Process control system calls: The demonstration of fork, execve andwait system calls along with zombie and orphan states

a. Implement the 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 state.

==========

```
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
void quicksort(int [10],int,int);
void main()
{
int pid, ppid, a[10], size, i;
printf("\nPARENT: My process-id is %d.", getpid());
printf("\n\nEnter the number of elements in the array: ");
scanf("%d", &size);
printf("Enter %d elements: ", size);
   for(i=0;i<size;i++)
     scanf("%d",&a[i]);
printf("\nForking now!\n");
pid = fork(); //Here, the child process is created and both child & parent run
//simultaneously. fork() returns 0 to child
process and pid of  //child to parent process.
So, if int pid here is 0, it means we  //are in
child process and if any +ve no., then in parent process
//where the +ve no. is the child's pid.
if(pid==0)
{
printf("\nCHILD: My process-id is %d.", getpid());
printf("\nCHILD: My parent's process-id is %d.", getppid());
quicksort(a, 0, size-1);
printf("\nSorted elements: ");
  for(i=0;i<size;i++)
     printf(" %d",a[i]);printf("\nCHILD: I am dying now");
printf("\n-----------------------------------------------------");
}
else if (pid>0)
{
wait(NULL);
printf("\nPARENT: I am back. ");
printf("\nPARENT: My process-id is %d.", getpid());
printf("\nPARENT: My child's process-id is %d.", pid);
quicksort(a, 0, size-1);
printf("\nSorted elements: ");
  for(i=0;i<size;i++)
     printf(" %d",a[i]);
printf("\nPARENT: I am dying now.");
printf("\n-----------------------------------------------------");

}
}
void quicksort(int a[10],int first,int last)
{
    int pivot,j,temp,i;
    if(first<last)
{
pivot=first;
```

```
        i=first;
        j=last;
        while(i<j){            while(a[i]<=a[pivot]&&i<last)
                i++;
            while(a[j]>a[pivot])
                j--;
            if(i<j){
temp=a[i];
                a[i]=a[j];
                a[j]=temp;
             }
         }
         temp=a[pivot];
         a[pivot]=a[j];
         a[j]=temp;
         quicksort(a,first,j-1);
         quicksort(a,j+1,last);
    }
}
```

Output:

```
pict@ubuntu:~/3807/Final$ gcc a_forkandsort.c
pict@ubuntu:~/3807/Final$ ./a.out
PARENT: My process-id is 2932.
Enter the number of elements in the array: 4
Enter 4 elements:
56
12
388
92
Forking now!
CHILD: My process-id is 2934.
CHILD: My parent's process-id is 2932.
Sorted elements:  12 56 92 388
CHILD: I am dying now
-----------------------------------------------------
PARENT: I am back.
PARENT: My process-id is 2932.
PARENT: My child's process-id is 2934.
Sorted elements:  12 56 92 388
PARENT: I am dying now.
-----------------------------------------------------
```

Orphan Process:

```
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
void quicksort(int [10],int,int);
void main()
{
int pid, ppid, a[10], size, i;
printf("\nPARENT: My process-id is %d.", getpid());
printf("\n\nEnter the number of elements in the array: ");
scanf("%d", &size);
printf("Enter %d elements: ", size);
   for(i=0;i<size;i++)
     scanf("%d",&a[i]);
printf("\nForking now!\n");
```

```
pid = fork();

//Here, the child process is created and both child & parent run
//simultaneously. fork() returns 0 to child  process and pid of parent process
to child
So, if int pid here is 0, it means we are in  child process and if any +ve no.,
then in parent process where the +ve no. is the child's pid.


//Child process

if(pid==0)
{
printf("\nCHILD: My process-id is %d.", getpid());
printf("\nCHILD: My parent's process-id is %d.", getppid());
quicksort(a, 0, size-1);
printf("\nSorted elements: ");
  for(i=0;i<size;i++)
     printf(" %d",a[i]);printf("\nCHILD: I am dying now");
printf("\n-----------------------------------------------------");
// ORPHAN
printf("\nCHILD: I am sleeping now.");
sleep(10);
printf("\n-----------------------------------------------------\n\n");
}

//Parent process

else if (pid>0)
{
system("wait");
printf("\nPARENT: I am back. ");
printf("\nPARENT: My process-id is %d.", getpid());
printf("\nPARENT: My child's process-id is %d.", pid);
quicksort(a, 0, size-1);
printf("\nSorted elements: ");
  for(i=0;i<size;i++)
     printf(" %d",a[i]);
printf("\nPARENT: I am dying now.");
printf("\n-----------------------------------------------------");
// ORPHAN
exit(0);
}
}
void quicksort(int a[10],int first,int last)
{
    int pivot,j,temp,i;
    if(first<last)
{
pivot=first;
        i=first;
        j=last;
        while(i<j){            while(a[i]<=a[pivot]&&i<last)
                i++;
            while(a[j]>a[pivot])
                j--;
            if(i<j){
temp=a[i];
                a[i]=a[j];
                a[j]=temp;
             }
         }
         temp=a[pivot];
         a[pivot]=a[j];
```

```
        a[j]=temp;
        quicksort(a,first,j-1);
        quicksort(a,j+1,last);
    }
}
```

Zombie:

```
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
void quicksort(int [10],int,int);
void main()
{
int pid, ppid, a[10], size, i;
printf("\nPARENT: My process-id is %d.", getpid());
printf("\n\nEnter the number of elements in the array: ");
scanf("%d", &size);
printf("Enter %d elements: ", size);
   for(i=0;i<size;i++)
     scanf("%d",&a[i]);
printf("\nForking now!\n");
pid = fork();
```

//Here, the child process is created and both child & parent run
//simultaneously. fork() returns 0 to child process and pid of parent process to child
So, if int pid here is 0, it means we are in child process and if any +ve no., then in parent process where the +ve no. is the child's pid.

//Child Process

```
if(pid==0)
{
printf("\nCHILD: My process-id is %d.", getpid());
printf("\nCHILD: My parent's process-id is %d.", getppid());
quicksort(a, 0, size-1);
printf("\nSorted elements: ");
  for(i=0;i<size;i++)
     printf(" %d",a[i]);printf("\nCHILD: I am dying now");
printf("\n-----------------------------------------------------");
```

// ZOMBIE
printf("\nCHILD: I am dying now");
printf("\n---\n\n");
exit(0);
}

//Parent process

```
else if (pid>0)
{

printf("\nPARENT: I am back. ");
printf("\nPARENT: My process-id is %d.", getpid());
printf("\nPARENT: My child's process-id is %d.", pid);
```

```
quicksort(a, 0, size-1);
printf("\nSorted elements: ");
  for(i=0;i<size;i++)
     printf(" %d",a[i]);
printf("\nPARENT: I am dying now.");
printf("\n-----------------------------------------------------------");

// ZOMBIE
printf("\nPARENT: I am sleeping now.");
sleep(10);
printf("\n-----------------------------------------------------------\n");

}
}


void quicksort(int a[10],int first,int last)
{
    int pivot,j,temp,i;
    if(first<last)
{
pivot=first;
        i=first;
        j=last;
        while(i<j){            while(a[i]<=a[pivot]&&i<last)
                i++;
            while(a[j]>a[pivot])
                j--;
            if(i<j){
temp=a[i];
                a[i]=a[j];
                a[j]=temp;
             }
         }
         temp=a[pivot];
         a[pivot]=a[j];
         a[j]=temp;
         quicksort(a,first,j-1);
         quicksort(a,j+1,last);
    }
}
```