What is an Operating System?

OS is a collection of software programs. It is the first program to run on the system.

What are the main functions?

Process management

Memory management

I/O management

File System Management

What is a kernel?

Core of OS which manages core features of OS.

Handles communication between hardware and software.

Services are used through system calls.

Layer of shell wraps around the kernel.

First program loaded on start up.

What are the main functions of the kernel?

Process management

Device management

Memory management

Interrupt Handling.

What are the different types of kernel?

Monolithic

Micro kernel.

What is micro kernel?

User services and Kernel services are implemented in a different address space.

What is a command interpreter?

Interrupt the command input from user through keyboard. It raises an interrupt.

What is a process?

A process is a program that is running instance of a task or job.

A program under execution.

Functions of a process?

Creation and deletion of system processes.

CPU scheduling

Process communication

Synchronization

What is an interrupt?

Interrupt a signal from a device causing context switch.

What is a daemon?

Process that runs in the background.

Daemon processes are indicated with a d appended → httpd

What is a pipe?

A communication mechanism used between two processes.

What is semaphore?

Hardware of Software tag variable which is used as lock to a common resource.

In multi tasking the actions are synchronized by using semaphores.

What kind of operations are possible?

Wait

Signal

What is context switching?

An instance when the CPU switches execution of one process to another.

Critical Section?

Part of code which can be executed only by 1 process at a time.

Mutex?

Lock which protects access to shared data resources.

Before entering a critical section the mutex is locked.

What is Synchronization?

Controlling access to a shared resource which should be available for 2 or more threads or process.

Different types of synchronization?

Mutex

Semaphore

Condition variable

Read write locks

Critical region

What are condition variables?

Access to a shared resource only after a condition is satisfied.

What are read write locks?

Controlling access to a shared resource.

It makes easy when a data has very less write operations and has high read operations.

What is a deadlock?

A process waiting for resources used by other processes of the same group and it never happens.

What are page frames?

Same sized memory divided in the physical memory by the virtual memory.

What is trashing?

Under allocation of pages for a process.

What is a thread?

It is a independent flow of control by a process.

Path of execution within a process.

What is multithreading?

The idea to achieve parallelism by dividing processes into multiple threads.

Process vs Thread

Process	Thread
Run in separate memory space	Threads share same memory space
Process don't share resources with other processes	Threads share resources with other threads, code section, data section and OS resources.

Disadvantages of threads?

Cannot be re used

They correct the address space of process.

Synchronization to concurrent read write access to memory.

What are the different types of threads?

User Level	Kernel Level
User threads are implemented by users.	Kernel threads are implemented by OS.
Java Thread and POSIX	Windows Solaris

What is a compiler?

Source code into object code.

What is a driver?

Software interface of a hardware.

What does bit size of cpu denote?

No of bytes of info a cpu can access from ram.

RAM?

Temporary storage stores program data and volatile.

Static Ram: Bit of data stored by a combination of 6 transistors.

DRAM: A pair of transistor and capacitor. Data processing on edges of the pulse.

ROM?

Static data info is stored permanently access speed is slow.

BIOS info or Bootstrap

Process Synchronization?

2 types of processes

Independent Processes: Execution of one process does not affect the execution of other processes.

Cooperative Processes: Execution of one process affects the execution of other.

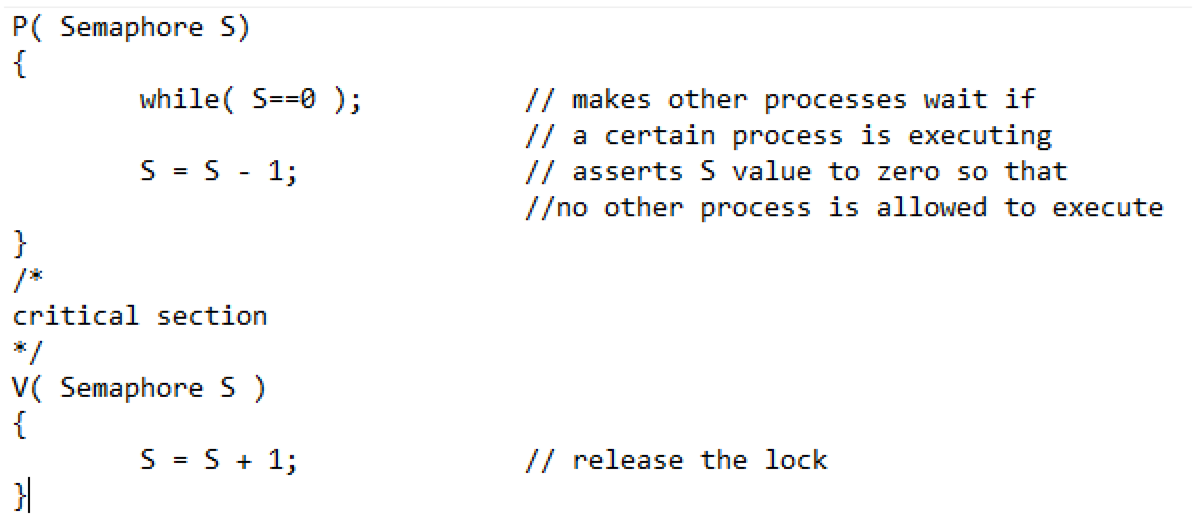
Process synchronization occurs in the case of co-operative process.

