Parallel & Distributed System

Servers

Md. Biplob Hosen

Lecturer, IIT-JU

Email: biplob.hosen@juniv.edu

Contents

- **Servers**
  - ✓ General Design Issues
  - ✓ Object Servers
  - ✓ Example: The Apache Web Server

- Reference Books
  - ✓ Distributed Systems: Principles and Paradigms, 3rd Edition by Andrew S. Tanenbaum & Maarten van Steen, Publisher: Pearson Prentice Hall **[CH-03]**.

General Design Issues of Servers

- A server is a **process i**mplementing a specific service on behalf of a collection of clients. Each server is organized in the same way: it waits for an incoming request from a client and subsequently ensures that the request is taken care of, after which it waits for the next incoming request.

Concurrent versus Iterative Servers:

- There are several ways to organize servers. In the case of an iterative server, the server itself handles the request and, if necessary, returns a response to the requesting client.
- A concurrent server does not handle the request itself, but passes it to a separate thread or another process, after which it immediately waits for the next request. A multithreaded server is an example of a concurrent server.
- An alternative implementation of a concurrent server is to fork a new process for each new incoming request. This approach is followed in many Unix systems. The thread or process that handles the request is responsible for returning a response to the requesting client.

Continue…

Contacting a Server: End points-

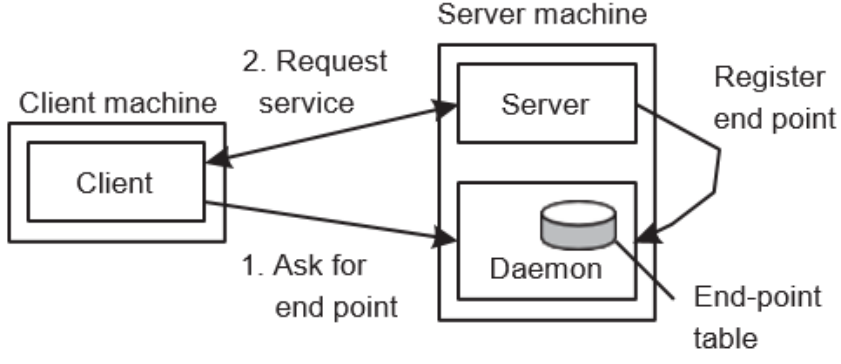
- Another issue is where clients contact a server.
- In all cases, clients send requests to an end point, also called a port, at the machine where the server is running.
- Each server listens to a specific end point.
- How do clients know the end point of a service?
- One approach is to globally assign end points for well-known services.
- For example, servers that handle Internet FTP requests always listen to TCP port 21.
- Likewise, an HTTP server for the World Wide Web will always listen to TCP port 80.
- These end points have been assigned by the Internet Assigned Numbers Authority (IANA).
- With assigned end points, the client needs to find only the network address of the machine where the server is running. Name services can be used for that purpose.

Continue…

Contacting a Server: End points-

- There are many services that do not require a preassigned end point.
- For example, a time-of-day server may use an end point that is dynamically assigned to it by its local operating system.
- In that case, a client will first have to look up the end point. One solution is to have a special daemon running on each machine that runs servers.
- The daemon keeps track of the current end point of each service implemented by a co-located server.
- The daemon itself listens to a well-known end point. A client will first contact the daemon, request the end point, and then contact the specific server, as shown.

