Lab-2 : Processes

- In this lab we will try to learn basic system calls such as **`fork, sleep, wait and exit`**.
- A process can create a child process using the fork system call. After the fork, the memory image of the child process is a complete copy of the memory image of the parent (the child and parent memory images may diverge subsequently). The fork system call returns in both the parent and child processes, with different return values. In the parent, it returns the pid of the new child process. In the child, it returns 0. Both processes then resume execution from the instruction right after fork, and can be scheduled as independent entities by the CPU scheduler.
- A typical use case of `fork` is to create a child and run the `exec` system call in the child. The `exec` system call loads a new executable into the memory image of the process calling it, enabling the child process to do something different from what the parent is already doing. All processes (beyond the first process) are created with this fork+exec combination in Unix-like operating systems.
- A process can terminate itself using the `exit` system call. Alternately, a process can be terminated by the OS under several circumstances (e.g., when the process misbehaves). Once a process gives up the CPU for the last time after the `exit` system call, the OS can proceed to reclaim the memory of the process (note that the memory of a process cannot be reclaimed during the `exit` system call itself, while the CPU is executing on its memory).
- `sleep` This system call is used when user wants to halt the current process for sometime. It keeps the locks hold on resources till the sleep time is over and again starts the execution of the process. Here process has control throughout the execution.
- `wait` This system call is used when user wants to halt the current process and it releases all the resources hold by the process and waits for some other process to execute.

Execute the following codes for understanding it better:

1) fork system call

```
#include<stdio.h>
#include<sys/types.h>
int main(){
        fork();
        fork();
        printf("Hello World\n");
        return 0;
}
```

2) sleep system call

```
#include<stdio.h>
#include<sys/types.h>
#include<stdlib.h>
int main() {
        int pid = fork();
        if(pid == -1) {
                printf("Cannot create a process\n");
                exit(1);
        }
        else if(pid == 0) {
                sleep(2);
                printf("Child process\n");
        }
        else {
                printf("Parent Process\n");
                exit(1);
        }
        return 0;
}
```

3) wait system call

```
include<stdio.h>
#include<stdlib.h>
int i = 10;
int main(){
        int pid = fork();
        if(pid == 0){
                printf("Initial value of i is %d\n", i);
                i += 10;
                printf("value of i is %d\n", i);
                printf("Child process terminated\n");
        }
        else {
                wait(0);
                printf("value of i in parent process is %d\n", i);
        }
        return 0;
}
```

Questions

1. Write a C program to demonstrate the zombie and orphan process.
2. Write a C program in which main program accepts the integers to be sorted. Main program uses the fork system call to create a new process called a child process. Parent process sorts the integers using merge sort and waits for child process using wait system call to sort the integers using quick sort. Also demonstrate zombie and orphan states.
3. Write a C program to execute processes from bottoom up which are created using fork system call?