

# Assignment-4

## Fork assignment-3

Subash Mylraj  
(CED18I051)

16 September 2020

**Question 1: Test drive a C program that creates Orphan and Zombie Processes.**

### Code: Zombie

---

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

int main (){

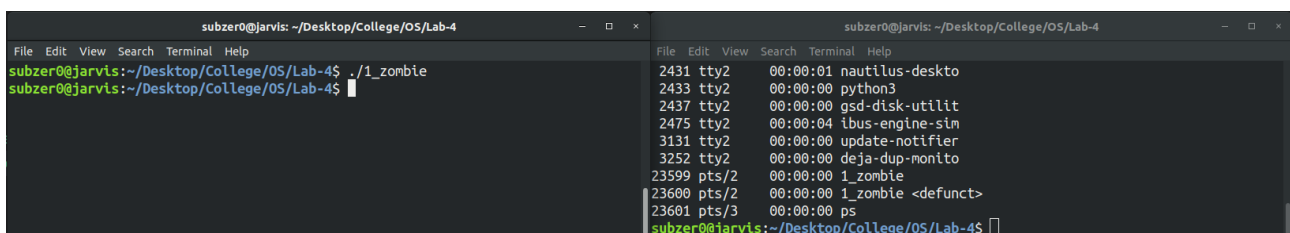
    pid_t pid = fork();

    if(pid == 0){
        exit(0);
    }
    else{
        sleep(10);
    }

    return 0;
}
```

---

### Output:



```
subzer0@jarvis: ~/Desktop/College/OS/Lab-4
File Edit View Search Terminal Help
subzer0@jarvis:~/Desktop/College/OS/Lab-4$ ./1_zombie
subzer0@jarvis:~/Desktop/College/OS/Lab-4$

subzer0@jarvis: ~/Desktop/College/OS/Lab-4
File Edit View Search Terminal Help
2431 tty2 00:00:01 nautilus-deskto
2433 tty2 00:00:00 python3
2437 tty2 00:00:00 gsd-disk-utilit
2475 tty2 00:00:04 ibus-engine-sim
3131 tty2 00:00:00 update-notifier
3252 tty2 00:00:00 deja-dup-monito
23599 pts/2 00:00:00 1_zombie
23600 pts/2 00:00:00 1_zombie <defunct>
23601 pts/3 00:00:00 ps
subzer0@jarvis:~/Desktop/College/OS/Lab-4$
```

### Code: Orphan

---

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

int main (){
```

```

pid_t pid = fork();

if(pid == 0){
    sleep(10);
}
else{
    exit(0);
}

return 0;
}

```

## Output:

```

subzer0@jarvis: ~/Desktop/College/OS/Lab-4
subzer0@jarvis:~/Desktop/College/OS/Lab-4$ ./1_orphan
subzer0@jarvis:~/Desktop/College/OS/Lab-4$

subzer0@jarvis:~/Desktop/College/OS/Lab-4$ ps
2406 tty2      00:00:00 gsd-printer
2431 tty2      00:00:01 nautilus-deskto
2433 tty2      00:00:00 python3
2437 tty2      00:00:00 gsd-disk-utilit
2475 tty2      00:00:04 ibus-engine-sim
3131 tty2      00:00:00 update-notifier
3252 tty2      00:00:00 deja-dup-monito
23776 pts/2    00:00:00 1_orphan
23777 pts/3    00:00:00 ps
subzer0@jarvis:~/Desktop/College/OS/Lab-4$

```

**Question 2:** Develop a multiprocessing version of Merge or Quick Sort. Extra credits would be given for those who implement both in a multiprocessing fashion [ increased no of processes to enhance the effect of parallelization]