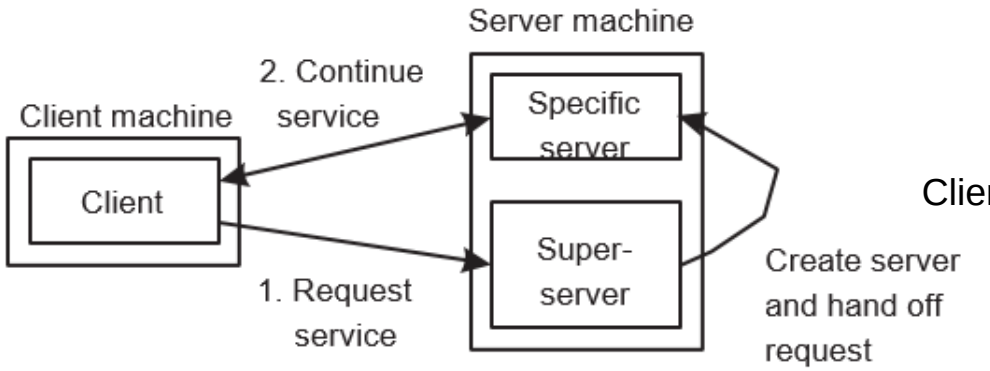
Dynamically assigning an end point



(a) Client-to-server binding using a daemon

Continue…

- It is common to associate an end point with a specific service, although implementing each service by a separate server may be a waste of resources.
- For example, in a typical Unix system, it is common to have lots of servers running simultaneously, with most of them passively waiting until a client request comes in.
- Instead of having to keep track of so many passive processes, it is often more efficient to have a single superserver listening to each end point associated with a specific service, as shown in the figure.
- For example, the inetd daemon in Unix listens to a number of well-known ports for Internet services. When a request comes in, the daemon forks a process to handle it. That process will exit when finished.



Client-to-server binding using a superserver

Dynamically assigning an end point

Continue…

Interrupting a Server:

- Another issue that needs to be taken into account when designing a server is whether and how a server can be interrupted.
- For example, consider a user who has just decided to upload a huge file to an FTP server.
- Then, suddenly realizing that it is the wrong file, he wants to interrupt the server to cancel further data transmission.
- There are several ways to do this. One approach that works only too well in the current Internet is for the user to abruptly exit the client application (which will automatically break the connection to the server), immediately restart it, and pretend nothing happened.
- The server will eventually tear down the old connection, thinking the client has probably crashed.

Continue…

- A much better approach for handling communication interrupts is to develop the client and server such that it is possible to send out-of-band data, which is data that is to be processed by the server before any other data from that client.
- One solution is to let the server listen to a separate control end point to which the client sends out-of-band data, while at the same time listening (with a lower priority) to the end point through which the normal data passes.
- Another solution is to send out-of-band data across the same connection through which the client is sending the original request.
- In TCP, for example, it is possible to transmit urgent data. When urgent data are received at the server, the latter is interrupted (e.g., through a signal in Unix systems), after which it can inspect the data and handle them accordingly.

Continue…

Stateless versus Stateful Servers:

- A final, important design issue, is whether or not the server is stateless.
- A stateless server does not keep information on the state of its clients, and can change its own state without having to inform any client.
- A Web server is stateless. It responds to incoming HTTP requests, which can be either for uploading a file to the server or for fetching a file. When the request has been processed, the Web server forgets the client completely.
- Likewise, the collection of files that a Web server manages, can be changed without clients having to be informed.
- Note that in many stateless designs, the server actually does maintain information on its clients, but crucial is the fact that if this information is lost, it will not lead to a disruption of the service offered by the server.
- For example, a Web server generally logs all client requests. This information is useful, for example, to decide whether certain documents should be replicated, and where they should be replicated to.

Continue…

- A particular form of a stateless design is where the server maintains what is known as **soft state**. In this case, the server promises to maintain state on behalf of the client, but only for a limited time.
- After that time has expired, the server falls back to default behavior, thereby discarding any information it kept on account of the associated client.
- An example of this type of state is a server promising to keep a client informed about updates, but only for a limited time. After that, the client is required to poll the server for updates.
- In contrast, a stateful server maintains persistent information on its clients which means that the information needs to be explicitly deleted by the server.
- A typical example is a file server that allows a client to keep a local copy of a file, even for performing update operations. Such a server would maintain a table containing (client, file) entries.

