CASE STUDY NO : 14

Case Study On :- Study of process management and memory management in Mach OS.

Goals of Mach -

1) Providing a base for building other operating systems (e.g., UNIX)
2) Supporting large sparse address spaces.
3) Allowing transparent access to network resources.
4) Exploiting parallelism in both the system and the applications.
5) Making Mach portable to a large collection of machines.

Process Management in Mach -

Process Management in Mach deals with processes, threads and scheduling.

Processes -

- The process port is used to communicate with the kernel.
- The bootstrap port is used for initialization when a process starts up.
- The exception port is used to report exceptions caused by the process. Typical exceptions are division by zero and illegal instruction executed.

- The registered ports are normally used to provide a way for the process to communicate with standard system servers.
- A process can be runnable or blocked.

Threads -

The active entities in Mach are the threads. They execute instructions and manipulate their registers and address spaces. Each thread belongs to exactly one process. A process cannot do anything unless it has one or more threads. All the threads in a process share the address space and all the process - wide resources. Nevertheless, threads also have private per- thread resources. One of these is the thread port, which is analogous to the process port. Each thread has its own thread port, which it uses to invoke thread- specific Kernel services.

Scheduling -

Mach scheduling has been heavily influenced by its goal of running on multiprocessoes. Since a single- processor system is effiectively a special case of a multiprocessor. The CPUs in a multiprocessor can be assigned to processor ~~setps~~ ~~sts~~ sets by software. Each CPU belongs to exactly one processor set. Threads can also be assigned to processor sets by software.

Memory Management in Mach -

- Mach has a powerful, elaborate, and highly flexible memory management system based on paging.
- The code of Mach's memory management is split into three parts. The first part is the pmap module, which runs in the kernel and is concerned with managing the MMU.
- The second part, the machine - independent kernel code, is concerned with processing page faults, managing address maps, and replacing page.
- The third part of the memory management code runs as a user process called a memory manager. It handles the logical part of the memory management system, primarily management ~~system~~, of the backing store.
- Sharing plays an important role in Mach. No special mechanism is needed for the threads in a process to share objects; they all see the same address space automatically.
- The kernel and the memory manager communicate through a well-defined protocol, making it possible for users to write their own memory mangers.

Virtual Memory -

- The conceptual model of memory that Mach user processes see is a large, linear virtual address space. The address space is supported by paging.

- A key concept relating to the use of virtual address space is the memory object. A memory object can be a file or other, more specialized data structure.
- In reality, there is a great deal more to say. Mach provides a great deal of fine-grained control over how the virtual pages are used.
- To start with, the address space can be used in a sparse way. For example, a process might have dozens of sections of the virtual address space in use.