Responsibilities of Operating System

Process Management:

• Scheduling processes and threads on the CPUs

• Creating and deleting both user and system processes

• Suspending and resuming processes

• Providing mechanisms for process synchronization

• Providing mechanisms for process communication

Memory Management:

• Keeping track of which parts of memory are currently being used and who is using them.

• Deciding which processes (or parts of processes) and data to move into and out of memory.

• Allocating and deallocating memory space as needed.

Storage Management:

• Creating and deleting files

• Creating and deleting directories to organize files

• Supporting primitives for manipulating files and directories

• Mapping files onto secondary storage

• Backing up files on stable (nonvolatile) storage media

Disk Management:

• Free-space management

• Storage allocation

• Disk scheduling

What fork() does and what exec(0 does

Message Passing vs Shared memoey

dtrace

lsmod

**Process Control Block**

Each process in operating system is represented by a process control block which is stored in protected mode in kernel stack area.

It contains the following information:

* **Process State**: New, Ready, Running, Waiting, Terminated
* **Program Counter**: Stores the address of next instruction.
* **CPU registers**: Accumulators, General purpose, Index registers and stack pointers
* **CPU scheduling information**: Process priority, pointers to scheduling queues
* **Memory management information**: Value of base/limit registers, page/segment tables
* **Accounting information**: CPU time used and time limits etc
* **I/O status information:** It includes I/O devices allocated to the process.

Whenever a process is interrupted its state is saved in PCB which is reloaded when process is resumed.

In a process which contains multiple threads, information for each thread is stored in PCB.

PCB is generally of size 1.7Kb/process

Within kernel processes are represented by doubly linked list of *task\_struct.*

**Context Switch**

When CPU encounters and interrupt, it stops the process under execution and based on interrupt executes interrupt service routine by checking out entry in interrupt vector table.

After pid = fork(), the pid of child is 0 while that of parent is >0, while <0 means an error.

The parent process can wait for the child process to terminate by invoking the wait() system call. The PCB of child is not released unless and until parent calls wait() system call.

**Inter-process Communication**

Two modes:

* Shared memory
* Message Passing

Shared memory is fast because message passing involves more system calls. In case of shared memory, system calls are required just to setup shared memory regions, after that there are just normal memory accesses.

The producer-consumer problem can be solved using either of them. In this, producer will first call shm\_open(), ftruncate() and mmap() to map the file descriptor returned by shm\_open() to its address space. mmap() then returns the pointer to that memory area.

Consumer also do shm\_open() in read-only mode and call mmap() to map the shared segment in its address space and finally calls shm\_unlink() to remove the shared object.

Producer:

shm fd=shm open("SHARED\_OBJECT\_NAME",O CREAT|O RDRW,0666);

ftruncate(shm fd,SIZE);

ptr=mmap(0,SIZE,PROT WRITE,MAP SHARED,shm fd,0);

sprintf(ptr,"%s","Hello World");

Consumer:

shm fd=shm open("SHARED\_OBJECT\_NAME",O RDONLY,0666);

ptr = mmap(0,SIZE,PROT READ,MAP SHARED,shm fd,0);

printf("%s",(char\*)ptr);

Processes across systems can employ Sockets to implements Client-Server architecture.

PA socket is identified by the combination of ip address and port. When a client requests to initiate a connection, it is assigned a port by its host computer. The packets travelling between the machines are delivered to the appropriate process based on the port number.

Sockets allow unstructured stream of data to be shared between the machines.

A socket is identified by :

Client IP : Client Port and Server IP : Server Port

To communicate with the server the client opens a new port on its side, it opens a socket on its side on its IP and waits for the connection

As soon as it gets the connection it creates a new socket combining the address of client and its own address

TCP 3 way handshake takes place in the listen function - Here handshake packets are sent in terms of Sync/Sync-Ack/Ack

In these handshakes there is a sequence number that tells whether the packets being sent are in sync or not.

Steps are :

Handshake : Automated process of negotiation that takes place between 2 nodes It dynamically sets the params required for the transmission

Some params are : tranfer rate, coding parity, alphabets,

3-way

A sends the seq number to B (x) : SYNC

B sends its own seq num. (y) and x+1 : SYNC-ACK

A sends the (y+1) : ACK

which is accepted by B and is not responded back

However HTTP can use unreliable protocols such as

The Simple Service Discovery Protocol (SSDP) is a network protocol based on the Internet Protocol Suite for advertisement and discovery of network services and presence information. the User Datagram Protocol (UDP), for example in Simple Service Discovery Protocol listen on UDP and TCP on the same port

UNIX-domain sockets are generally more flexible than named pipes. Some of their advantages are:

* You can use them for more than two processes communicating (eg. a server process with potentially multiple client processes connecting);
* They are bidirectional;
* They support passing kernel-verified UID / GID credentials between processes;
* They support passing file descriptors between processes;
* They support packet and sequenced packet modes.

To use many of these features, you need to use the send() / recv() family of system calls rather than write() / read().

Pipes:

Pipes act as an conduit between the 2 communicating processes. They typically provides one of the simpler ways to communicate. Ordinary pipes are unidirectional.

Pipes are created using pipe(int fd[]) function call which creates 2 pipes one for reading(fd[0]) purpose and another for writing purpose(fd[1]). These pipes can be accessed using regular read() write() system calls. Pipes are used for communication between parent and child.

There are two types of pipe: Named and Unnamed.

For Unnamed pipes, parent-child relationship is required and pipes are ceased to exist once processes are terminated. While for the named pipes, this condition doesn’t hold since it doesn’t need any relationship and it is bidirectional too. Named pipes are referred as FIFO’s in UNIX.

Thread

A thread is basic unit of CPU utilization. It comprises of a stack pointer, program counter, register set and stack. It shares with other threads executing in same process the code segment, data segment, open files and signals.

Advanages of threads

* Responsiveness
* No special setup required sharing of data as in processes.
* Overhead of creation is less than fork() and too s context switch

Asynchronous threading – Parent and child independent

Synchronous threading – Parent waits for child to complete. Helpful especially in cases when parent needs to sum up the results of children.

pthread\_t pid;

pthread\_attr\_t attr;

pthread\_attr\_init(&attr);

pthread\_create(&id, &attr, <function>, <data>);

pthread\_join(pid, NULL);

Thread pools is a good strategy to provide multiprogramming. In this, we maintain the active set of threads in a pool. When a task arrives, it is immediately assigned a thread. Advantages of thread pools :

* Servicing a request with already created thread is faster than creating a new thread.
* A thread pool limits the number of threads that can exist.

pthread\_cancel(thread\_id) cancels a thread named by thread\_id. It can be used in situations when multiple threads are querying from the database. When thread returns the result, other should stop performing operation to which we should sent cancel signal.