Continue…

- This approach can improve the performance of read and write operations as perceived by the client.
- Performance improvement over stateless servers is often an important benefit of stateful designs.
- However, the example also illustrates the major drawback of stateful servers. If the server crashes, it has to recover its table of (client, file) entries, or otherwise it cannot guarantee that it has processed the most recent updates on a file.
- In general, a stateful server needs to recover its entire state as it was just before the crash.
- Enabling recovery can introduce considerable complexity.
- In a stateless design, no special measures need to be taken at all for a crashed server to recover.
- It simply starts running again, and waits for client requests to come in.

Continue…

- One should actually make a distinction between session state and permanent state.
- The example above is typical for session state: it is associated with a series of operations by a single user and should be maintained for a some time, but not indefinitely.
- As it turns out, session state is often maintained in three-tiered client-server architectures, where the application server actually needs to access a database server through a series of queries before being able to respond to the requesting client.
- The issue here is that no real harm is done if session state is lost, provided that the client can simply re-issue the original request.
- This observation allows for simpler and less reliable storage of state.
- What remains for permanent state is typically information maintained in databases, such as customer information, keys associated with purchased software, etc.

Continue…

- However, for most distributed systems, maintaining session state already implies a stateful design requiring special measures when failures do happen and making explicit assumptions about the durability of state stored at the server.
- When designing a server, the choice for a stateless or stateful design should not affect the services provided by the server.
- For example, if files have to be opened before they can be read from, or written to, then a stateless server should one way or the other mimic this behavior.
- A common solution is that the server responds to a read or write request by first opening the referred file, then does the actual read or write operation, and immediately closes the file again.
- In other cases, a server may want to keep a record on a client's behavior so that it can more effectively respond to its requests.

Continue…

- For example, Web servers sometimes offer the possibility to immediately direct a client to his favorite pages.
- This approach is possible only if the server has history information on that client.
- When the server cannot maintain state, a common solution is then to let the client send along additional information on its previous accesses.
- In the case of the Web, this information is often transparently stored by the client's browser in what is called a cookie, which is a small piece of data containing client-specific information that is of interest to the server. Cookies are never executed by a browser; they are merely stored.
- The first time a client accesses a server, the latter sends a cookie along with the requested Web pages back to the browser, after which the browser safely tucks the cookie away.
- Each subsequent time the client accesses the server, its cookie for that server is sent along with the request.

Object Servers

- Object servers are needed for distributed objects.
- The important difference between a general object server and other servers is that an object server by itself does not provide a specific service. Specific services are implemented by the objects that reside in the server.
- The server provides only the means to invoke local objects, based on requests from remote clients. So, it is easy to change services by simply adding and removing objects.
- An object server acts as a place where objects live. An object consists of two parts: data representing its state & the code for executing its methods.
- Whether or not these parts are separated, or whether method implementations are shared by objects, depends on the object server.
- Also, there are differences in the way an object server invokes its objects.
- For example, in a multithreaded server, each object may be assigned a separate thread, or a separate thread may be used for each invocation request.

Continue…

- For an object to be invoked, the object server needs to know which code to execute, on which data it should operate, whether it should start a separate thread to take care of the invocation, and so on.
- A simple approach is to assume that all objects look alike and that there is only one way to invoke an object.
- Unfortunately, such an approach is generally inflexible and often unnecessarily constrains to developers of distributed objects.
- A much better approach is for a server to support different policies.
- Consider, for example, a transient object: an object that exists only as long as its server exists, but possibly for a shorter period of time.
- An in-memory, read-only copy of a file could typically be implemented as a transient object.
- Likewise, a calculator could also be implemented as a transient object.
- A reasonable policy is to create a transient object at the first invocation request and to destroy it as soon as no clients are bound to it anymore.

