Assignment 2B

AIM:
Implement the C program in which main program accepts an integer array. Main program uses the FORK system call to create a new process called a child process. Parent process sorts an integer array and passes the sorted array to child process through the command line arguments of EXECVE system call. The child process uses EXECVE system call to load new program which display array in reverse order.

Theory:

New program execution within the existing process (The exec Function) The fork function creates a copy of the calling process, but many applications require the child process to execute code that is different from that of the parent. The exec family of functions provides a facility for overlaying the process image of the calling process with a new image. The traditional way to use the fork–exec combination is for the child to execute (with an exec function) the new program while the parent continues to execute the original code.
exec() system call:
The exec() system call is used after a fork() system call by one of the two processes to replace the memory space with a new program. The exec() system call loads a binary file into memory (destroying image of the program containing the exec() system call) and go their separate ways. Within the exec family there are functions that vary slightly in their capabilities. exec family: 1. execl() and execlp(): execl(): It permits us to pass a list of command line arguments to the program to be executed. The list of arguments is terminated by NULL. e.g. execl("/bin/ls", "ls", "-l", NULL); execlp(): It does same job except that it will use environment variable PATH to determine which executable to process. Thus a fully qualified path name would not have to be used. The function execlp() can also take the fully qualified name as it also resolves explicitly. e.g. execlp("ls", "ls", "-l", NULL)

execve( ):
int execve(const char *filename, char *const argv[ ], char *const envp[ ]); It executes the program pointed to by filename. filename must be either a binary executable, or a script starting with a line of the form: argv is an array of argument strings passed to the new program. By convention, the first of these strings should contain the filename associated with the file being executed. envp is an array of strings, conventionally of the form key=value, which are

Assignment 2B

passed as environment to the new program. Both argv and envp must be terminated by a NULL pointer. The argument vector and environment can be accessed by the called program's main function, when it is defined as: int main(int argc, char *argv[ ] , char *envp[ ])] execve() does not return on success, and the text, data, bss, and stack of the calling process are overwritten by that of the program loaded. All exec functions return –1 if unsuccessful. In case of success these functions never return to the calling function

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

void bubbleSort(int arr[], int n) {
    int temp, i, j;
    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

int main() {
    int n, i;
    printf("Enter the number of integers you want to sort: ");
```

Assignment 2B

```
    scanf("%d", &n);
    int arr[n];

    printf("Enter %d integers:\n", n);
    for (i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    // Forking a child process
    pid_t pid = fork();

    if (pid < 0) {
        printf("Fork failed.\n");
        exit(1);
    } else if (pid == 0) { // Child process
        printf("\nArray in Sorted Order\n");
        bubbleSort(arr, n);
        for (i = 0; i < n; i++) {
            printf("%d ", arr[i]);
        }
        printf("\nChild process\n");
        printf("Child Process id %d \n", getpid());
        printf("Parent Process id %d \n", getppid());

        // Converting integer array to string array for execve
        char arr_str[n][10];
        char* args[n + 2];
        args[0] = "./assignchild"; // Executable name for child process
        for (i = 0; i < n; i++) {
            sprintf(arr_str[i], "%d", arr[i]);
            args[i + 1] = arr_str[i];
        }
        args[n + 1] = NULL; // Mark the end of arguments
        execve(args[0], args, NULL); // Execute the child process
    } else { // Parent process
        printf("\nParent process\n");
        printf("Process id %d \n", getpid());
    }
    return 0;
}

- ----------------------------------------------------------------------------------------
#include<stdio.h>
#include<unistd.h>
```

Assignment 2B

```
int main(int argc , char *argv[]){
        printf("In child file \n");
        printf("Array in reverse order\n");
        for (int i = argc- 1 ; i >=0; i--){
      printf("%s ", argv[i]);
   }
        return 0;
}
```

- ----------

Output

```
pict@pict:~$ cd Documents
pict@pict:~/Documents$ cd trial
pict@pict:~/Documents/trial$ gcc assignchild.c -o assignchild
pict@pict:~/Documents/trial$ gcc assign2b.c -o assign
pict@pict:~/Documents/trial$ ./assign
Enter the number of integers you want to sort: 5
Enter 5 integers:
7 2 8 3 4

Parent process
Process id 6129
Array in Sorted Order

Child process
2 3 4 7 8 Child Process id 6130
Parent Process id 6129
pict@pict:~/Documents/trial$ In child file
Array in reverse order
Current Process id 6130
8 7 4 3 2 ./assignchild
```