OPERATING SYSTEMS - COM301P

LAB ASSIGNMENT - 5

PIPES

VIKNESH RAJARAMON

COE18B060

1) Parent sets up a string which is read by child, reversed there and read back the parent

LOGIC:

- We create two pipes - **fd_parent** and **fd_child**.
- **fd_parent** allows the **Parent process to write** and the **Child process to read**.
- **fd_child** allows the **Child process to write** and the **Parent process to read**.
- In the **parent process**, we **close the reading end of fd_parent pipe** and **close the writing end of the fd_child pipe**.
- In the **child process**, we **close the writing end of fd_parent pipe** and **close the reading end of the fd_child pipe**.
- We get the input string from the user and **write the string to the fd_parent pipe in the parent process**.
- We **read the string from the fd_parent pipe** and **reverse the string in the child process**.
- Once the string is reversed, we **write the string back to the parent process using the fd_child pipe**.
- In the parent process, the **string returned by the child process is read from the fd_child pipe** and displayed back to the user.

C CODE:

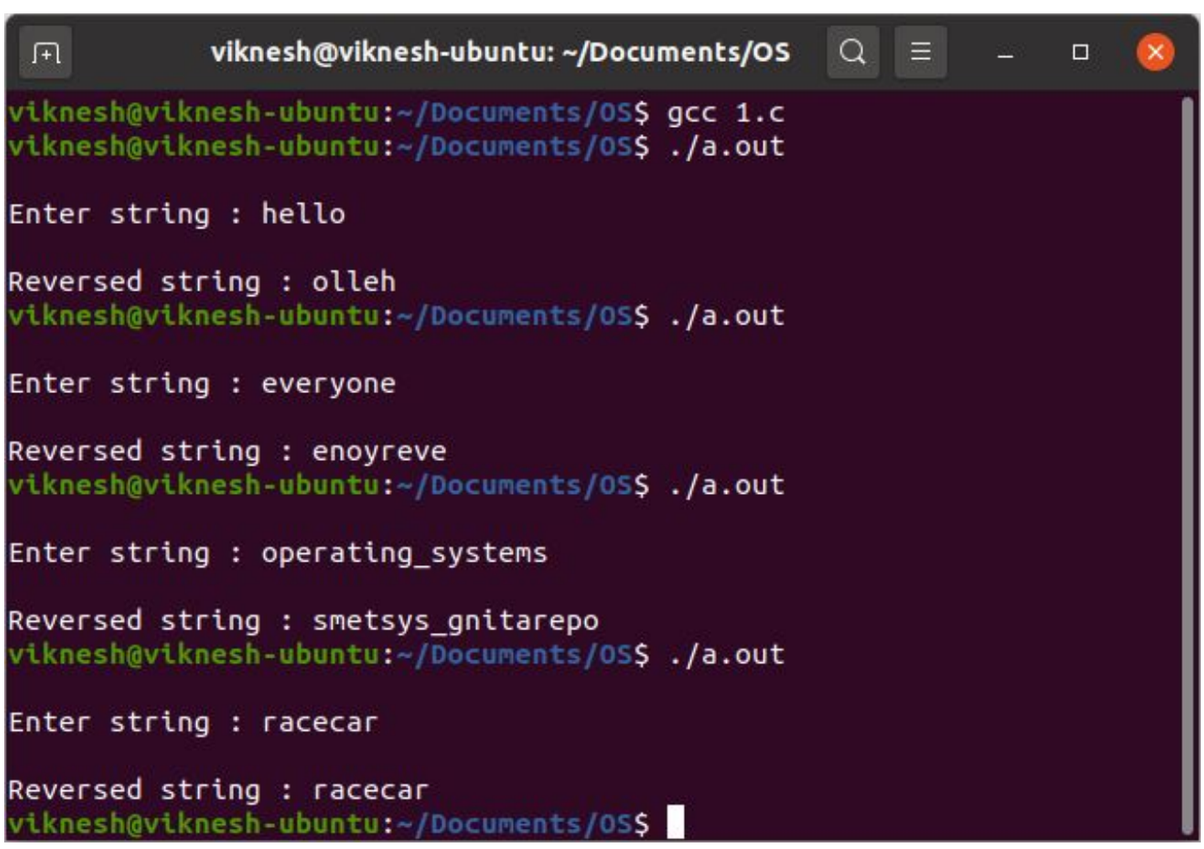
```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<sys/types.h>
#include<sys/wait.h>
#include<unistd.h>

#define MAXSIZE 1024

int main()
{
    char string[MAXSIZE];
    memset(string,0,MAXSIZE);
    int fd_parent[2];//Parent writes and Child Reads
    int fd_child[2];//Child writes and Parent Reads
    if(pipe(fd_parent)==-1)
    {
            perror("Pipe failed...\n");
            exit(0);
    }
    if(pipe(fd_child)==-1)
    {
            perror("Pipe failed...\n");
            exit(0);
    }
    pid_t pid;
    pid=fork();
    if(pid>0)
    {
            close(fd_parent[0]);
            close(fd_child[1]);
```

```
        printf("\nEnter string : ");
        scanf("%s",string);
        write(fd_parent[1],string,strlen(string)+1);
        read(fd_child[0],string,MAXSIZE);
        printf("\nReversed string : %s\n",string);
    }
    else if(pid==0)
    {
        close(fd_parent[1]);
        close(fd_child[0]);
        read(fd_parent[0],string,MAXSIZE);
        int len=strlen(string);
        for(int i=0;i<len/2;++i)
        {
            char temp=string[i];
            string[i]=string[len-i-1];
            string[len-i-1]=temp;
        }
        write(fd_child[1],string,strlen(string)+1);
        exit(0);
    }
    else
    {
        perror("Fork failed...\n");
        exit(0);
    }
    return 0;
}
```