Continue…

- The advantage of this approach is that a transient object will need a server's resources only as long as the object is really needed.
- The drawback is that an invocation may take some time to complete, because the object needs to be created first.
- Therefore, an alternative policy is sometimes to create all transient objects at the time the server is initialized, at the cost of consuming resources even when no client is making use of the object.
- In a similar fashion, a server could follow the policy that each of its objects is placed in a memory segment of its own.
- In other words, objects share neither code nor data.
- Such a policy may be necessary when objects need to be separated for security reasons.
- In the latter case, the server will need to provide special measures, or require support from the underlying operating system, to ensure that segment boundaries are not violated.

Continue…

- The alternative approach is to let objects at least share their code.
- For example, a database containing objects that belong to the same class can be efficiently implemented by loading the class implementation only once into the server.
- When a request for an object invocation comes in, the server need only fetch that object's state and execute the requested method.
- Likewise, there are many different policies with respect to threading. The simplest approach is to implement the server with only a single thread of control.
- Alternatively, the server may have several threads, one for each of its objects.
- Whenever an invocation request comes in for an object, the server passes the request to the thread responsible for that object.
- If the thread is currently busy, the request is temporarily queued.

Continue…

- The advantage of this approach is that objects are automatically protected against concurrent access: all invocations are serialized through the single thread associated with the object.
- Of course, it is also possible to use a separate thread for each invocation request, requiring that objects should have already been protected against concurrent access.
- Independent of using a thread per object or thread per method is the choice of whether threads are created on demand or the server maintains a pool of threads.
- Generally there is no single best policy. Which one to use depends on whether threads are available, how much performance matters, and similar factors.
- Decisions on how to invoke an object are commonly referred to as **activation policies**, to emphasize that in many cases the object itself must first be brought into the server's address space (i.e., activated) before it can actually be invoked.
- Such a mechanism is sometimes called an **object adapter**, or alternatively an **object wrapper**. An object adapter can best be thought of as software implementing a specific activation policy.

Continue…

- An object adapter has one or more objects under its control. Because a server should be capable of simultaneously supporting objects that require different activation policies, several object adapters may reside in the same server.
- When an invocation request is delivered to the server, the request is first dispatched to the appropriate object adapter, as shown in the figure.