## Code:

```

#include<stdio.h>
#include<sys/shm.h>
#include<unistd.h>
#include<stdlib.h>
#include<sys/wait.h>

void merge_sort(int arr[], int s, int e);
void merge(int arr[], int s, int mid, int e);

int main (){

    key_t key = ftok("merge_sort", 8);
    int shmid = shmget(key, sizeof(int)*20, 0666|IPC_CREAT);
    int *arr = shmat(shmid, 0, 0);

    int array[] = {3, 1, 5, 2, 3, 6, 8, 5, 9, 0}, n = sizeof(array)/sizeof(int);

    printf("Input Array: ");
    for(int i=0; i<n; i++){
        printf("%d ", array[i]);
        arr[i] = array[i];
    }
    printf("\n");

    merge_sort(arr, 0, n-1);

    printf("Output Array: ");

```

```

    for(int i=0; i<n; i++){
        printf("%d ", arr[i]);
    }
    printf("\n");

    shmdt(arr);
    shmctl(shmid, IPC_RMID, NULL);

    return 0;
}

void merge(int arr[], int s, int mid, int e){

    int temp[e-s+1];
    int i = 0;
    int l = s;
    int r = mid+1;
    while( l<=mid && r<=e ){

        if(arr[l] < arr[r]){
            temp[i] = arr[l];
            l++; i++;
        }
        else if(arr[l] > arr[r]){
            temp[i] = arr[r];
            r++; i++;
        }
        else if(arr[l] == arr[r]){
            temp[i] = arr[r];
            i++; r++;
            temp[i] = arr[l];
            i++; l++;
        }
    }

    while(l <= mid){
        temp[i] = arr[l];
        i++; l++;
    }

    while(r <= e){
        temp[i] = arr[r];
        i++; r++;
    }

    for(int j=0; j<i; j++){
        arr[s+j] = temp[j];
    }
}

void merge_sort(int arr[], int s, int e){

    if(s < e){

        int mid = (s + e)/2;

        pid_t right_pid, left_pid;
        left_pid = fork();

        if(left_pid == 0){
            merge_sort(arr, s, mid);
            exit(0);
        }
        else{
            right_pid = fork();
            if(right_pid == 0){

```

```

        merge_sort(arr, mid+1, e);
        exit(0);
    }
}

int status;

waitpid(left_pid, &status, 0);
waitpid(right_pid, &status, 0);

merge(arr, s, mid, e);
}
}

```

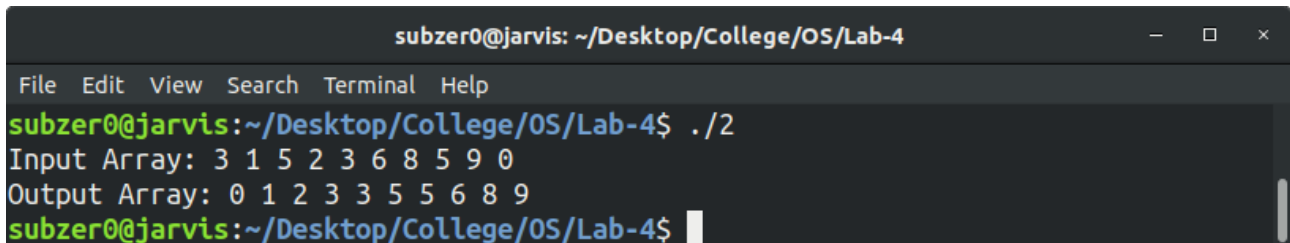
---

## Explanation:

The above code is a multiprocessing implementation of merge sort. Basically, every division is done by a new process. Every merge is done by a process when its subproblems are completed. Hence, the placement of `waitpid()` can be justified.

Shared memory was used to make sure that the changes done to the array are available across all the processes.

## Output:



```

subzer0@jarvis: ~/Desktop/College/OS/Lab-4
File Edit View Search Terminal Help
subzer0@jarvis:~/Desktop/College/OS/Lab-4$ ./2
Input Array: 3 1 5 2 3 6 8 5 9 0
Output Array: 0 1 2 3 3 5 5 6 8 9
subzer0@jarvis:~/Desktop/College/OS/Lab-4$

```

**Question 3: Develop a C program to count the maximum number of processes that can be created using fork call.**

## Code:

---

```

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

int main (){

    int count = 0;

    while(1){
        pid_t pid = fork();
        if(pid == 0){
            sleep(10);
            exit(0);
        }
        count += 1;
        if(pid < 0){
            printf("Could not create more processes\n");
            break;
        }
    }
}

