CMPSCI 677 Operating Systems **Spring 2013**

Lecture 7: February 13

Lecturer: Prashant Shenoy *Scribe: Siddharth Gupta*

7.1 Migration Models Continued

Code Migration The entire code is migrated to the new machine and the execution is started from the start. There are two ways in which this can be done:-

- The code can be copied from one machine to another and the execution is started as a new process.
- The code can be loaded by already running process and then executed from within the process. For example, browser process downloads the code of Java Applet to its machine first and then runs the Applet process.

Process Migration In this case a process that was executing on a machine A is moved to a new machine B for processing from the same point it left in machine A. This can be implemented in two ways:-

- **Remote Cloning -** The process is halted in existing machine and entire process is cloned to a new machine, the process on 1st machine is terminated and then processing on 2nd machine starts.
- Copy the state of the process, RAM pages and other resources to disk and load the disk on a new machine.

7.1.1 Models of Migration

As discussed previously, mobility is of two types, strong and weak mobility, and each of them can be sender and receiver initiated mobility, thus combinations of these lead to 8 different migration models.

7.1.2 Resource 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 quite a tedious task as many constraints are involved which are discussed below. First we need to discuss different types of process to resource bindings

- If the process running on system A was accessing any resources such as opened a file for reading / writing or had opened a socket tied to I/P address then such resources need to be migrated along with the process to the new System B, since process will still need them while executing at B. **Binding by Identifier** is the most 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. To resolve the problem of socket, tunneling can be created between machine A and B, and the machine received on the port for that particular process on A can be forwarded to B, but it is easily said than done.

- **Binding by Value**, suppose the process was running in JVM2.0 environment in system A, thus when it migrates to system B, it will expect JVM2.0 environment to be present to continue execution. Thus when process migrates to B, a reference to the JVM2.0 present on B has to be provided to the process.
- **Binding by Type**, This is the loosest form of binding. Suppose the process was accessing hardware devices such as a network printer, then when it moves to machine B, this particular resource needs to be rebound to the process so that it can access the printer.

Now we need to discuss if at all 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, now moving these resources to new machine B is possible but quite a grueling task. Thus it is preferable to create global references for such resources that can be accessed over network.
- **Fixed Resources**, are like switches or hardware devices which cant be moved. For these generally global referencing technique is used. For example, open socket connections can be accessed through tunneling.

Different combinations of the bindings produce different type of complexity and each has to be handled in a different way. Process migration might not be a good idea in all scenarios and can lead to bad results, thus feasible process migrations should be carried out.

7.1.3 Migration in Heterogeneous Systems

When a process is moved in a heterogeneous environment, i.e. between machines using different architectures (Intel/ Spark/ 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, these methods are not efficient way of doing migration, these issue can be resolved using Virtual Machines or programs running on JVM since it is platform independent.

7.1.4 Virtual Machine Migration

This is by far the most widely used method for migration till date. VMs such as VMware, Parallels have a feature to suspend them, in this scenario, the process running in VM is paused and VM image is saved on the disk. This image can now be loaded in a totally different machine with an entirely different architecture and when VM is resumed the process inside that starts executing from the point where it was paused.

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 Iterative copying of memory state, in this, while the VM is running on machine A the memory state of the processes running in VM is copied to the new machine B, and once the copy is completed, the VM in machine B can start executing and 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 old machine A to new machine B while process is still being executed in A. Thus during the duration of copy, process in machine A would have made some changes to RAM pages and now those become redundant in machine B. Thus to overcome this, iterative copying of dirty pages is done until only a fraction of pages remain, which can be copied during the time of switch. Thus smoothing is achieved during live migration.

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. Now this wont create a problem because process inside the VM was assigned a virtual IP address and it can remain same when moved to machine B, only the VM needs to connect to the physical address of new machine which is hidden from the process.

7.1.5 CASE STUDY VIRUS and MALWARE

Use of code migration has been done to spread virus and malware on systems. It can be 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).

7.2 SERVER DESIGN ISSUES

7.2.1 SERVER DESIGN

There are two types of server design choices:-

Iterative - It is a single threaded server which processes the requests in a queue. Once it is processing a request it keeps on adding incoming requests in the 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 anytime on the server.

Concurrent or Multi Threaded 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. There is also one more flavor to multi threaded server, unlike the above case where new thread is spawned every time a new process comes in, there is a pool of already spawned threads in the server which are ready to serve the requests. A dispatcher or scheduler thread handles this pool of pre-spawned threads.

7.2.2 Advantages and Disadvantages of Pre-Spawned threads vs Threads Spawned dynamically

- When thread is spawned dynamically, the server can process any number of incoming requests since there is no limit on number of threads to be spawned which is not the case in Pre-spawned threads. However dynamic spawning of threads might lead to performance issues since huge number of requests can exhaust the server resources and thus slow the overall performance of the system.
- In pre-spawned threads scenario, if large number of requests come in which exhausts the worker thread pool, then new requests will have to wait for processing until one of the worker thread becomes free.
- In dynamic spawning of thread there is an overhead of creation and deletion of threads (system calls, resource allocation) but it doesnt apply for Pre-spawned threads since this overhead is only during server initialization and threads lasts for lifetime of server.

Thus we see that there is good and bad to both the cases. The server design can be optimized according to the system resources available for it. For example, Apache server varies the number of threads to be spawned dynamically depending on the system resources. The servers initially start with a small pool of pre-spawned threads, and if that gets exhausted it generates another pool of threads, in this case there wont be large number of idle pre-spawned threads in the system and also it will keep in check the availability of system resources.

7.2.3 Server Location

To communicate with the server, the client should know the address of the server and the port on which it is listening on. There are several ways in which client can know server information. The most trivial being hardcoding the server info in the client. But this is not a scalable mechanism. Another approach is, when the server comes up, it appends its address to a common well-known directory service. Thus a client wanting to connect to a server will look for address of that server in the directory and connect to it. Examples of directory service servers are, LDAP or port mapper in UNIX.

How will client know the location of directory service? The address of the server can be setup in the system settings of the client manually by the system administrator but not hardcoded in it. Second approach would be to broadcast the address of directory service in the network at periodic intervals and the client can listen to such broadcasts and subscribe to services according to its needs.

7.2.4 Stateful or Stateless Servers

Stateful servers are those, which keep a list of their connected clients, for example, an e-commerce web-server would like to keep a list of clients connected to it along with their session details such as number of items in shopping cart.

Stateless servers on the other hand dont keep any information about the connected clients. In this case, the client will keep its own session information, for example, the e-commerce website may set some cookies in the system to maintain the state of shopping cart etc.

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.

7.2.5 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.

- **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.