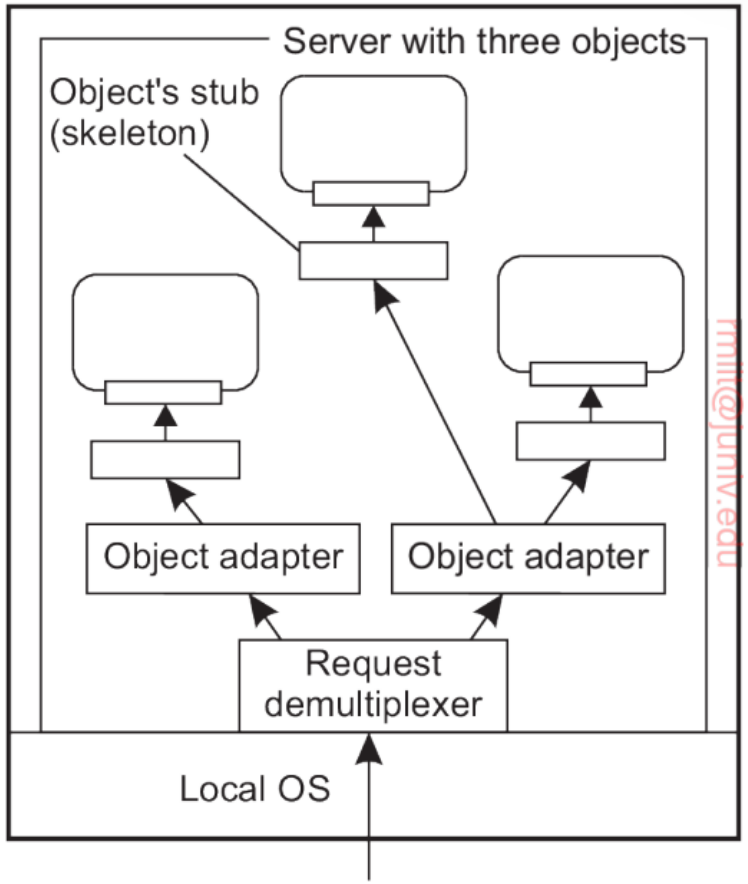


Fig - An object server supporting different activation policies

- The only issue that is important to an object adapter is that it can extract an object reference from an invocation request, and subsequently dispatch the request to the referenced object following a specific activation policy.

Continue…

- As is also illustrated that rather than passing the request directly to the object, an adapter hands an invocation request to the server-side stub of that object.
- The stub, also called a skeleton, is generated from the interface definitions of the object, unmarshals the request and invokes the appropriate method.
- An object adapter can support different activation policies by simply configuring it at runtime.
- For example, in CORBA-compliant systems, it is possible to specify whether an object should continue to exist after its associated adapter has stopped.
- Likewise, an adapter can be configured to generate object identifiers, or to let the application provide one.
- As a final example, an adapter can be configured to operate in single-threaded or multithreaded mode as we explained above.

Example: The Apache Web Server

- An Apache Web server is an example of a server that balances the separation between policies and mechanisms. It is also an extremely popular server, estimated to be used to host approximately 50% of all Web sites.
- Apache is a complex piece of software, and with the numerous enhancements to the types of documents that are now offered in the Web, it is important that the server is highly configurable and extensible, and at the same time largely independent of specific platforms.
- Making the server platform independent is realized by essentially providing its own basic runtime environment, which is then subsequently implemented for different operating systems.
- This runtime environment, known as the Apache Portable Runtime (APR), is a library that provides a platform independent interface for file handling, networking, locking, threads, and so on.
- When extending Apache, portability is guaranteed, provided that only calls to the APR are made and that calls to platform-specific libraries are avoided.

Continue…

- From a certain perspective, Apache can be considered as a completely general server tailored to produce a response to an incoming request.
- Of course, there are all kinds of hidden dependencies and assumptions by which Apache turns out to be primarily suited for handling requests for Web documents.
- For example, Web browsers and servers use HTTP as their communication protocol.
- HTTP is virtually always implemented on top of TCP, for which reason the core of Apache assumes that all incoming requests adhere to a TCP-based connection-oriented way of communication.
- Requests based on UDP cannot be handled without modifying the Apache core.
- However, the Apache core makes few assumptions on how incoming requests should be handled.
- Its overall organization is shown in the following figure.