```

```

    }
}

printf("Total number of processes created: %d\n", count);
wait(NULL);

return 0;
}

```

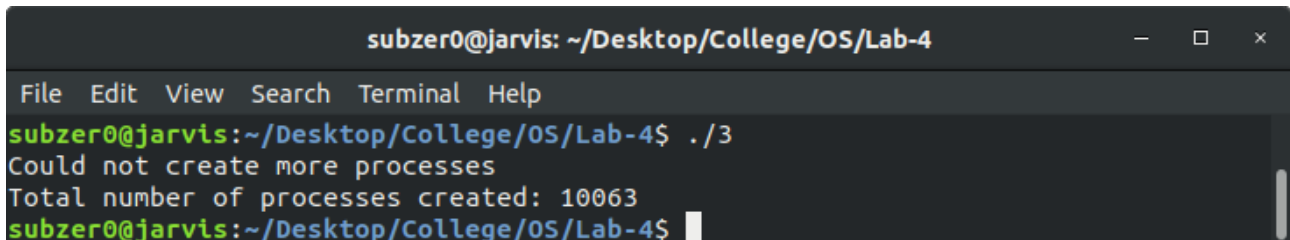
---

## Explanation:

This code creates child processes in an infinite loop till it reaches a point when it cannot create anymore processes. This limit can be assumed as the cap on the number of processes that can be created.

To avoid the issue of child processes calling fork repeatedly, the child processes are made to sleep for a time of 10 seconds. So essentially, this program finds the total number of processes that can be created in about 10 seconds (its a bit more than 10 seconds as there is time taken for context switching and other statements). Increasing the sleep time for each process, produced similar results.

## Output:



```

subzer0@jarvis: ~/Desktop/College/OS/Lab-4
File Edit View Search Terminal Help
subzer0@jarvis:~/Desktop/College/OS/Lab-4$ ./3
Could not create more processes
Total number of processes created: 10063
subzer0@jarvis:~/Desktop/College/OS/Lab-4$

```

**Question 4:** Develop your own command shell [say mark it with @] that accepts user commands (System or User Binaries), executes the commands and returns the prompt for further user interaction. Also extend this to support a history feature (if the user types !6 at the command prompt; it shud display the most recent execute 6 commands). You may provide validation features such as !10 when there are only 9 files to display the entire history contents and other validations required for the history feature;

## Code:

---

```

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

struct list{
    int count;
    struct node *head;
};

```

```

struct node{
    char *command;
    struct node *next;
};

struct list history;

void insert (char *);
void show (int);
void exec_cmd(char *[]);

int main (){

    char hostname[HOST_NAME_MAX], username[LOGIN_NAME_MAX], work_dir[PATH_MAX];
    gethostname(hostname, 20);
    getlogin_r(username, 20);

    history.count = 0;
    history.head = NULL;

    printf("\033[1;31m-----Welcome to the new terminal-----\033[0m\n");
    char *argv[100];

    while(1){

        getcwd(work_dir, sizeof(work_dir));
        printf("\033[1;36m%s@\033[1;35m%s:\033[01;33m%s\033[1;31m@ \033[0m", username, hostname,
            work_dir);
        char *argv[100];
        exec_cmd(argv);

    }
    return 0;
}

void exec_cmd(char *argv[100]){

    char *user_input = (char *)malloc(100);
    scanf(" %[\n]", user_input);

    char *cmd = (char *)malloc(100);
    strcpy(cmd, user_input);
    insert(cmd);

    argv[0] = strtok(user_input, " ");
    int i=0;

    while (argv[i] != NULL) {
        i++;
        argv[i] = strtok(NULL, " ");
    }

    if(strcmp(argv[0], "exit")==0){
        exit(0);
    }
    else if(argv[0][0] == '!'){
        if(argv[1] != NULL){
            show(atoi(argv[1]));
        }
        else{
            show(10);
        }
    }
    else if(strcmp(argv[0], "cd")==0){
        chdir(argv[1]);
    }
}

