

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/lis", "lis", "-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("lis", "lis", "-l", NULL);

2. execv() and execvp():

execv(): It does same job as execl() except that command line arguments can be passed to it in the form of an array of pointers to string.

e.g.

Operating Systems Laboratory

```
char *argv[] = ("ls", "-l", NULL);
```

```
execv("/bin/ls", argv);
```

execvp(): It does same job expect 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.

e.g.

```
execvp("ls", argv);
```

3. `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 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.

CODE:-

//Main Process

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
```

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

```
int main() {
    int n;
    printf("Enter the size of array: ");
    scanf("%d",&n);
    int arr[n];
    printf("Enter the elements of the array: ");
    for(int i=0;i<n;i++){
        scanf("%d",&arr[i]);
    }
}
```

```
pid_t pid = fork();
```

```
if (pid < 0) {
    fprintf(stderr, "Fork failed\n");
    return 1;
} else if (pid == 0) {
```

```

// Child process
printf("Main process start: (pid) %d\n",getpid());
sortArray(arr, n);
printf("Sorted array: ");
for (int i = 0; i < n; i++) {
    printf("%d ", arr[i]);
}
printf("\n");

    char *args[n + 2];
args[0] = "./cp";
for (int i = 0; i < n; i++) {
    char buffer[10];
    snprintf(buffer, sizeof(buffer), "%d", arr[i]); // converts int to string format and
pass them into the buffer
    args[i + 1] = strdup(buffer); //strdup:- (string duplicate) create a new copy of the
string
}
args[n + 1] = NULL; // indicate the end of the list of command-line arguments.

execve(args[0], args, NULL);
perror("execve");
printf("Main process end: (pid) %d\n",getpid());
return 1;

} else {
    // Parent process
    wait(NULL);
    printf("Parent process finished.\n");

}


return 0;
}

```

```
//CHILD PROCESS
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int main(int argc, char *argv[]) {
    pid_t pid = fork();
    printf("Child process start: (pid) %d\n",getpid());
    printf("Array in reverse order: ");
    for (int i = argc - 1; i > 0; i--) {
        printf("%s ", argv[i]);
    }
    printf("\n");
    printf("Child process start: (pid) %d\n",getpid());
    return 0;
}
```

OUTPUT:-



```
sahil@sahil-Lenovo-IdeaPad-S145-15IWL: ~/Desktop/OS_PRACTICALS
sahil@sahil-Lenovo-IdeaPad-S145-15IWL:~/Desktop/OS_PRACTICALS$ gcc childProcess.c -o cp
sahil@sahil-Lenovo-IdeaPad-S145-15IWL:~/Desktop/OS_PRACTICALS$ gcc mainProcess.c -o mp
sahil@sahil-Lenovo-IdeaPad-S145-15IWL:~/Desktop/OS_PRACTICALS$ ./mp
Enter the size of array: 5
Enter the elements of the array: 1 5 4 6 2
Main process start: (pid) 12989
Sorted array: 1 2 4 5 6
Child process start: (pid) 12989
Array in reverse order: 6 5 4 2 1
Child process start: (pid) 12989
Child process start: (pid) 12990
Array in reverse order: 6 5 4 2 1
Child process start: (pid) 12990
Parent process finished.
sahil@sahil-Lenovo-IdeaPad-S145-15IWL:~/Desktop/OS_PRACTICALS$ █
```