OUTPUT:



2) Parent sets up string 1 and child sets up string 2. String 2 concatenated to string 1 at parent end and then read back at the child end.

LOGIC:

- We create two pipes - **fd_parent** and **fd_child**.
- **fd_parent** allows the **Parent process to write** and the **Child process to read**.
- **fd_child** allows the **Child process to write** and the **Parent process to read**.
- In the **parent process**, we **close the reading end of fd_parent pipe** and **close the writing end of the fd_child pipe**.
- In the **child process**, we **close the writing end of fd_parent pipe** and **close the reading end of the fd_child pipe**.
- We get the **string1 and string2 from the user** and **write the string2 to the fd_child pipe in the child process**.
- We **read the string from the fd_child pipe** and **concatenate both strings separated by a space between them in the parent process**.
- Once the strings are concatenated, we **write the string back to the child process using the fd_parent pipe** and **wait for the child process** to finish its execution.
- In the child process, the **string returned by the parent process is read from the fd_parent pipe** and displayed back to the user after which the child process is terminated.

C CODE:

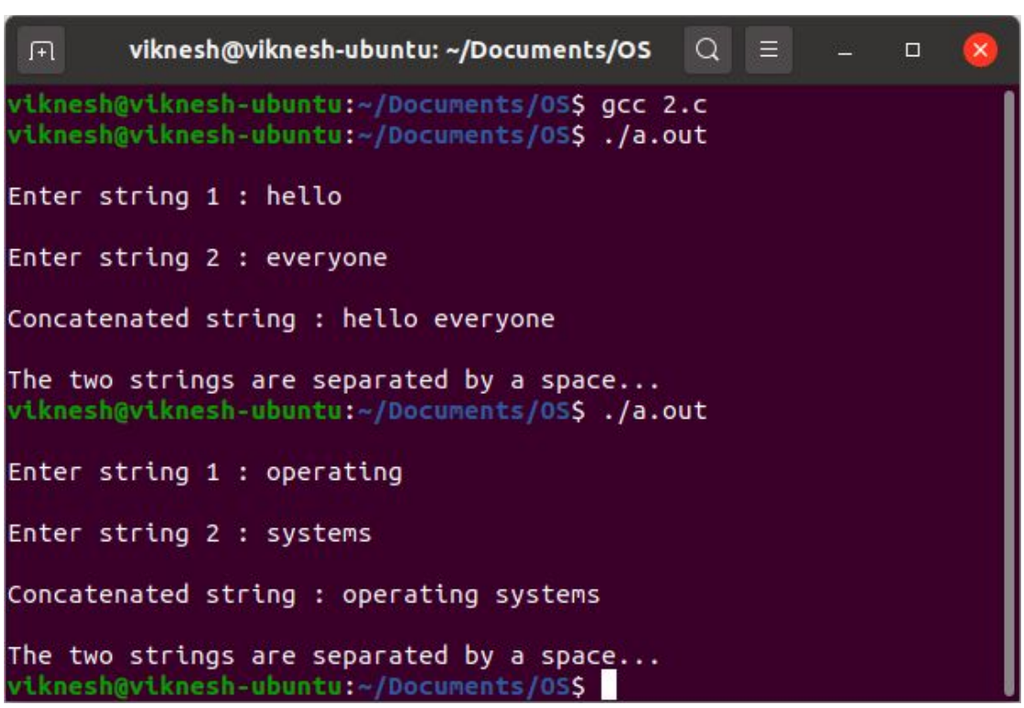
```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<sys/types.h>
#include<sys/wait.h>
#include<unistd.h>

#define MAXSIZE 1024

int main()
{
    char string1[MAXSIZE],string2[MAXSIZE];
    memset(string1,0,MAXSIZE);
    memset(string2,0,MAXSIZE);
    int fd_parent[2];//Parent writes and Child Reads
    int fd_child[2];//Child writes and Parent Reads
    if(pipe(fd_parent)==-1)
    {
            perror("Pipe failed...\n");
            exit(0);
    }
    if(pipe(fd_child)==-1)
    {
            perror("Pipe failed...\n");
            exit(0);
    }
