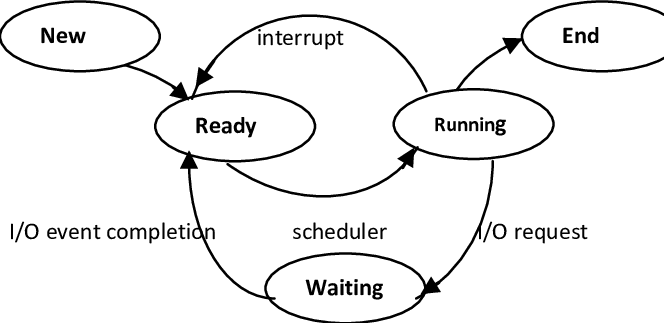
**Assignment No 3**

Q1)



Q2)

In Operating System, the fork() system call is used by a process to create another process. The process that used the fork() system call is the parent process and process consequently created is known as the child process.

Details about these are given as follows −

Process

A process is an active program i.e a program that is under execution. It is more than the program code as it includes the program counter, process stack, registers, program code etc. Compared to this, the program code is only the text section.

A process changes its state as it executes. This state partially depends on the current activity of a process. The different states that a process is in during its execution are new, ready, running, blocked, terminated.

A process control block is associated with each of the processes. It contains important information about the process it is associated with such as process state, process number, program counter, list of files and registers, CPU information, memory information etc.

Parent Process

All the processes in operating system are created when a process executes the fork() system call except the startup process. The process that used the fork() system call is the parent process. In other words, a parent process is one that creates a child process. A parent process may have multiple child processes but a child process only one parent process.

On the success of a fork() system call, the PID of the child process is returned to the parent process and 0 is returned to the child process. On the failure of a fork() system call, -1 is returned to the parent process and a child process is not created.

Child Process

A child process is a process created by a parent process in operating system using a fork() system call. A child process may also be called a subprocess or a subtask.

A child process is created as its parent process’s copy and inherits most of its attributes. If a child process has no parent process, it was created directly by the kernel.

If a child process exits or is interrupted, then a SIGCHLD signal is send to the parent process.

Q3)

A process whose parent process no more exists i.e. either finished or terminated without waiting for its child process to terminate is called an orphan process.

Q4)

A process which has finished the execution but still has entry in the process table to report to its parent process is known as a zombie process. A child process always first becomes a zombie before being removed from the process table. The parent process reads the exit status of the child process which reaps off the child process entry from the process table.

Q5)

**getpid()** returns the process ID of the current process.

Q6)

**getppid()** returns the process ID of the parent of the calling process. If the calling process was created by the fork() function and the parent process still exists at the time of the getppid function call, this function returns the process ID of the parent process. Otherwise, this function returns a value of 1 which is the process id for init process.

Q7)

**fork()** is how you create new processes in Unix. When you call fork, you're creating a copy of your own process that has its own address space. This allows multiple tasks to run independently of one another as though they each had the full memory of the machine to themselves.

Q8)

The **wait()** system call suspends execution of the current process until one of its children terminates.

The **waitpid()** system call suspends execution of the current process until a child specified by pid argument has changed state. By default, waitpid() waits only for terminated children.

Q9)

what you do is you type this command – kill -s SIGCHLD pid ​Replace the pid with the id of the parent process so that the parent process will remove all the child processes that are dead and completed.

Q10)

An unintentionally orphaned process is created when its parent process crashes or terminates.

Q11)

User space is system memory allocated to running applications.

Q12)

Kernel space is where the kernel (i.e., the core of the operating system) executes (i.e., runs) and provides its services.

Q13)

Inter process communication (IPC) is used for exchanging data between multiple threads in one or more processes or programs. ... It is a set of programming interface which allow a programmer to coordinate activities among various program processes which can run concurrently in an operating system.

Q14)

Pipes are a form of Inter-Process Communication (IPC) implemented on Unix

To create an unnamed pipe------ call the following function (system call): int pipe (int fd[2]);

Q15)

A named pipe is a named, one-way or duplex pipe for communication between the pipe server and one or more pipe clients.

Named pipes can be used to provide communication between processes on the same computer or between processes on different computers across a network.

Q16)

An unnamed pipe is only used for communication between a child and it’s parent process, while a named pipe can be used for communication between two unnamed process as well.

Q17)

The exec() family of functions replaces the current process image with a new process image.

The exec() family of functions replaces the current process image with a new process image. The functions described in this manual page are front-ends for execve(2). (See the manual page for execve(2) for further details about the replacement of the current process image.)

The initial argument for these functions is the name of a file that is to be executed.

The const char \*arg and subsequent ellipses in the execl(), execlp(), and execle() functions can be thought of as arg0, arg1, ..., argn. Together they describe a list of one or more pointers to null-terminated strings that represent the argument list available to the executed program. The first argument, by convention, should point to the filename associated with the file being executed. The list of arguments must be terminated by a NULL pointer, and, since these are variadic functions, this pointer must be cast (char \*) NULL.

The execv(), execvp(), and execvpe() functions provide an array of pointers to null-terminated strings that represent the argument list available to the new program. The first argument, by convention, should point to the filename associated with the file being executed. The array of pointers must be terminated by a NULL pointer.

Q18)

Semaphores are integer variables that are used to solve the critical section problem by using two atomic operations, wait and signal that are used for process synchronization.

**Wait**

The wait operation decrements the value of its argument S, if it is positive. If S is negative or zero, then no operation is performed.

wait(S)

{

while (S<=0);

S--;

}

Q19)

Mutex and Semaphore both provide synchronization services but they are not the same.

**Mutex**

Mutex is a mutual exclusion object that synchronizes access to a resource. It is created with a unique name at the start of a program. The Mutex is a locking mechanism that makes sure only one thread can acquire the Mutex at a time and enter the critical section. This thread only releases the Mutex when it exits the critical section.

A Mutex is different than a semaphore as it is a locking mechanism while a semaphore is a signalling mechanism. A binary semaphore can be used as a Mutex but a Mutex can never be used as a semaphore.

**Semaphore**

A semaphore is a signalling mechanism and a thread that is waiting on a semaphore can be signaled by another thread. This is different than a mutex as the mutex can be signaled only by the thread that called the wait function.

A semaphore uses two atomic operations, wait and signal for process synchronization.