DC ASSIGNMENT - 1

Group No-10
Group Members
Priyanka Awatramani (05)
Sneha Lalwani (39)
Sartha Tambe (66)

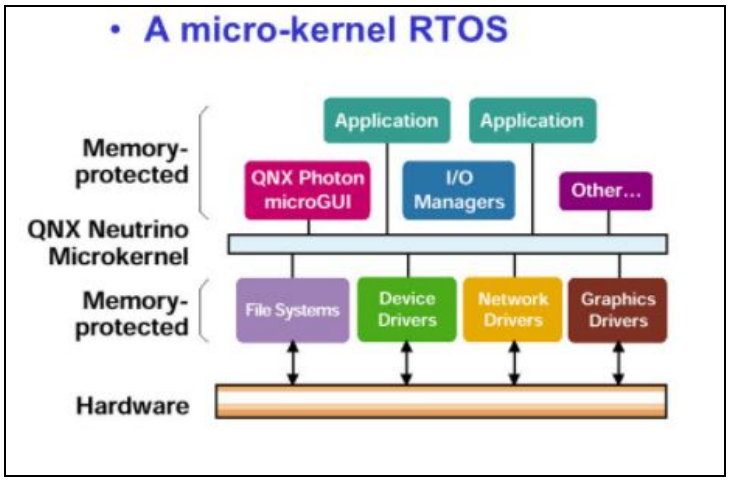
Use of Embedded RPC based service communication for Microkernel RTOS Architecture

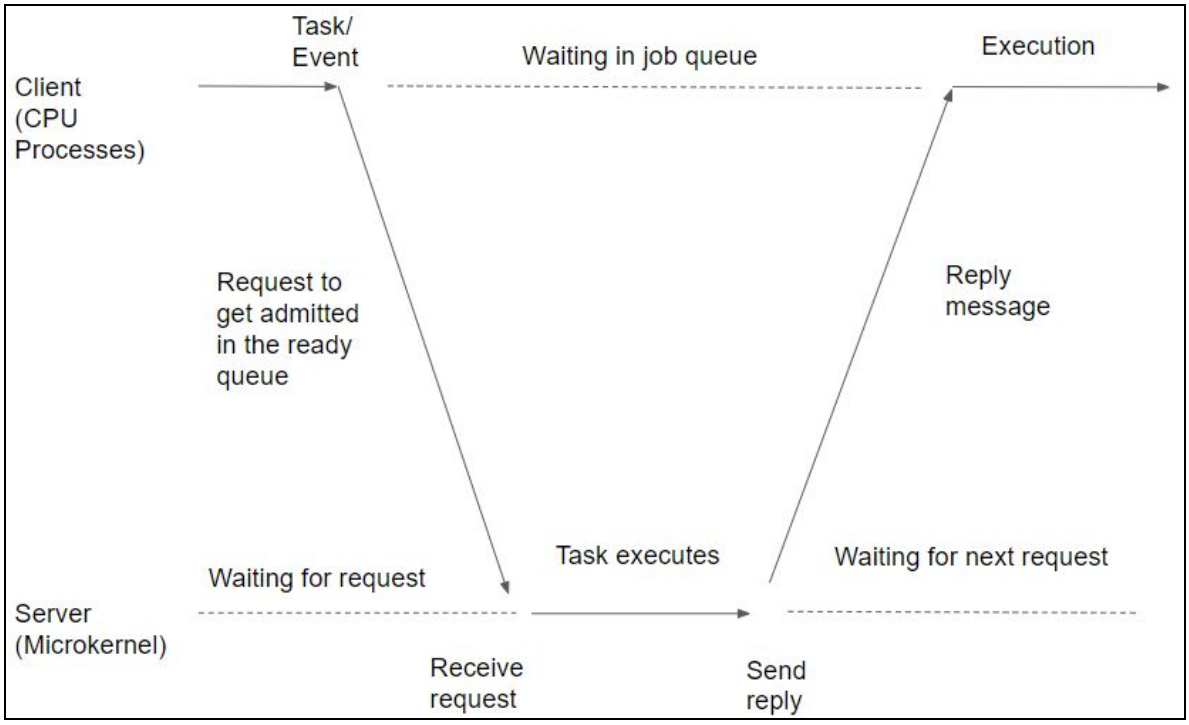
1. Introduction:

Communication in microkernel-based systems is much more frequent than system calls known from monolithic kernels. This can be attributed to the placement of system services into their own protection domains. Communication has to be fast to avoid unnecessary overhead. Also, communication channels in microkernel-based systems are used for more than just remote procedure calls. In distributed systems, it is state of the art to use tools to generate stubs for the communication between components. Remote Procedure Call (RPC) is a set of methods to communicate with two processes which may be in the same computer or different computer. It includes direct & indirect communication, synchronous & asynchronous communication and explicit buffering. Additionally to RPC, microkernel-based systems use communication to signal state between threads with simple messages.