Semaphore?

Code logic use to resolve the critical section problem.

Steps:

1. Initially S is set to 1.
2. A process ready to execute sets S value to zero, thereby not allowing other processes to execute.
3. After execution of critical section, it re initializes S value to 1.



```
P( Semaphore S)
{
        while( S==0 );          // makes other processes wait if
                                // a certain process is executing
        S = S - 1;              // asserts S value to zero so that
                                //no other process is allowed to execute
}
/*
critical section
*/
V( Semaphore S )
{
        S = S + 1;              // release the lock
}
```

Mutex Lock

```
Peterson's Solution for Mutual Exclusion

Conidering two processes with thread id 0 and 1

void lock_init()
{
    flag[0] = flag[1] = 0;      // Initialize lock by reseting the
                                // desire of both the threads to acquire the locks.
    turn = 0;
}

void lock(int self)
{
        flag[self] = 1;         // Set flag[self] = 1 saying you want to acquire lock

        turn = 1-self;          // But, first give the other thread the chance to
                                // acquire lock

        while (flag[1-self]==1 && turn==1-self) ;       // Wait until the other thread looses the desire
                                                        // to acquire lock or it is your turn to get the lock.
}

void unlock(int self)
{
        flag[self] = 0;         // You do not desire to acquire lock in future.
                                // This will allow the other thread to acquire
                                // the lock.

}
```

How to create a child process?

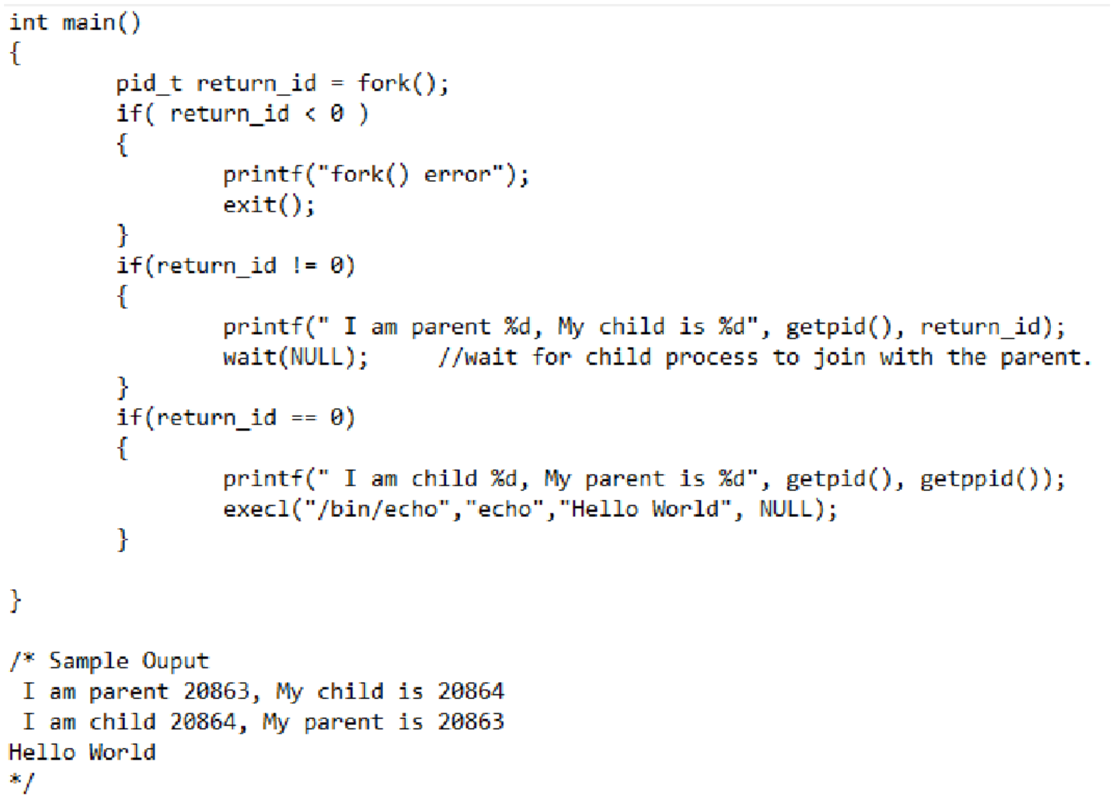
Parent process creates the child process by calling the fork().