    printf("\nEnter string 1 : ");
    scanf("%s",string1);
    printf("\n");
```

```
    pid_t pid;
    pid=fork();
    if(pid>0)
    {
            close(fd_parent[0]);
            close(fd_child[1]);
            read(fd_child[0],string2,MAXSIZE);
            close(fd_child[0]);
            strcat(string1," ");
            strcat(string1,string2);
            write(fd_parent[1],string1,strlen(string1)+1);
            wait(NULL);
    }
    else if(pid==0)
    {
            close(fd_parent[1]);
            close(fd_child[0]);
            printf("Enter string 2 : ");
            scanf("%s",string2);
            printf("\n");
            write(fd_child[1],string2,strlen(string2)+1);
            close(fd_child[1]);
            read(fd_parent[0],string1,MAXSIZE);
            printf("Concatenated string : %s\n",string1);
            printf("\nThe two strings are separated by a space...\n");
            exit(0);
    }
    else
    {
            perror("Fork failed...\n");
            exit(0);
    }
    return 0;
}
```

OUTPUT:



3) Substring generation at child end of a string setup at parent process end.

LOGIC:

- Since one-way communication is required, we create only one pipe - **fd**.
- **fd** allows the **Parent process to write** and the **Child process to read**.
- In the **parent process**, we **close the reading end of fd pipe**.
- In the **child process**, we **close the writing end of fd_parent pipe**.
- We get the **string from the user** and **write the string to the fd pipe in the parent process**.
- We also get the **start index** and **end index** and **write it to the fd pipe in the parent process**.
- We **read the string, start index and end index from the fd pipe** and **generate the substring in the child process**.
- In the child process, the **substring generated is displayed back to the user** after which the child process is terminated.

