3.Write a shell script for bubble sort.

CODE:

```
#!/bin/bash
arr=( "$@" )
echo We will take array elements as command line arguments
printf "Array in original order : "
echo ${arr[*]}
for((i=0; i < $# ;i++))
do
for((j=0; j < $# - 1;j++))
do
      if [ ${arr[j]} -gt ${arr[$((j+1))]} ]
      then
            temp=${arr[j]}
            arr[$j]=${arr[$((j+1))]}
        arr[$((j+1))]=$temp
      fi
      done
done
printf "Array in sorted order : "
echo ${arr[*]}
```

OUTPUT:

```
linuxmint@jc69:~/Documents/OSLAB/Day5$ chmod +x 6.sh
linuxmint@jc69:~/Documents/OSLAB/Day5$ ./6.sh 9 7 2 5
We will take array elements as command line arguments
Array in original order : 9 7 2 5
Array in sorted order : 2 5 7 9
```

4. Write a program to create a child process which executes an already compiled Fibonacci series program.

CODE:

//fiboexec.c

```c
#include <stdio.h>
#include <unistd.h>
int main(){
    //Use this to compile the C program and execute it
    //char* args[] = {"sh", "-c", "gcc fibo.c -o fibo;./fibo",NULL};
    //Use this to only run an executable
    char* args[] = {"./fibo", NULL};
    execvp(args[0], args);
    return 0;
}
```

//fibo.c

```c
#include <stdio.h>
int fibo(int n){
    if (n <= 1) return n;
    else return fibo(n-1) + fibo(n-2);
}
int main(){
    int n;
    printf("Enter n for finding nth Fibonacci number(starting from 0) : ");
    scanf("%d", &n);
    printf("%dth Fibonacci number is : %d\n", n, fibo(n));
    return 0;
}
```

OUTPUT:

```
[redacted]$ gcc fiboexec.c -o fiboexec
[redacted]$ ./fiboexec
Enter n for finding nth Fibonacci number(starting from 0) : 4
4th Fibonacci number is : 3
```

5. Write a program to demonstrate a zombie process and also explain necessary conditions for the zombie process.

CODE:

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
int main()
{
    pid_t child_id = fork();
    if (child_id > 0) {
        sleep(50);
        printf("This is parent process.\n");
    }
    else if (child_id == 0) exit(EXIT_SUCCESS);
    return 0;
}
```

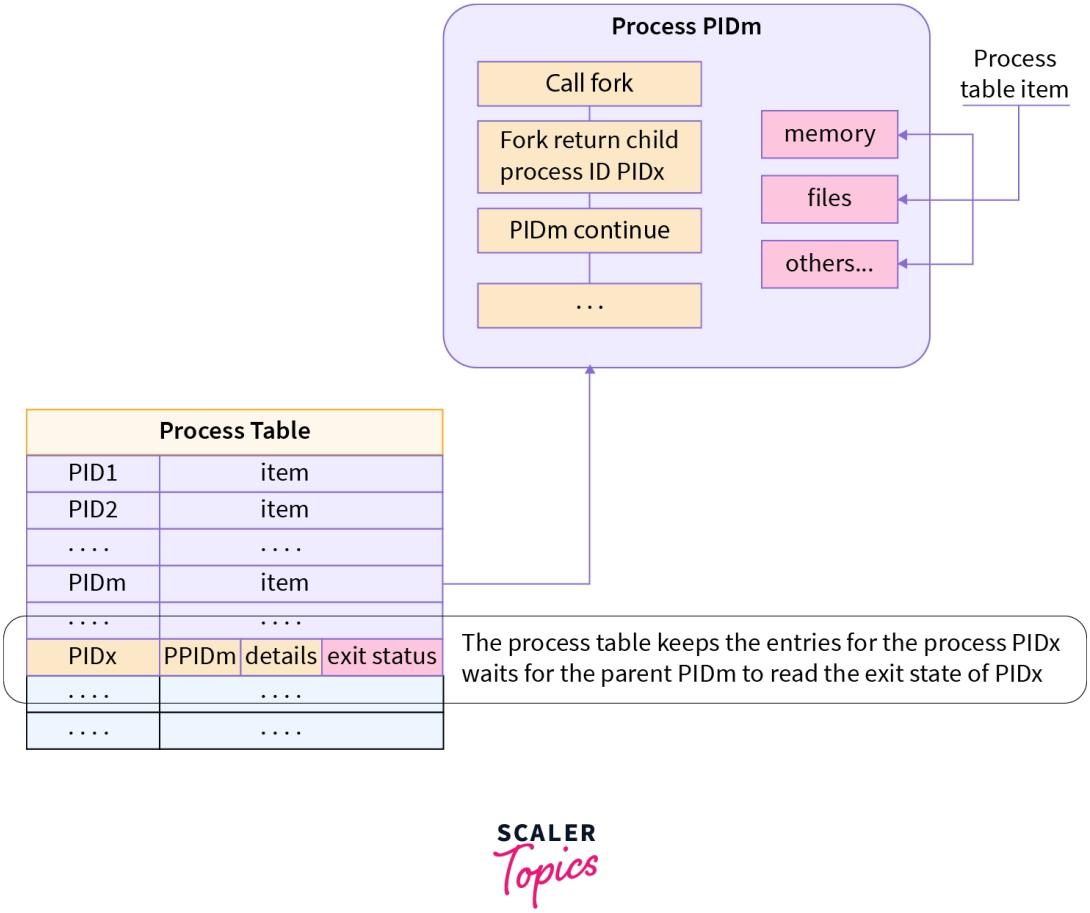
OUTPUT:

