the identifier like a url and it is unattached, then we would need to move that resource to the new machine. If however, the resource is fixed, then we would need to provide remote access to the old machine over a network. If no access is provided to the resource then the process will break.

7.2.5.1 Network Socket Migration

If a process has a socket open on the original machine A and that process is moved to machine B, then a tunnel will be created and used to forward all the packets received on the original socket in machine A from the source to machine B. Though complicated, this is the only way the connection remains active since the process is not aware of the migration. If the process is aware, then it could communicate the new port and IP information with the source.

7.2.6 Migration in Heterogeneous Systems

When a process is moved in a heterogeneous environment, i.e. between machines using different architectures (Intel / ARM), this leads to a real challenge since the binaries, assembly code instructions are all different for different architectures. The code will need to be recompiled according to new architecture and care has to be taken about little Endian and big Endian formats etc. One way to achieve migration on heterogeneous system is binary translation of code on the fly, i.e. each instruction received is converted to new machine-readable instruction. However, this is not the efficient way to do migration. These issue can be resolved using virtual machines or for instance programs running on JVM since it is platform independent. Other instances include migrating scripts such as python programs. As long as a python interpreter is present in the new machine, the code can be executed.

7.2.7 Virtual Machine Migration

VMs are migrated when the server it is on cannot handle more capacity. One way to do this is to shut down the VM and copy the virtual disk from one machine to another where it can be restarted. There will be a downtime in this method of migration since the VM has to be suspended, moved and then resumed. This downtime can be averted using live migration. For live migration we can assume that the disk is shared between the two machines. This is the case in a LAN environment. However, the memory state is moved using *iterative copying.* In this technique, while the VM is running on machine A the memory states of the process running in the VM is copied to the new machine B, and once the copy is completed, the VM in machine B can start executing after the VM in machine A is terminated. Thus user will not see any delay while switching. During live migration of process, the RAM pages are copied from machine A to machine B while process is still being executed in A. During the duration of copy, the process in machine A would have made some changes to the RAM pages. To overcome this, iterative copying of the new pages is done until only a fraction of pages remain. The idea is that in each iteration, the number of pages transferred decreases since each consecutive iteration is shorter than the previous one. When this fraction is small enough, all the new pages are copied to the new machine, the original machine is paused and the new machine resumes from where the old machine paused.

Now suppose, the process in VM of machine A uses network connection to access data through an open port, then in this scenario, the switch can be informed to reroute the data to a new IP address of machine B. This won't create a problem because the process inside the VM was assigned a virtual IP address through a logical Ethernet card and it remains the same when moved to machine B, only the VM needs to connect to the physical address of the new machine which is hidden from the process.

7.2.8 Case Study: Viruses and Malware

Viruses and malware can be propagated using code mobility. It can be either sender-initiated (proactive viruses go and affect other machines) or receiver-initiated (user clicks on malicious link and the virus code gets downloaded onto host machine).

7.3 Communications in Distributed Systems

This lecture deals with communications between different processes in distributed systems.

In a distributed system there are more than one component that are communicating over the network. Communication between any two components is inherent in any distributed system and the question is what abstraction the system should provide to enable this communication. At a very high level communication can be of two types:

Unstructured Communication It's essentially sharing memory or sharing data structures. Data can be put in shared buffer, processes can come and pick it up. In this manner processes can interact with each other and pass messages between them. Application designer has to decide whether to use shared memory or shared data structures. No network support is required in such communications. But this is not a popular way of message passing in commercial systems.

Structured Communication Most of the communication in distributed system is structured. These constitute of IPCs (Interprocess communications), RPCs or sending explicit message over sockets. Processes on the same machine can use either of the above methods. However, in distributed systems processes are on different machines. To communicate across machines, networking is required to pass messages. In this course mainly Structured Communication will be discussed.

7.3.1 Communication Protocols

This section gives a brief recap of the network protocol stack. Suppose an application on Machine A wants to communicate with an application on Machine B, the message from application needs to be processed through several layers of stack on Machine A, then go over a physical network and then reverse processing takes place in the protocol stack of Machine B, which then delivers the message to the application. Each layer has protocol which it follows to communicate with other layer in the network stack. Different layers in the network stack are discussed below :

Physical Layer: It's essentially the lowest level of the stack (Ethernet, WiFi, 3G etc) which is the medium for communication between devices on network.