```

```

else{
    pid_t pid = vfork();
    if(pid == 0){
        execvp(argv[0], argv);
    }
    else{
        wait(NULL);
    }
}
}

void insert (char *cmd){

    struct node* temp = (struct node*)malloc(sizeof(struct node));
    temp->command = cmd;

    if(history.head == NULL){
        temp->next = NULL;
        history.head = temp;
        history.count = 1;
        return;
    }

    temp->next = history.head;
    history.head = temp;

    if(history.count == 10){
        struct node* temp = history.head, temp1;

        for(int i=0; i<9; i++){
            temp = temp->next;
        }
        temp->next = NULL;
    }
    else{
        history.count += 1;
    }
}

void show (int n){

    struct node* temp = history.head;
    int counter = 0;
    while(temp != NULL && counter < n){
        printf("%s\n", temp->command);
        temp = temp->next;
        counter += 1;
    }
}

```

---

## Explanation:

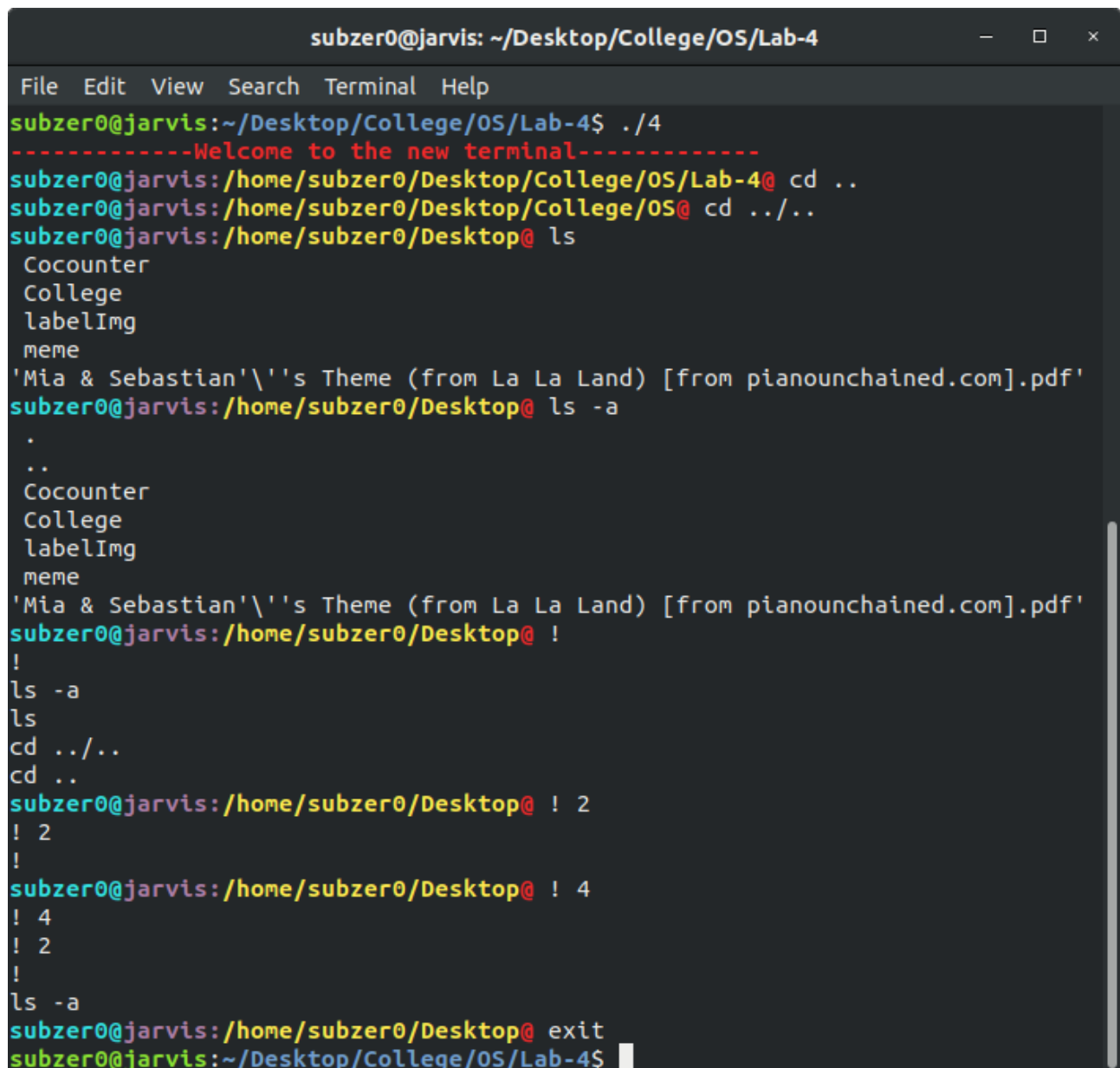
This code creates a new terminal interface. Any command that is located in the `/bin/` directory will work. Sadly piping logics do not work on this terminal. The command `exit` can be used to exit the terminal. Though it is to be noted that, this command is not called through `exec` statements.