C CODE:

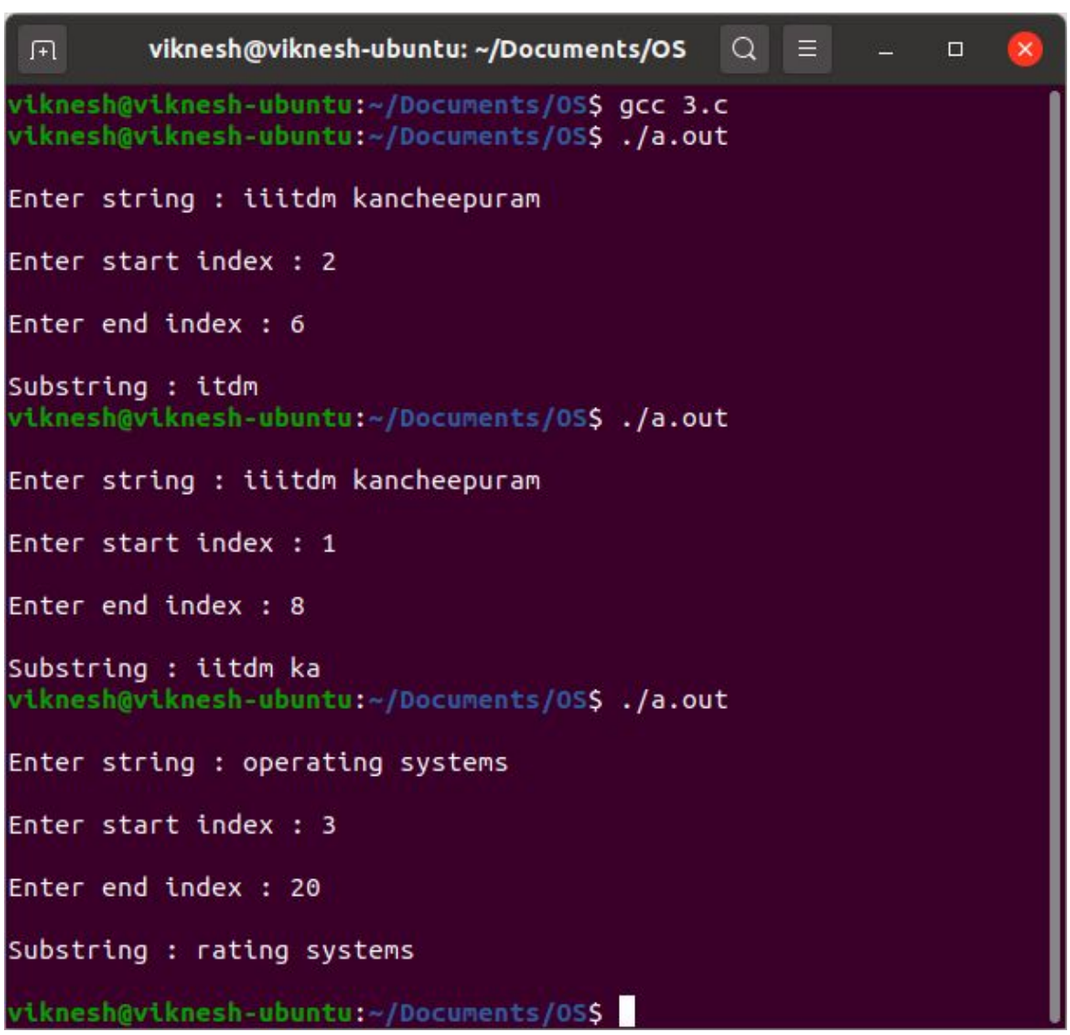
```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<sys/types.h>
#include<sys/wait.h>
#include<unistd.h>

#define MAXSIZE 1024

int main()
{
    char string[MAXSIZE];
    int start_index,end_index;
    memset(string,0,MAXSIZE);
    int fd[2];//Parent writes and Child Reads
    if(pipe(fd)==-1)
    {
        perror("Pipe failed...\n");
        exit(0);
    }
    pid_t pid;
    pid=fork();
    if(pid>0)
    {
        close(fd[0]);
        printf("\nEnter string : ");
        fgets(string,MAXSIZE,stdin);
        printf("\nEnter start index : ");
        scanf("%d",&start_index);
        printf("\nEnter end index : ");
        scanf("%d",&end_index);
        printf("\n");
        write(fd[1],string,MAXSIZE);
        write(fd[1],&start_index,sizeof(start_index));
        write(fd[1],&end_index,sizeof(end_index));
        wait(NULL);
```

```
    }
    else if(pid==0)
    {
        close(fd[1]);
        read(fd[0],string,MAXSIZE);
        read(fd[0],&start_index,sizeof(start_index));
        read(fd[0],&end_index,sizeof(end_index));
        int length=strlen(string);
        printf("Substring : ");
        int i=start_index;
        while((i<=end_index) && (string[i]!='\0'))
        {
            printf("%c",string[i]);
            ++i;
        }
        printf("\n");
        exit(0);
    }
    else
    {
        perror("Fork failed...\n");
        exit(0);
    }
    return 0;
}
```

