OPERATING SYSTEMS PRACTICE (COM301P)

Name: Vinayak Sethi **Roll No:** COE18B061

Assignment 5

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

Filename: Q1_StringReversal.c

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

void ReverseString(char *string, int begin, int end);

int main()
{
   int pipefd1[2], pipefd2[2]; //0 -> read, 1 -> write

   char input_str[100];
   char reverse_str[100];
   pid_t pid;

   if(pipe(pipefd1) == -1)
   {
       printf("Unable to create pipe 1..\n");
       exit(EXIT_FAILURE);
   }

   if(pipe(pipefd2) == -1)
   {
```

```
        printf("Unable to create pipe 2..\n");
        exit(EXIT_FAILURE);
    }

    pid = fork();

    if(pid == 0) //child block
    {
        close(pipefd1[1]); //close writing end of parent pipe

        //Read a string from parent pipe
        read(pipefd1[0], reverse_str, 100);

        printf("\nIn Child :- String read from parent pipe: %s\n",
reverse_str);

        //Reverse the string given by parent
        ReverseString(reverse_str, 0, strlen(reverse_str) - 1);

        //close both reading ends
        close(pipefd1[0]);
        close(pipefd2[0]);

        //Write reversed string and close the writing end
        write(pipefd2[1], reverse_str, 100);
        close(pipefd2[1]);

    }

    else //parent block
    {
        close(pipefd1[0]); // Close reading end of parent pipe

        printf("\nIn Parent :- Enter a string: ");
        gets(input_str);

        //write input string and close writing end of parent pipe
        write(pipefd1[1], input_str, strlen(input_str) + 1);
```

```
        close(pipefd1[1]);

        //wait for child to send the reverse string
        wait(NULL);

        close(pipefd2[1]); //close writing end of second pipe

        //read string from child, print it and close the reading end
        read(pipefd2[0], reverse_str, 100);
        printf("\nIn Parent :- Reversed string from child side:
%s\n\n", reverse_str);
        close(pipefd2[0]);
    }

    return 0;
}

void ReverseString(char *string, int begin, int end)
{
    char c;

    if(begin >= end)
        return;

    c = *(string + begin);
    *(string + begin) = *(string + end);
    *(string + end) = c;

    ReverseString(string, ++begin , --end);
}
```

Output:

```
vinayak@vinayak-Swift-SF315-52G: ~/Documents/OS/Lab/Lab5
File  Edit  View  Search  Terminal  Help
vinayak@vinayak-Swift-SF315-52G:~/Documents/OS/Lab/Lab5$ make Q1_StringReversal
make: 'Q1_StringReversal' is up to date.
vinayak@vinayak-Swift-SF315-52G:~/Documents/OS/Lab/Lab5$ ./Q1_StringReversal

In Parent :- Enter a string: Hello World

In Child :- String read from parent pipe: Hello World

In Parent :- Reversed string from child side: dlroW olleH

vinayak@vinayak-Swift-SF315-52G:~/Documents/OS/Lab/Lab5$ ./Q1_StringReversal

In Parent :- Enter a string: Stay Safe :)

In Child :- String read from parent pipe: Stay Safe :)

In Parent :- Reversed string from child side: ): efaS yatS

vinayak@vinayak-Swift-SF315-52G:~/Documents/OS/Lab/Lab5$
```

Explanation:

pipe() is used for passing information from one process to another. pipe() is unidirectional therefore, for two-way communication between processes, two pipes can be set up, one for each direction.

Inside Parent Process: We firstly close the reading end of the first pipe (pipefd1[0]) then write the string through the writing end of the pipe (pipefd1[1]). Now the parent will **wait** until the child process is finished. After the child process, the parent will close the writing end of the second pipe(pipefd2[1]) and read the string through the reading end of pipe (pipefd2[0]) and print the string.

Inside Child Process: Child reads the string sent by the parent process by closing the writing end of the first pipe (pipefd1[1]) and after reading, it

reverses the string and passes the string to the parent process via pipefd2 pipe and will exit.

(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.

Filename: Q2_StringConcatination.c

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

int main()
{
   int pipefd1[2], pipefd2[2]; //0 -> read, 1 -> write

   char input_str1[100];
   char input_str2[100];
   char concat_str[100];
   pid_t pid;

   if(pipe(pipefd1) == -1)
   {
       printf("Unable to create pipe 1..\n");
       exit(EXIT_FAILURE);
   }

   if(pipe(pipefd2) == -1)
   {
       printf("Unable to create pipe 2..\n");
       exit(EXIT_FAILURE);
   }

   pid = fork();
```

```
   if(pid == 0) //child block
   {
       sleep(5);
       close(pipefd2[0]); //close reading end of child pipe
       close(pipefd1[1]); //close writing end of parent pipe

       printf("\nIn Child :- Enter string 2: ");
       gets(input_str2);

       //write the input string to child pipe and close it
       write(pipefd2[1], input_str2, strlen(input_str2) + 1);
       close(pipefd2[1]);

       //read string from parent, print it and close the reading end
       read(pipefd1[0], concat_str, 100);
       printf("\nIn Child :- Concatenated string from parent side:
%s\n\n", concat_str);
       close(pipefd1[0]);
   }

   else //parent block
   {
       close(pipefd1[0]); //close reading end of parent pipe
       close(pipefd2[1]); //close writing end of child pipe

       printf("\nIn Parent :- Enter string 1: ");
       gets(input_str1);

       //read the input string by child from the child pipe
       read(pipefd2[0], input_str2, 100);
       printf("\nIn Parent :- Strings to be concatenated are %s and
%s\n", input_str1, input_str2);

       //Concat the child string to the parent string
       strcat(input_str1, input_str2);

       close(pipefd2[0]); //close reading end of second pipe
```

```
        //Write concatinated string and close the writing end of
parent pipe
        write(pipefd1[1], input_str1, 100);
        close(pipefd1[1]);
    }

    return 0;
}
```

Output:

```
vinayak@vinayak-Swift-SF315-52G: ~/Documents/OS/Lab/Lab5
File  Edit  View  Search  Terminal  Help
vinayak@vinayak-Swift-SF315-52G:~/Documents/OS/Lab/Lab5$ make Q2_StringConcatenation
make: 'Q2_StringConcatenation' is up to date.
vinayak@vinayak-Swift-SF315-52G:~/Documents/OS/Lab/Lab5$ ./Q2_StringConcatenation

In Parent :- Enter string 1: Hello

In Child :- Enter string 2: World

In Parent :- Strings to be concatenated are Hello and World

In Child :- Concatenated string from parent side: HelloWorld

vinayak@vinayak-Swift-SF315-52G:~/Documents/OS/Lab/Lab5$ ./Q2_StringConcatenation

In Parent :- Enter string 1: Vinayak

In Child :- Enter string 2:  Sethi

In Parent :- Strings to be concatenated are Vinayak and  Sethi

In Child :- Concatenated string from parent side: Vinayak Sethi

vinayak@vinayak-Swift-SF315-52G:~/Documents/OS/Lab/Lab5$
```

Explanation:

Inside Parent Process: We firstly close the reading end of the first pipe (pipefd1[0]) and the writing end of the second pipe (pipefd2[1]) then take the first input string from user and reading the another string given by user

in child block and concatenated it and the parent will close the reading end of the second pipe(pipefd2[0]) and sends the concatenated string to child block by writing concatenated string on first pipe(pipefd1[1]) and close its writing end(pipefd1[1]).

Inside Child Process: It takes string 2 as input from the user and sends it to the parent pipe. Then Child reads the string sent by the parent process by closing the writing end of the first pipe (pipefd1[1]) and after reading, it prints the concatenated string sent by the first pipe and closes its reading end(pipefd1[0]).

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

Filename: Q3_SubstringGeneration.c

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

void GenerateSubstr(char *input_str, char *substr, int start, int
end);

int main()
{
   int pipefd1[2], pipefd2[2]; //0 -> read, 1 -> write

   char input_str[100];
   char substr[100];
   int start, end;
   pid_t pid;

   if(pipe(pipefd1) == -1)
   {
```

```
        printf("Unable to create pipe 1..\n");
        exit(EXIT_FAILURE);
    }

    if(pipe(pipefd2) == -1)
    {
        printf("Unable to create pipe 2..\n");
        exit(EXIT_FAILURE);
    }

    pid = fork();

    if(pid == 0) //child block
    {
        close(pipefd1[1]); //close writing end of parent pipe

        //Read a string from parent pipe
        read(pipefd1[0], input_str, 100);

        printf("\nIn Child :- String read from parent pipe: %s\n",
input_str);

        printf("\nIn Child :- Enter the start index for substring: ");
        scanf("%d", &start);
        printf("In Child :- Enter the end index for substring: ");
        scanf("%d", &end);

        //Generate the substring for the string given by parent
        GenerateSubstr(input_str, substr, start, end);

        //close both reading ends
        close(pipefd1[0]);
        close(pipefd2[0]);

        //Write substring and close the writing end
        write(pipefd2[1], substr, 100);
        close(pipefd2[1]);
```

```
    }

    else //parent block
    {
        close(pipefd1[0]); // Close reading end of parent pipe

        printf("\nIn Parent :- Enter a string: ");
        gets(input_str);

        //write input string and close writing end of parent pipe
        write(pipefd1[1], input_str, strlen(input_str) + 1);
        close(pipefd1[1]);

        //wait for child to send the substring
        wait(NULL);

        close(pipefd2[1]); //close writing end of second pipe

        //read substring from child, print it and close the reading
end
        read(pipefd2[0], substr, 100);
        printf("\nIn Parent :- Substring generated from child side:
%s\n\n", substr);
        close(pipefd2[0]);
    }

    return 0;
}

void GenerateSubstr(char *input_str, char *substr, int start, int
end)
{
    int count = 0;

    while(count < (end - start) + 1)
    {
        substr[count] = input_str[start+count];
        count ++;
```

```
    }
    substr[count] = '\0';
}
```

Output:

```
vinayak@vinayak-Swift-SF315-52G: ~/Documents/OS/Lab/Lab5
File  Edit  View  Search  Terminal  Help
vinayak@vinayak-Swift-SF315-52G:~/Documents/OS/Lab/Lab5$ make Q3_SubstringGeneration
make: 'Q3_SubstringGeneration' is up to date.
vinayak@vinayak-Swift-SF315-52G:~/Documents/OS/Lab/Lab5$ ./Q3_SubstringGeneration

In Parent :- Enter a string: Hello World!

In Child :- String read from parent pipe: Hello World!

In Child :- Enter the start index for substring: 2
In Child :- Enter the end index for substring: 8

In Parent :- Substring generated from child side: llo Wor

vinayak@vinayak-Swift-SF315-52G:~/Documents/OS/Lab/Lab5$ ./Q3_SubstringGeneration

In Parent :- Enter a string: India is the best

In Child :- String read from parent pipe: India is the best

In Child :- Enter the start index for substring: 1
In Child :- Enter the end index for substring: 15

In Parent :- Substring generated from child side: ndia is the bes

vinayak@vinayak-Swift-SF315-52G:~/Documents/OS/Lab/Lab5$
```

Explanation:

Inside Parent Process: We firstly close the reading end of the first pipe (pipefd1[0]) then write the string through the writing end of the pipe (pipefd1[1]). Now the parent will **wait** until the child process is finished. After the child process, the parent will close the writing end of the second pipe(pipefd2[1]) and read the string through the reading end of pipe (pipefd2[0]) and print the string.

Inside Child Process: Child reads the string sent by the parent process by closing the writing end of the first pipe (pipefd1[1]) and after reading, it generates a substring from a starting index to the ending index given by the user, and passes the string to the parent process via pipefd2 pipe and will exit.

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

Filename: Q4_PalindromeCheck.c

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

void ReverseString(char *string, int begin, int end);
int palindrome(char *string1, char *string2);

int main()
{
   int pipefd1[2], pipefd2[2]; //0 -> read, 1 -> write
   int result;
   char input_str[100];
   char reverse_str[100];
   pid_t pid;

   if(pipe(pipefd1) == -1)
   {
       printf("Unable to create pipe 1..\n");
       exit(EXIT_FAILURE);
   }

   if(pipe(pipefd2) == -1)
   {
       printf("Unable to create pipe 2..\n");
```

```
        exit(EXIT_FAILURE);
    }

    pid = fork();

    if(pid == 0) //child block
    {
        close(pipefd1[1]); //close writing end of parent pipe

        //Read a string from parent pipe
        read(pipefd1[0], reverse_str, 100);

        printf("\nIn Child :- String read from parent pipe: %s\n",
reverse_str);

        //Reverse the string given by parent
        ReverseString(reverse_str, 0, strlen(reverse_str) - 1);

        //close both reading ends
        close(pipefd1[0]);
        close(pipefd2[0]);

        //Write reversed string and close the writing end
        write(pipefd2[1], reverse_str, 100);
        close(pipefd2[1]);

    }

    else //parent block
    {
        close(pipefd1[0]); // Close reading end of parent pipe

        printf("\nIn Parent :- Enter a string: ");
        gets(input_str);

        //write input string and close writing end of parent pipe
        write(pipefd1[1], input_str, strlen(input_str) + 1);
        close(pipefd1[1]);
```

```
        //wait for child to send the reverse string
        wait(NULL);

        close(pipefd2[1]); //close writing end of second pipe

        //read string from child, print it
        read(pipefd2[0], reverse_str, 100);
        printf("\nIn Parent :- Reversed string from child side: %s\n",
reverse_str);

        printf("\nIn Parent :- Palindrome Check..\n");

        //check given string is palindrome or not
        result = palindrome(input_str, reverse_str);
        if(result)
            printf("\t     Given string %s, is a palindrome\n\n",
input_str);
        else
            printf("\t     Given string %s, is not a palindrome\n\n",
input_str);

        //close the reading end of child
        close(pipefd2[0]);
    }

    return 0;
}

void ReverseString(char *string, int begin, int end)
{
    char c;

    if(begin >= end)
        return;

    c = *(string + begin);
    *(string + begin) = *(string + end);
```

```
    *(string + end) = c;

    ReverseString(string, ++begin , --end);
}

int palindrome(char *string1, char *string2)
{
    return strcmp(string1, string2) == 0;
}
```

Output:

```
vinayak@vinayak-Swift-SF315-52G: ~/Documents/OS/Lab/Lab5
File  Edit  View  Search  Terminal  Help
vinayak@vinayak-Swift-SF315-52G:~/Documents/OS/Lab/Lab5$ make Q4_PalindromeCheck
make: 'Q4_PalindromeCheck' is up to date.
vinayak@vinayak-Swift-SF315-52G:~/Documents/OS/Lab/Lab5$ ./Q4_PalindromeCheck

In Parent :- Enter a string: aibohphobia

In Child :- String read from parent pipe: aibohphobia

In Parent :- Reversed string from child side: aibohphobia

In Parent :- Palindrome Check..
             Given string aibohphobia, is a palindrome

vinayak@vinayak-Swift-SF315-52G:~/Documents/OS/Lab/Lab5$ ./Q4_PalindromeCheck

In Parent :- Enter a string: Vinayak

In Child :- String read from parent pipe: Vinayak

In Parent :- Reversed string from child side: kayaniV

In Parent :- Palindrome Check..
             Given string Vinayak, is not a palindrome

vinayak@vinayak-Swift-SF315-52G:~/Documents/OS/Lab/Lab5$
```

Explanation:

Inside Parent Process: We firstly close the reading end of the first pipe (pipefd1[0]) then write the string through the writing end of the pipe

(pipefd1[1]). Now the parent will **wait** until the child process is finished. After the child process, the parent will close the writing end of the second pipe(pipefd2[1]) and read the string through the reading end of pipe (pipefd2[0]) and check whether the given string is palindrome or not and print the corresponding result of it.

Inside Child Process: Child reads the string sent by the parent process by closing the writing end of the first pipe (pipefd1[1]) and after reading, it reverses the string and passes the string to the parent process via pipefd2 pipe and will exit.