Application: CPU Scheduling (process which allows one process to use the CPU while the execution of another process is on hold due to unavailability of any resource)

2. Present your design:





CPU Scheduling using Remote Procedure Call

3. Explain the working concept of your applications and layered architecture of communication protocol

The kernel is the core part of the operating system, thus it is meant for handling the most important services only. Thus only the most important services are inside the kernel and rest of the OS services are present inside the system application program. The most important service of OS is CPU scheduling. The aim of CPU scheduling is to make the system efficient, fast and fair.In Microkernel architecture, CPU scheduling service is kept in kernel address space. Whenever the CPU becomes idle, the operating system must select one of the processes in the ready queue to be executed. The selection process is carried out by the short-term scheduler (or CPU scheduler). The scheduler selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.

In a Remote Procedure Call, a client invokes a client stub procedure, passing parameters in the usual way. The client stub resides within the client's own address space. The client stub marshalls(packs) the parameters into a message. Marshalling includes converting the representation of the parameters into a standard format, and copying each parameter into the message. The client stub passes the message to the transport layer, which sends it to the remote server machine. On the server, the transport layer passes the message to a server stub, which demarshalls (unpack) the parameters and calls the desired server routine using the regular procedure call mechanism. When the server procedure completes, it returns to the server stub (e.g: via a normal procedure call return), which marshalls the return values into a message. The server stub then hands the message to the transport layer. The transport layer

sends the result message back to the client transport layer, which hands the message back to the client stub. The client stub demarshalls the return parameters and execution returns to the caller.

Our application is going to be an RPC-based application that is going to perform CPU scheduling.

4. Explain and JUSTIFY the communication model (RPC/RMI/CORBA/COM/EJB/gRPC)for your application for the optimal design :
In terms of following:

A. **Client/Server Design : Stateless/ Stateful**
The server of our application will be **stateful** since every processor should know which processor is doing what task to handle failure of an individual processor. The server should remember client data (state) from one request to the next. They may have to, therefore, keep track of which clients have opened which files, current read and write pointers for files, which files have been locked by which clients, etc. When two subsequent calls are made by a client to the Stateful servers, some state information of the service performed for the client is stored by the server process, which is used while executing the next call. The server has to keep the information of the session and other details.

B. **Server Creation Semantic**
Instance-per-session exists for the entire session for which Client & Server interact. It can maintain intercall state information to minimize the overhead involved in server creation and destruction for a client-server session. In RPC (Remote Procedure Call) a client process lies in a server process that is totally independent of the client process. Since a server process is independent of a client process that makes a remote procedure call to it. Server processes may either be created and installed before their client processes or be created on a demand basis.

C. **Persistent and Transient Communication**
The server of our application is **Transient** since once a task is completed, the processor is free to do other tasks. The server will exist only for the time server is in existence. A message is stored only as long as the sending and receiving applications are executing (the sender and the receiver must be executing in parallel).

D. **Synchronous / Asynchronous Communication also (Request, Request/Reply,Request/Reply/Ack, Callback nature, Receipt based, Delivery based etc.)**
The server could be **both synchronous** and **asynchronous**. If all processors are doing the same task or dependent task then synchronisation is needed whereas if all of the processor doing independent tasks then it would be asynchronous.
Our system will use a **Request/Reply** mechanism. RPC is especially well suited for client-server (e.g. query-response) interaction in which the flow of control alternates between the caller and callee.

E. Call semantics

The semantics of calling a regular procedure are simple: a procedure is executed exactly once when we call it. With a remote procedure, the exactly once aspect is quite difficult to achieve. RPC systems will generally offer either at least once or at most once semantics or a choice between them. If a function may be run any number of times without harm, it is idempotent (e.g., time of day, math functions, read static data). Otherwise, it is a non idempotent function.