History of the last 10 commands can be displayed by typing `!`. Further, typing `!5` shows the last 5 commands used. Typing a value more than 10 will only print the last 10 commands used.

Colours were added using `\033` in `printf` statements.

Directories can be changed by using the `cd` command. This command is not called through `exec` calls rather by calling the `chdir` functions. The prompt displays the current username and hostname using `gethostname()` and `getlogin_r()` functions. The current working directory information is obtained by using the `getcwd()` function.

## Output:



```
subzer0@jarvis: ~/Desktop/College/OS/Lab-4
File Edit View Search Terminal Help
subzer0@jarvis:~/Desktop/College/OS/Lab-4$ ./4
-----Welcome to the new terminal-----
subzer0@jarvis:/home/subzer0/Desktop/College/OS/Lab-4@ cd ..
subzer0@jarvis:/home/subzer0/Desktop/College/OS@ cd ../../
subzer0@jarvis:/home/subzer0/Desktop@ ls
Cocounter
College
labelImg
meme
'Mia & Sebastian'\''s Theme (from La La Land) [from pianounchained.com].pdf'
subzer0@jarvis:/home/subzer0/Desktop@ ls -a
.
..
Cocounter
College
labelImg
meme
'Mia & Sebastian'\''s Theme (from La La Land) [from pianounchained.com].pdf'
subzer0@jarvis:/home/subzer0/Desktop@ !
!
ls -a
ls
cd ../../
cd ..
subzer0@jarvis:/home/subzer0/Desktop@ ! 2
! 2
!
subzer0@jarvis:/home/subzer0/Desktop@ ! 4
! 4
! 2
!
ls -a
subzer0@jarvis:/home/subzer0/Desktop@ exit
subzer0@jarvis:~/Desktop/College/OS/Lab-4$
```



## Question 5: Develop a multiprocessing version of Histogram generator to count the occurrence of various characters in a given text.

Code:

---

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

int main (){

    struct histo{
        char chars[95];
        int len;
    };

    char *text = "hi there! how are you?";

    char cur_char = text[0];
    int lens = 1, count = 1;

    key_t key = ftok("shmfile", 65);
    int shmid = shmget(key,sizeof(struct histo),0666|IPC_CREAT);
    struct histo *hist = shmat(shmid, 0, 0);

    printf("String: %s\n", text);
    printf("-----Histogram-----\n");

    hist->len = 1;
    hist->chars[hist->len-1] = cur_char;

    for(int i=1; i<strlen(text); i++){

        if(text[i] == cur_char){
            count += 1;
        }
        else{
            // check if character exists in the list.
            // if not, then add to it and fork
            // else continue
            // printf("%c: %c\n", cur_char, text[i]);
            int flag = 0;
            for(int j=0; j<hist->len; j++){
                if(hist->chars[j] == text[i]){
                    flag = 1;
                    break;
                }
            }
            if(flag == 0){
                hist->chars[hist->len] = text[i];
                hist->len += 1;
                pid_t pid = fork();
                if(pid == 0){
                    count = 1;
                    cur_char = text[i];
                }
                else{
                    continue;
                }
            }
        }
    }
}
```