Fork() will return

Child process pid → 0

Parent process → Child Pid

fork()	→ Create child process
getpid()	→ Get the process id of currently executing task.
getppid()	→ Gets the parent pid of the child process
wait(NULL)	→ Parent process waits for the execution of child process
execl()	→ executes the commands in the argument list



```
int main()
{
        pid_t return_id = fork();
        if( return_id < 0 )
        {
                printf("fork() error");
                exit();
        }
        if(return_id != 0)
        {
                printf(" I am parent %d, My child is %d", getpid(), return_id);
                wait(NULL);     //wait for child process to join with the parent.
        }
        if(return_id == 0)
        {
                printf(" I am child %d, My parent is %d", getpid(), getppid());
                execl("/bin/echo","echo","Hello World", NULL);
        }

}

/* Sample Ouput
 I am parent 20863, My child is 20864
 I am child 20864, My parent is 20863
Hello World
*/
```

Allocation of program variables on memory stack

4 parts of memory from top to bottom

Heap

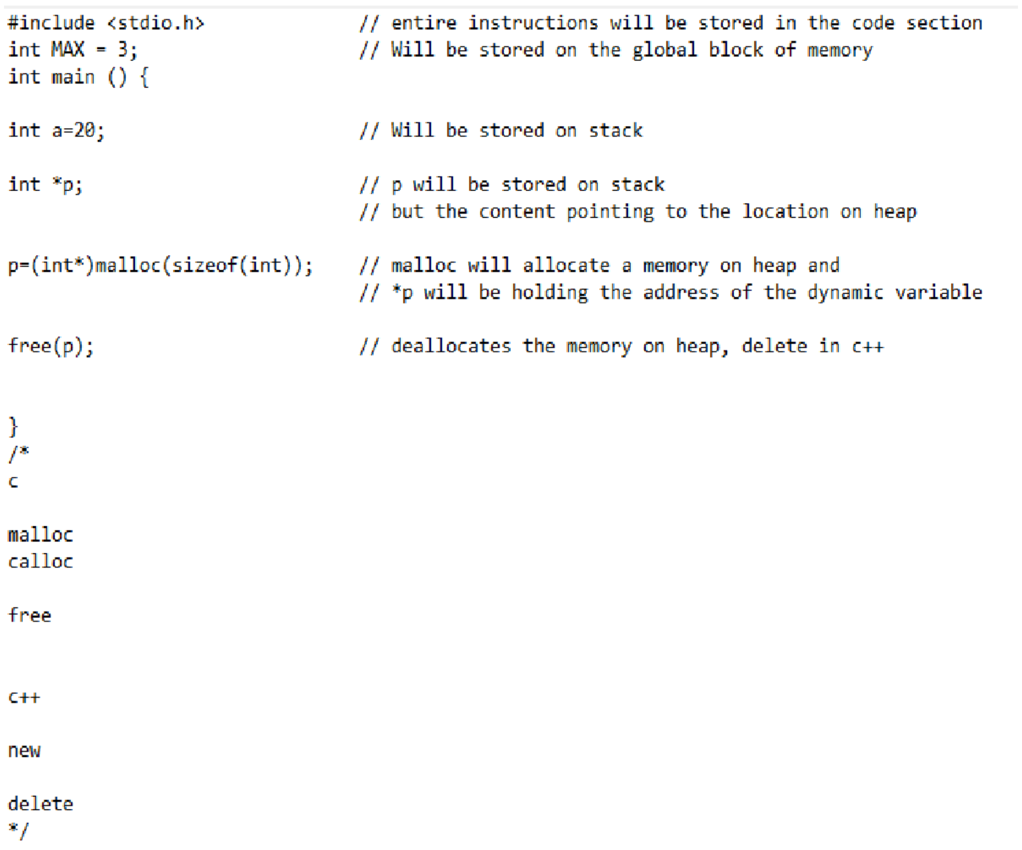
Stack

Static/global

Code section

The memory allocated for heap does not extend after the end address that is why it decrements the counter.

The memory for stack is limited and fixed and program cannot request above its allocated space.



```
#include <stdio.h>              // entire instructions will be stored in the code section
int MAX = 3;                    // Will be stored on the global block of memory
int main () {

int a=20;                       // Will be stored on stack

int *p;                         // p will be stored on stack
                                // but the content pointing to the location on heap

p=(int*)malloc(sizeof(int));    // malloc will allocate a memory on heap and
                                // *p will be holding the address of the dynamic variable

free(p);                        // deallocates the memory on heap, delete in c++


}
/*
c

malloc
calloc

free


c++

new

delete
*/
```

https://www.geeksforgeeks.org/operating-system-user-level-thread-vs-kernel-level-thread/