Possibly or maybe: No retry of call; no certainty of results

- At-least once: Retry implemented but no duplicate filtering
- Exactly once: Retry and duplicate filtering are implemented
- Others: At-most once; last-one call semantics etc

F. Concurrent Access to Multiple Servers

Concurrent Access to Multiple Servers can be done using Multithreading. Benefits of eRPC include Multithreading of servers when built with an RTOS.When a server receives a message, it services the message using a separate thread.

G. Serving Multiple Requests simultaneously

Our application has to serve multiple requests simultaneously since the throughput needs to be increased. Multitasking allows many more tasks to be run than there are CPUs. A good way to handle multiple requests efficiently is by using a multi-threaded application where different threads can execute tasks simultaneously. Thus we can handle multiple requests from clients at the same time.

H. Reducing Per-Call Workload of Servers

The workload on servers is reduced by registers and cache as they are faster to access since they are in the processor. Caching reduces the workload of the remote web server by spreading the data widely among the proxy caches over the WAN.The use of registers can be more effective in reducing the bus traffic than cache memory of the same size.

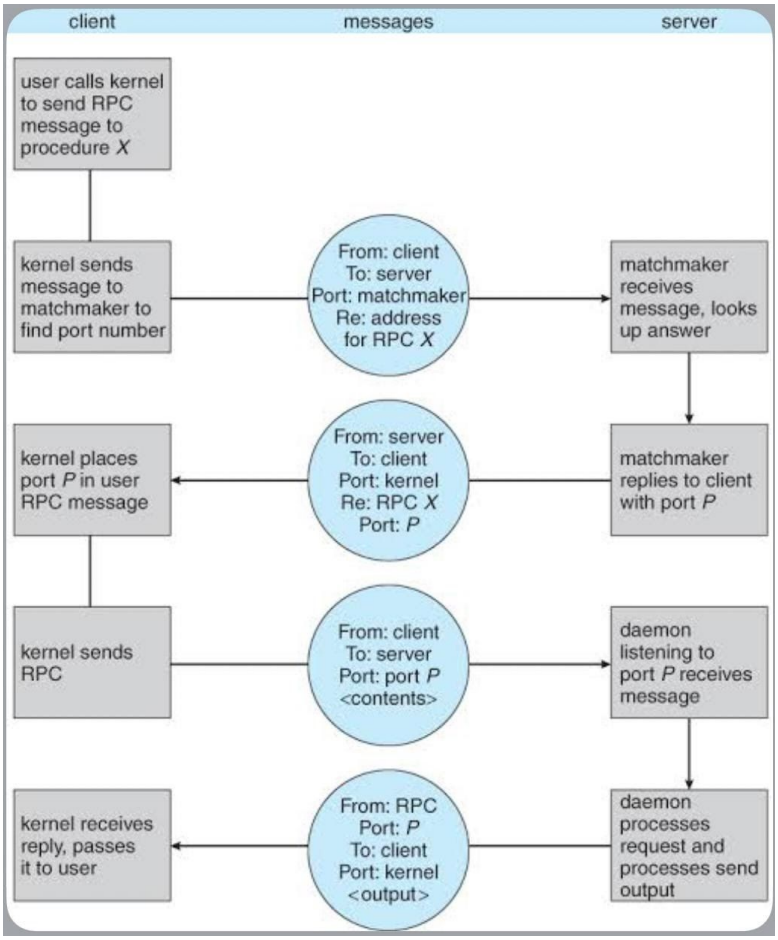
I. Reply Caching of Idempotent Remote Procedures

The RPC object will be used to store cache configuration. It will store a timedelta for the cache duration and a list of context keys from which the request depends. On the answer of a request with cache information, the server will add in the headers the context keys and the duration. The client will store in a size limited dictionary the result of a request if the cache headers are available and the expiration date. It will construct the dictionary key using the arguments and the context keys converted into a hashable object. On the next request, the client will first construct the key and check it exists in the cache and if it is still valid. Then it will use the cached result if possible or it will make the request to the server.

J. Proper Selection of Timeout Values

The timeout values will depend on the processor and also on the system designer. If not configured, default timeout will be 60 seconds.

K. Proper Design of RPC Protocol Specification



Embedded RPC-based Service Communication