Data link Layer: This layer is known as MAC protocol which is used to construct the packet in lowest layer and puts it out on the medium, essentially the Ethernet protocol runs on this layer. Data packets are encoded and decoded into bits. It furnishes transmission protocol knowledge and management and handles errors in the physical layer, flow control and frame synchronization.

Network Layer: Routing and forwarding are functions of this layer, as well as addressing, internetworking, error handling, congestion control and packet sequencing. This layer is responsible for hop-by-hop communication. A data packet sent out by machine A might have to traverse through multiple hops before it reaches its final destination. When packet is delivered at one hop it decides what next has to be taken at this point until it reaches its final destination.

Transport Layer: This layer is responsible for end to end communication. This layer provides transparent transfer of data between end systems, or hosts, and is responsible for end-to-end error recovery and flow control (e.g. rate of transmission etc.). It ensures complete data transfer.

Application Layer: This layer supports application and end-user processes. This layer provides application services for file transfers, e-mail, and other network software services. Telnet, HTTP and FTP are applications that exist entirely in the application level.

7.3.2 Layered Protocols

The actual message which needs to be sent to a destination is called *Payload*, as the message traverses down the network stack each layer processes the message and adds a headers to the message packet. For example, *http* is an application layer protocol, thus http header may be added in front of the payload. In Transport Layer, TCP will add a header to the payload. In Network Layer, IP will add a header which denotes the source and destination addresses of the machines which is used at each hop to identity where the packet came from and where it is headed to according to which routing of that packet is done. In DataLink layer, suppose Ethernet is being used to transmit the message over network, then a Ethernet header is added to the payload. At the trailer end, there can be a checksum which is used to check for errors in transmission. At the receiving end, each layer analyses the header and then strips it off and hands it to the layer above it. Receiver application layer receives the original message passed from sender application layer.

7.3.3 Middleware Protocols

In distributed systems environment, the middleware layer sits in between Application Layer and the Transport Layer or the Operating System. The middleware protocol provides higher level service to application which is even much more advanced than session and presentation layer. For example, Multicast, Broadcast, Unicast services are provided by the middleware. Application just needs to send a packet for broadcast to the middleware and then later takes care of sending the message to the network. Essentially middleware abstracts the application from the TCP / IP and other networking aspects of the distributed system. This will be discussed more in upcoming classes.

7.3.4 Client-Server TCP

In a typical client server architecture, client constructs a request and sends it over to the server, server processes the request at its end and sends back the response to the client. The diagram in the slides show how message passing is done using TCP / IP. For a client to send a message to the server, the client has to establish a TCP connection to the server first, i.e. a 3-way handshake is required to establish a TCP connection. The client sends a SYN packet to the server, which means that the client wants to establish a TCP connection with the server. Server in turn sends back a SYN and a ACK(SYN) packet back to the client. ACK(SYN) is the acknowledgement sent from the server to the client and SYN is sent to the client to denote that server wants to setup a connection with it. The client then acknowledges the SYN request of server by sending a ACK(SYN) message back to the server. Thus this is a 3-way handshake mechanism to establish a TCP connection. Only after this TCP connection have been setup, the client can send a request message to the server. One the client sends the request to the server, it send a message FIN to the server denoting that the client has finished sending the message to the server. Now server processes the request and while it is processing the request it sends back a ACK(req + FIN) to denote that it has received the request and finish message. Once the request is processed in the server the answer is send back to the client and then a FIN message is sent to the client to denote end of reply. The client then acknowledges back to the server by sending ACK(FIN) message that it has received the answer. The last 3 messages were to tear down the TCP connection. Thus we see that in total 9 packets were sent for the whole process which in turn results in overhead.

To overcome this overhead, Client-Server TCP is used to optimize the entire process.Essentially in this the messages are batched together to reduce the overhead. The first packet sent from client is (SYN, request, FIN) , which means, that client wants to setup TCP connection with the server, the request is embedded in the same packet and then the client says it is done sending the message. The server then processes the request and sends back SYN,ACK(FIN),answer, FIN which means that it wants to setup a connection with the client, then acknowledge clients Finish request, answer to the request and then finish message from the server side. Then client just send ACK(FIN) which means that it has received the answer from the server and TCP connection can be torn down now. This architecture was merely a proposal but in real it is seldom used. This architecture is most useful for sending one message and receiving one reply from server. Otherwise in general, the 1st method of sending messages is used.