Continue…

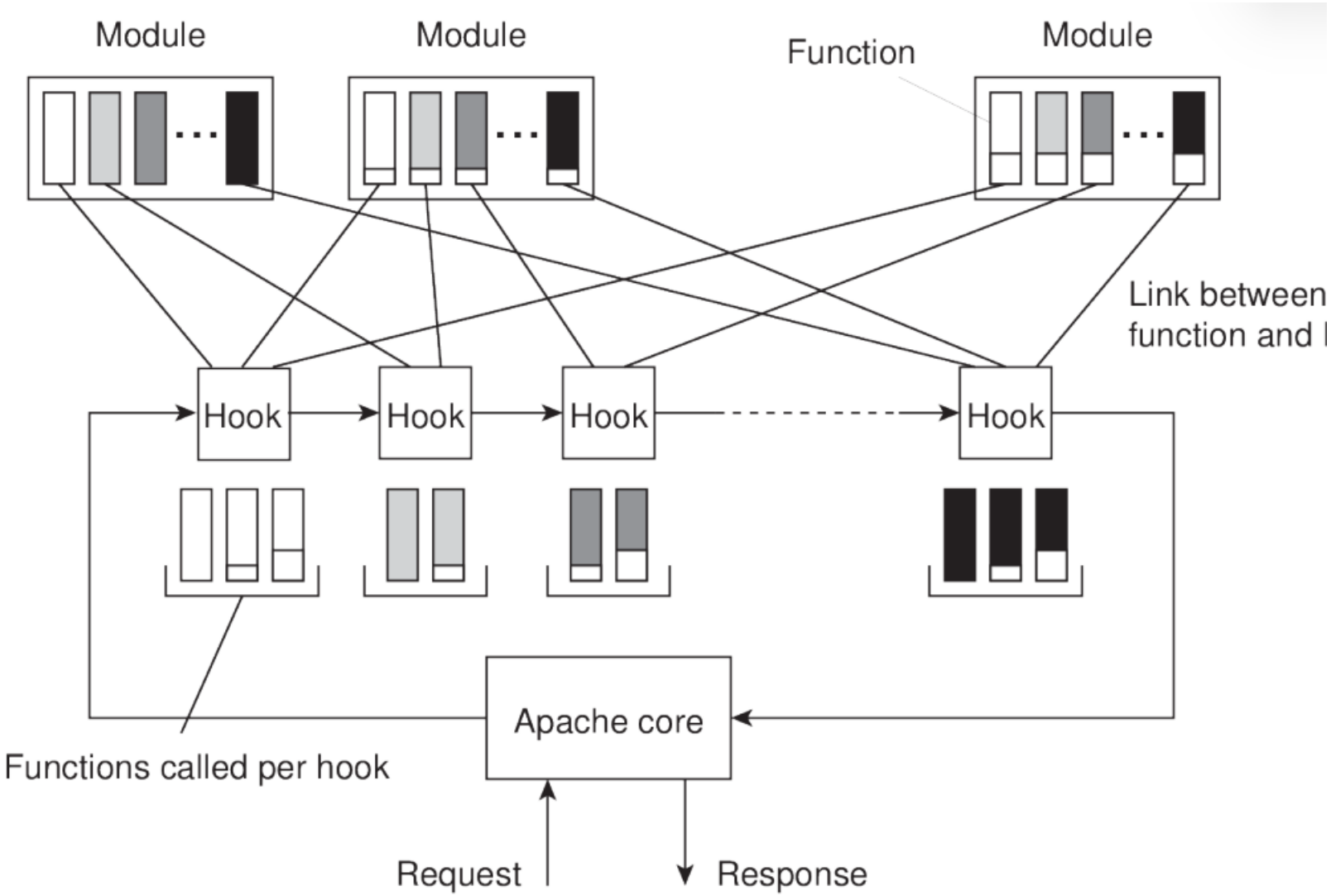


Figure The general organization of the Apache Web server

Continue…

- Fundamental to this organization is the concept of a **hook**, which is nothing but a placeholder for a specific group of functions.
- The Apache core assumes that requests are processed in a number of phases, each phase consisting of a few hooks.
- Each hook thus represents a group of similar actions that need to be executed as part of processing a request.
- For example, there is a hook to translate a URL to a local file name.
- Such a translation will almost certainly need to be done when processing a request.
- Likewise, there is a hook for writing information to a log, a hook for checking a client's identification, a hook for checking access rights, and a hook for checking which MIME type the request is related to.
- The hooks are processed in a predetermined order.
- Apache enforces a specific flow of control concerning the processing of requests.

Continue…

- The functions associated with a hook are all provided by separate modules.
- Every hook can contain a set of functions that each should match a specific function prototype.
- A module developer will write functions for specific hooks. When compiling Apache, the developer specifies which function should be added to which hook.
- Because there may be many modules, each hook will contain several functions.
- Normally, modules are considered to be mutual independent, so that functions in the same hook will be executed in some arbitrary order.
- However, Apache can also handle module dependencies by letting a developer specify an ordering in which functions from different modules should be processed.

Thank You ☺