```

    }
    else{
        continue;
    }
}
}

printf("%c: %d\n", cur_char, count);
shmdt(hist);
shmctl(shmid,IPC_RMID,NULL);

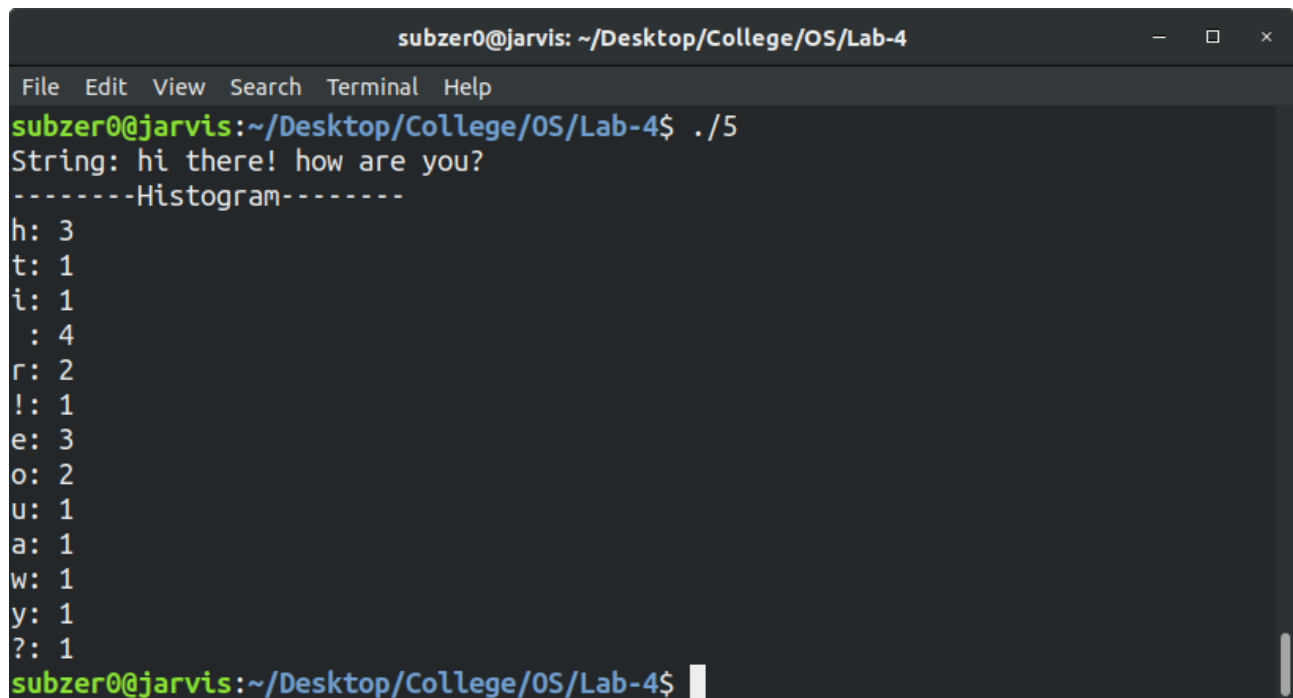
return 0;
}

```

## Explanation:

This code generates a histogram of the letters used in a given string. This code uses a multiprocessing approach. For every unique letter, a new process is created to count its occurrence. Once each process traverses the string, each of them prints the character they were responsible for and their respective count. Shared memory was used to maintain a visit array (basically keeps track of all the unique characters) to make sure multiple processes don't count the same character.

## Output:



```

subzer0@jarvis: ~/Desktop/College/OS/Lab-4
File Edit View Search Terminal Help
subzer0@jarvis:~/Desktop/College/OS/Lab-4$ ./5
String: hi there! how are you?
-----Histogram-----
h: 3
t: 1
i: 1
 : 4
r: 2
!: 1
e: 3
o: 2
u: 1
a: 1
w: 1
y: 1
?: 1
subzer0@jarvis:~/Desktop/College/OS/Lab-4$

```

**Question 6:** Develop a multiprocessing version of matrix multiplication. Say for a result 3\*3 matrix the most efficient form of parallelization can be 9 processes, each of which computes the net resultant value of a row (matrix1) multiplied by column (matrix2). For programmers convenience you can start with 4 processes, but as I said each result value can be computed parallel independent of the other processes in execution.

**Code:**

---

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

int main (){

    int A[] [3] = {
        {3, 1, 5},
        {2, 7, 9},
        {4, 1, 2}
    };

    int B[] [2] = {
        {1, 1},
        {3, 1},
        {0, 1}
    };

    int const m=3, n=3, p=3, q=2; // n = p;

    int prod[m][q], sum=0;

    for(int c=0; c<m; c++){
        for(int d=0; d<q; d++){
            for(int k=0; k<p; k++){
                pid_t pid = vfork();
                if(pid == 0){
                    sum = sum + A[c][k] * B[k][d];
                    exit(0);
                }
            }
            prod[c][d] = sum;
            sum = 0;
        }
    }

    for(int i=0; i<m; i++){
        for(int j=0; j<q; j++){
            printf("%d ", prod[i][j]);
        }
        printf("\n");
    }

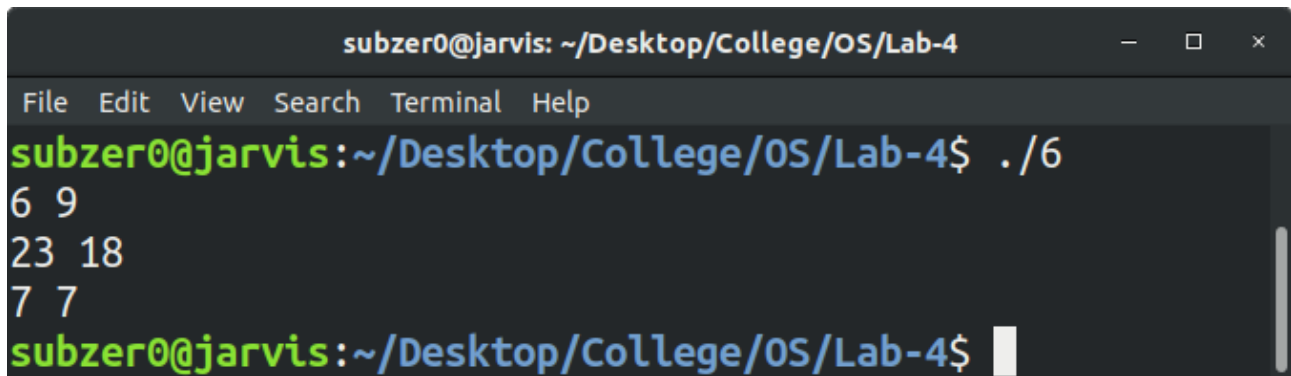
    return 0;
}
```

---

**Explanation:**

Appropriately placed fork calls with shared memory in the usual matrix multiplication algorithm ensures that a multiprocessing version of matrix multiplication can be achieved.

## Output:

A terminal window titled 'subzer0@jarvis: ~/Desktop/College/OS/Lab-4' with a menu bar (File, Edit, View, Search, Terminal, Help). The prompt is 'subzer0@jarvis:~/Desktop/College/OS/Lab-4\$' and the command './6' has been executed. The output consists of three lines of numbers: '6 9', '23 18', and '7 7'. The prompt is now 'subzer0@jarvis:~/Desktop/College/OS/Lab-4\$' with a cursor.

```
subzer0@jarvis: ~/Desktop/College/OS/Lab-4
File Edit View Search Terminal Help
subzer0@jarvis:~/Desktop/College/OS/Lab-4$ ./6
6 9
23 18
7 7
subzer0@jarvis:~/Desktop/College/OS/Lab-4$
```