```
This is parent process.
```

What is the Zombie process?

Zombie process is also known as *"dead"* process. Ideally when a process completes its execution, its entry from the process table should be removed but this does not happen in case of a zombie process.

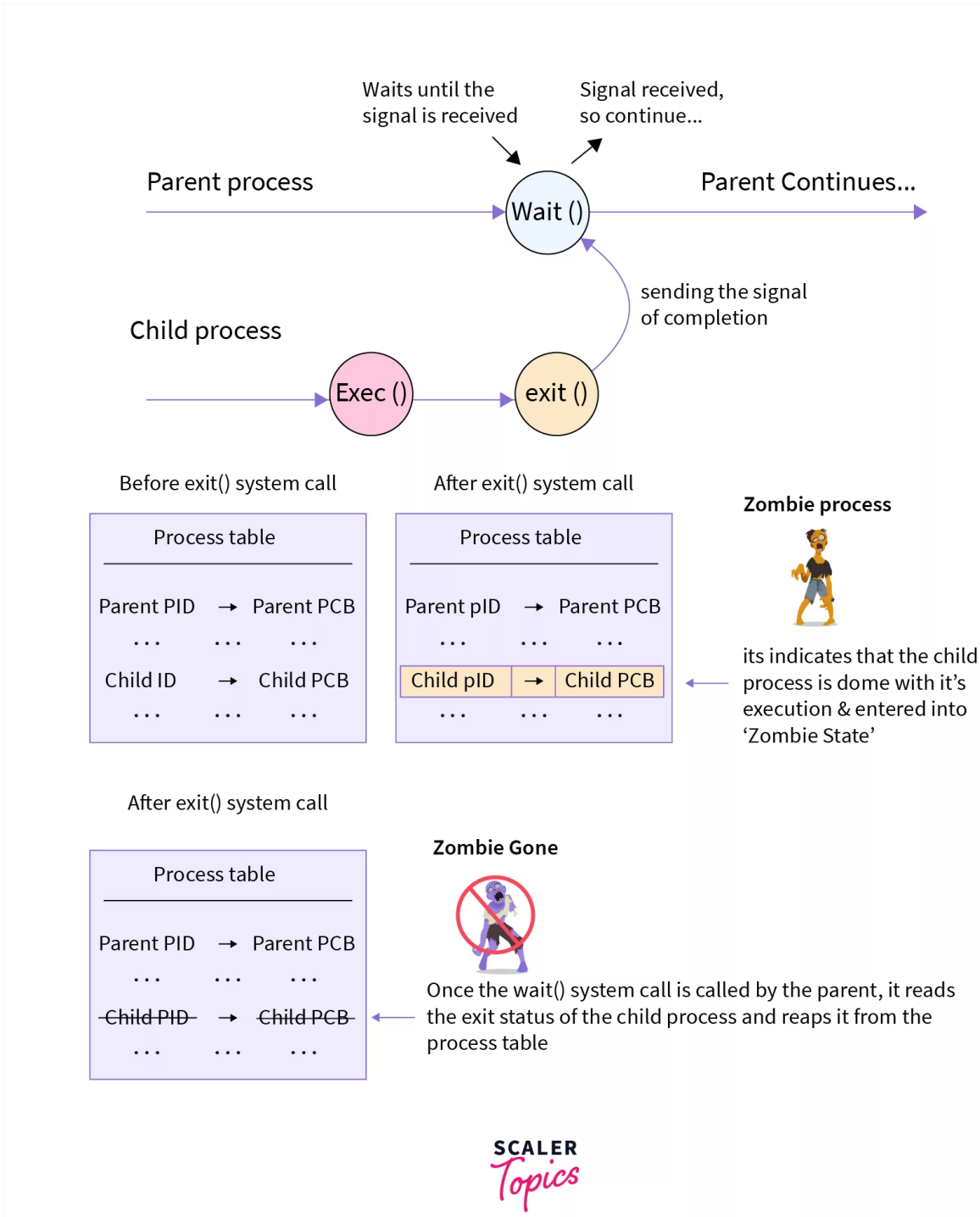
Analogy: Zombie, mythological, is a dead person revived physically. Similarly, a zombie process in os is a dead process (completed its execution) but is still revived (its entry is present in memory).



Note: Process table is a data structure in RAM to store information about a process.

What happens with the zombie processes?

- wait() system call is used for removal of zombie processes.
- wait() call ensures that the parent doesn't execute or sits idle till the child process is completed.
- When the child process completes executing ,the parent process removes entries of the child process from the process table. This is called "reaping of child".



Source : https://www.scaler.com/topics/operating-system/zombie-and-orphan-process-in-os/

7. Write a shell script to display numbers using an array.

CODE:

```
#!/bin/bash
arr=( "$@" )
echo We are creating array using command line arguments
```

```
printf "Array elements are : "
echo ${arr[@]}
```

OUTPUT:

```
linuxmint@jc69:~/Documents/OSLAB/Day5$ ./4.sh apple munni 5
We are creating array using command line arguments
Array elements are : apple munni 5
```

9. Write a shell script to generate all combinations of a 3 digit number.

CODE:

```
combine() {
    local limit=$[ 1 << $# ]
    local args=("$@")
    for ((value = 1; value < limit; value++)); do
        local parts=()
        for ((i = 0; i < $#; i++)); do
            [ $[(1<<i) & value] -ne 0 ] && parts[${#parts[@]}]="${args[i]}"
        done
        echo "${parts[@]}"
    done
}
combine $1 $2 $3
```

OUTPUT:

```
[redacted]$ ./combo.sh 1 3 2
1
3
1 3
2
1 2
3 2
1 3 2
```

10. Write a shell script to print a given number in reverse order.

```
#!/bin/bash
printf "Enter string : "
read str
revstr=`echo $str | rev`
echo "Original String : $str"
echo "Reversed String : $revstr"
```

OUTPUT:

```
linuxmint@jc69:~/Documents/OSLAB/Day5$ ./11.sh
Enter string : youtube
Original String : youtube
Reversed String : ebutuoy
```

Q.Write a shell script to generate all permutations of a 3 digit number.

CODE:

```
function swap() {
    local string=$1
    local len=${#string}
    local from=$2
    local to=$3
    local i=0
    local s=""
    while [[ $i -lt $len ]]; do
        if [[ $i -eq $from ]]; then
            s=$s${string:$to:1}
        elif [[ $i -eq $to ]]; then
            s=$s${string:$from:1}
        else
            s=$s${string:$i:1}
        fi
        i=$(($i+1))
    done
    echo $s
}

function perm() {
    local string=$1
    local len=${#string}
    local idx=$2
    if [[ $idx -ge $len ]]; then
        echo $string
    else
        local i=$idx
        while [[ $i -lt $len ]]; do
            perm $(swap $string $i $idx) $((idx+1))
            i=$((i+1))
        done
    fi
}

perm $1
```