OUTPUT:



4) String reversal and palindrome check using pipes / shared memory.

LOGIC:

- We create two pipes - **fd_parent** and **fd_child**.
- **fd_parent** allows the **Parent process to write** and the **Child process to read**.
- **fd_child** allows the **Child process to write** and the **Parent process to read**.
- In the **parent process**, we **close the reading end of fd_parent pipe** and **close the writing end of the fd_child pipe**.
- In the **child process**, we **close the writing end of fd_parent pipe** and **close the reading end of the fd_child pipe**.
- We get the **string from the user** and **write the string to the fd_parent pipe in the parent process**.
- We **read the string from the fd_parent pipe** and **reverse the string in the child process**.
- Once the string is reversed, we **write the string back to the parent process using the fd_child pipe** and terminate the child process.
- The **reversed string is read from the fd_child pipe** and displayed to the user in the parent process.
- In the **parent process**, we **check if the string is a palindrome or not**.

C CODE:

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

#define MAXSIZE 1024

void convert(char str1[],char str2[])
{
    int length=strlen(str1);
    int j=0;
    for(int i=0;i<length;++i)
    {
        if((str1[i]>=65) && (str1[i]<=90))
        {
            str2[j]=str1[i]+32;
            ++j;
        }
        else if((str1[i]>=97) && (str1[i]<=122))
        {
            str2[j]=str1[i];
            ++j;
        }
    }
    str2[j]='\0';
}

void palindrome_check(char str1[],char str2[])
{
```

```
    char string[MAXSIZE],rev_string[MAXSIZE];
    convert(str1,string);
    convert(str2,rev_string);
    if(strcmp(string,rev_string)==0)
    {
            printf("\nIT IS A PALINDROME...\n");
    }
    else
    {
            printf("\nNOT A PALINDROME...\n");
    }
}

int main()
{
    char string[MAXSIZE],rev_string[MAXSIZE];
    memset(string,0,MAXSIZE);
    memset(rev_string,0,MAXSIZE);
    int fd_parent[2];//Parent writes and Child Reads
    int fd_child[2];//Child writes and Parent Reads
    if(pipe(fd_parent)==-1)
    {
            perror("Pipe failed...\n");
            exit(0);
    }
    if(pipe(fd_child)==-1)
    {
            perror("Pipe failed...\n");
            exit(0);
    }
    pid_t pid;
    pid=fork();
    if(pid>0)
    {
            close(fd_parent[0]);
            close(fd_child[1]);
            printf("\nEnter a string : ");
            fgets(string,MAXSIZE,stdin);
            write(fd_parent[1],string,strlen(string)+1);
            read(fd_child[0],rev_string,MAXSIZE);
            printf("\nReversed string : %s\n",rev_string);
            palindrome_check(string,rev_string);
    }
    else if(pid==0)
    {
            close(fd_parent[1]);
            close(fd_child[0]);
            read(fd_parent[0],string,MAXSIZE);
            int len=strlen(string);
            for(int i=0;i<len;++i)
            {
                    rev_string[len-i-1]=string[i];
            }
            rev_string[len]='\0';
            write(fd_child[1],rev_string,strlen(rev_string)+1);
            exit(0);
    }
```

```
    else
    {
            perror("Fork failed...\n");
            exit(0);
    }
    return 0;
}
```

OUTPUT: