**OPERATING SYSTEM ASSIGNMENT # 2**

**TASK 2 REPORT**

**GROUP MEMBERS:**

* Muhibullah Sherwani
* Faraz Ahmed
* Faris Jamil

Class ID: 101348

Date: 21-April-2019

**TASK 2 CODE EXPLANATION:**

**Process Creation:**

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/types.h>

#include<string.h>

#include<sys/wait.h>

int main()

{

// We use two pipes

// First pipe to send input string from parent

// Second pipe to send concatenated string from child

int fd1[2]; // Used to store two ends of first pipe

int fd2[2]; // Used to store two ends of second pipe

char fixed\_str[] = "MFF Program";

char input\_str[100];

pid\_t p;

if (pipe(fd1)==-1)

{

fprintf(stderr, "Pipe Failed" );

return 1;

}

if (pipe(fd2)==-1)

{

fprintf(stderr, "Pipe Failed" );

return 1;

}

printf("Enter Input: \n");

scanf("%s", input\_str);

p = fork();

if (p < 0)

{

fprintf(stderr, "fork Failed" );

return 1;

}

// Parent process

else if (p > 0)

{

char concat\_str[100];

close(fd1[0]); // Close reading end of first pipe

// Write input string and close writing end of first

// pipe.

write(fd1[1], input\_str, strlen(input\_str)+1);

close(fd1[1]);

// Wait for child to send a string

wait(NULL);

close(fd2[1]); // Close writing end of second pipe

// Read string from child, print it and close

// reading end.

read(fd2[0], concat\_str, 100);

// Concatenated string Show With Names

printf("Input string Showing %s\n", concat\_str);

close(fd2[0]);

}

// child process

else

{

close(fd1[1]); // Close writing end of first pipe

// Read a string using first pipe

char concat\_str[100];

read(fd1[0], concat\_str, 100);

// Concatenate a fixed string with it

int k = strlen(concat\_str);

int i;

for (i=0; i<strlen(fixed\_str); i++)

concat\_str[k++] = fixed\_str[i];

concat\_str[k] = '\0'; // string ends with '\0'

// Close both reading ends

close(fd1[0]);

close(fd2[0]);

// Write concatenated string and close writing end

write(fd2[1], concat\_str, strlen(concat\_str)+1);

close(fd2[1]);

exit(0);

}

}

**I/O Redirection:**

#include <unistd.h>

#include <sys/wait.h>

#include <stdio.h>

#include <stdlib.h>

int main(int argc, char \*argv[]) {

int fds[2]; // an array that will hold two file descriptors

pipe(fds); // populates fds with two file descriptors

pid\_t pid = fork(); // create child process that is a clone of the parent

char input\_str[100];

if (pid == 0) { // if pid == 0, then this is the child process

dup2(fds[0], STDIN\_FILENO); // fds[0] (the read end of pipe) donates its data to file descriptor 0

close(fds[0]); // file descriptor no longer needed in child since stdin is a copy

close(fds[1]); // file descriptor unused in child

char \*argv[] = {(char \*)"sort", NULL}; // create argument vector

if (execvp(argv[0], argv) < 0) exit(0); // run sort command (exit if something went wrong)

}

// if we reach here, we are in parent process

close(fds[0]); // file descriptor unused in parent

const char \*words[] = {"banana", "peach", "apple" , "mango"};

// write input to the writable file descriptor so it can be read in from child:

size\_t numwords = sizeof(words)/sizeof(words[0]);

for (size\_t i = 0; i < numwords; i++) {

dprintf(fds[1], "%s\n", words[i]);

}

// send EOF so child can continue (child blocks until all input has been processed):

close(fds[1]);

int status;

pid\_t wpid = waitpid(pid, &status, 0); // wait for child to finish before exiting

return wpid == pid && WIFEXITED(status) ? WEXITSTATUS(status) : -1;

}

**Redirection And Compare:**

#include <unistd.h>

#include <sys/wait.h>

#include <stdio.h>

#include <stdlib.h>

int main(int argc, char \*argv[]) {

// if first command in pipeline has input redirection

if (hasInputFile && is1stCommand) {

int fdin = open(inputFile, O\_RDONLY, 0644);

dup2(fdin, STDIN\_FILENO);

close(fdin);

}

// if last command in pipeline has output redirection

if (hasOutputFile && isLastCommand) {

int fdout = open(outputFile, O\_WRONLY | O\_CREAT | O\_TRUNC, 0644);

dup2(fdout, STDOUT\_FILENO);

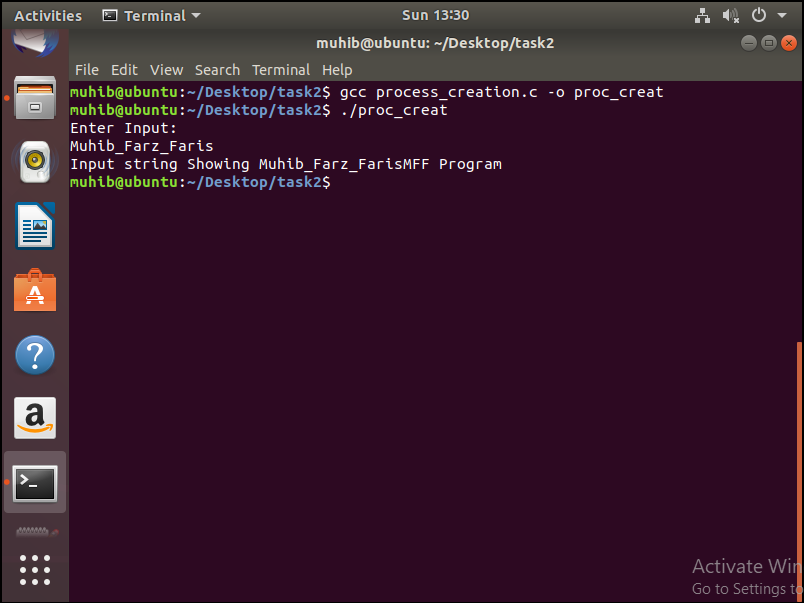
close(fdout);

}

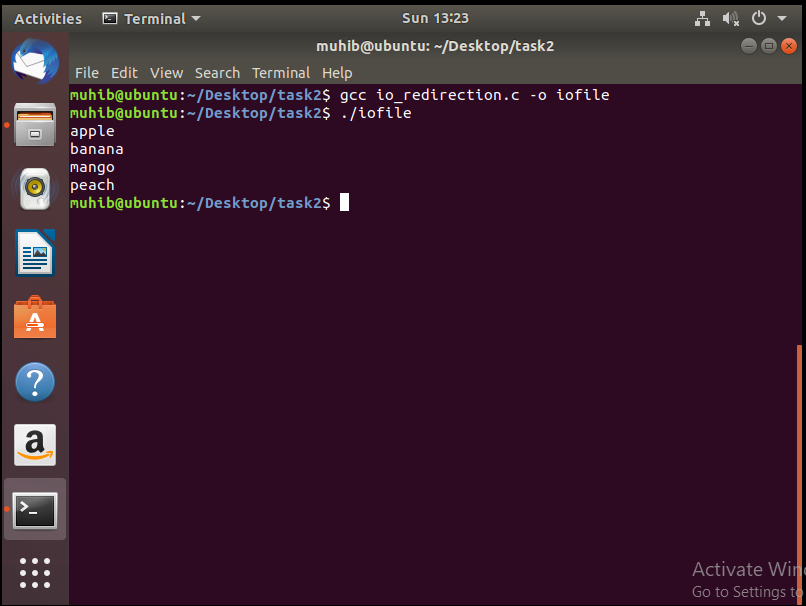
}

**TASK 2 CODE EXCECUTION SCREENSHOTS:**

**PROCESS CREATION:**

****

**I/O REDIRECTION:**

****