**ASSIGNMENT -4**

**IMPLEMENTATION OF PIPE AND SHELL**

Name – Ashutosh Soni

Id – 2018ucp1505

Question: Three processes have to be created, first process reads data from a user specified file and through pipe sends this data to second process. Second process shall trim this input by removing extra spaces (only one blank between two words but there should be no blank before punctuation marks and there should be one blank after punctuation marks. Third process shall change the case of first character of every word to capital and rest to small case. This process should print these characters. This process should count number of words and pass this to first process, which shall print this.

Answer:

**Program for the same is:**

#include<bits/stdc++.h>

#include<unistd.h>

using namespace std;

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

// used for three files for required purpose

int fd1[2]; // Send data from first to second process

int fd2[2]; // Send data from second to third process

int fd3[2]; // Send data from third to first process

char content[10000];

char read\_content[10000];

char spaceless\_content[10000];

char read\_spaceless\_content[10000];

char output\_content[10000];

int result[1];

// Creating three pipes

if(pipe(fd1) == -1){

cout<<"Unable to create pipe"<<endl;

return -1;

}

if(pipe(fd2) == -1){

cout<<"Unable to create pipe"<<endl;

return -1;

}

if(pipe(fd3) == -1){

cout<<"Unable to create pipe"<<endl;

return -1;

}

pid\_t pid1;

pid1 = fork();

if(pid1 < 0){

cout<<"Unable to create fork .."<<endl;

return -1;

}

else if(pid1 == 0){

// This process reads data from first pipe and trims the extra space in data

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

close(fd3[1]); // Close writing end of third pipe

close(fd2[0]); // Close reading end of second pipe

close(fd3[0]); // Close reading end of third pipe

read(fd1[0],read\_content,sizeof(read\_content));

int k=0;

for(int i=0;i<=strlen(read\_content);i++){

if(read\_content[i-1] == ' ' and read\_content[i] == ' '){

continue;

}

else{

spaceless\_content[k] = read\_content[i];

k++;

}

}

write(fd2[1],spaceless\_content,k-1);

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

cout<<"Action of second program is completed"<<endl;

cout<<endl;

}

else{

pid\_t pid2;

pid2 = fork();

if(pid2 < 0){

cout<<"Unable to create fork .."<<endl;

return -1;

}

else if(pid2 == 0){

// last Program

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

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

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

close(fd3[0]); // Close reading end of third pipe

read(fd2[0],read\_spaceless\_content,sizeof(read\_spaceless\_content));

int count=1;

for(int i=0;i<=strlen(read\_spaceless\_content);i++){

if(i==0){

output\_content[i] = read\_spaceless\_content[i];

}

else if(read\_spaceless\_content[i-1]==' '){

if(read\_spaceless\_content[i]>='a'){

output\_content[i] = read\_spaceless\_content[i]-('a' - 'A');

count++;

}

else{

output\_content[i] = read\_spaceless\_content[i];

}

}

else{

output\_content[i] = read\_spaceless\_content[i];

}

}

cout<<"The output after desired changes is "<<endl;

for(int i=0;i<strlen(read\_spaceless\_content);i++){

cout<<output\_content[i];

}

int num[1];

num[0] = count;

write(fd3[1],num,sizeof(num));

close(fd3[1]); // Close writing end of third file

cout<<endl;

cout<<"Action of third program is completed"<<endl;

cout<<endl;

}

else{

// This process reads data from a user specified file and through pipe sends this data to second process

// Reading data from file

string filename;

filename = "data.txt";

FILE \*file = fopen(filename.c\_str(),"r");;

int i=0;

while(!feof(file)){

char c =getc(file);

content[i] = c;

i++;

}

content[i] = '\0';

fclose(file);

// Reading done

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

close(fd3[1]); // Closing writing end of third pipe.

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

close(fd2[0]); // Closing reading end of second pipe.

write(fd1[1],content,i-1);

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

read(fd3[0],result,sizeof(result));

cout<<"Total number of words in the file is: ";

cout<<result[0]<<endl;

cout<<"Action of first program is completed"<<endl;

cout<<endl;

}

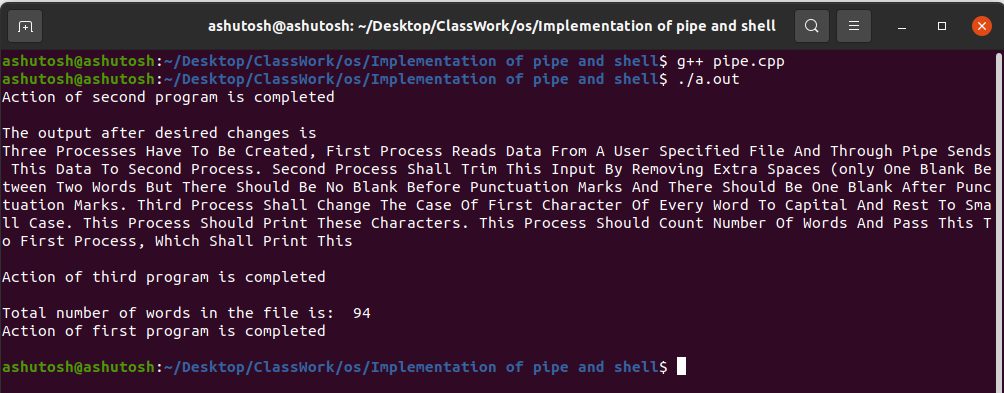
}

return 0;

}

**The data.txt file I am using is:**

Three processes have to be created, First process reads data from a user specified file and through pipe sends this data to second process. Second process shall trim this input by removing extra spaces (only one blank between two words but there should be no blank before punctuation marks and there should be one blank after punctuation marks. Third process shall change the case of first character of every word to capital and rest to small case. This process should print these characters. This process should count number of words and pass this to first process, which shall print this

Output of the program is:

Question 2: You all have used shell (a shell is created when you launch terminal). So in this part, you shall write code for a very simple shell. Your shell

1. prints out a prompt of your choice ( Some examples PathName>, PathName::, PathName>>>);

2. reads user input (expected commands of execution such as ls, rm, mkdir); These commands can have their parameters such as ls -l, ls -a, ls -al

3. parses the line into the program name and an array of parameters (separated by delimiters).

4. uses the fork() system call to spawn a new child process;

5. The child process then uses the exec() system call (or one of its variants) to launch the specified program;

6. The parent process (the shell) uses the wait() system call (or one of its variants) to wait for the child to terminate;

7. Once the child (the launched program) finishes, the shell repeats the loop by jumping to Step 1.

8. When user types exit, shell terminates.

Answer:

**Program for the following:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

/\* The array below will hold the arguments: args[0] is the command. \*/

static char\* args[512];

pid\_t pid;

int command\_pipe[2];

#define READ 0

#define WRITE 1

static int command(int input, int first, int last)

{

int pipettes[2];

/\* Invoke pipe \*/

pipe( pipettes );

pid = fork();

/\*

SCHEME:

STDIN --> O --> O --> O --> STDOUT

\*/

if (pid == 0) {

if (first == 1 && last == 0 && input == 0) {

// First command

dup2( pipettes[WRITE], STDOUT\_FILENO );

} else if (first == 0 && last == 0 && input != 0) {

// Middle command

dup2(input, STDIN\_FILENO);

dup2(pipettes[WRITE], STDOUT\_FILENO);

} else {

// Last command

dup2( input, STDIN\_FILENO );

}

if (execvp( args[0], args) == -1)

\_exit(EXIT\_FAILURE); // If child fails

}

if (input != 0)

close(input);

// Nothing more needs to be written

close(pipettes[WRITE]);

// If it's the last command, nothing more needs to be read

if (last == 1)

close(pipettes[READ]);

return pipettes[READ];

}

/\* Final cleanup, 'wait' for processes to terminate.

\* n : Number of times 'command' was invoked.

\*/

static void cleanup(int n)

{

int i;

for (i = 0; i < n; ++i)

wait(NULL);

}

static int run(char\* cmd, int input, int first, int last);

static char line[1024];

static int n = 0; /\* number of calls to 'command' \*/

int main()

{

printf("SIMPLE SHELL: Type 'exit' or send EOF to exit.\n");

while (1) {

/\* Print the command prompt \*/

printf("$> ");

fflush(NULL);

/\* Read a command line \*/

if (!fgets(line, 1024, stdin))

return 0;

int input = 0;

int first = 1;

char\* cmd = line;

char\* next = strchr(cmd, '|'); /\* Find first '|' \*/

while (next != NULL) {

/\* 'next' points to '|' \*/

\*next = '\0';

input = run(cmd, input, first, 0);

cmd = next + 1;

next = strchr(cmd, '|'); /\* Find next '|' \*/

first = 0;

}

input = run(cmd, input, first, 1);

cleanup(n);

n = 0;

}

return 0;

}

static void split(char\* cmd);

static int run(char\* cmd, int input, int first, int last)

{

split(cmd);

if (args[0] != NULL) {

if (strcmp(args[0], "exit") == 0)

exit(0);

n += 1;

return command(input, first, last);

}

return 0;

}

static char\* skipwhite(char\* s)

{

while (isspace(\*s)) ++s;

return s;

}

static void split(char\* cmd)

{

cmd = skipwhite(cmd);

char\* next = strchr(cmd, ' ');

int i = 0;

while(next != NULL) {

next[0] = '\0';

args[i] = cmd;

++i;

cmd = skipwhite(next + 1);

next = strchr(cmd, ' ');

}

if (cmd[0] != '\0') {

args[i] = cmd;

next = strchr(cmd, '\n');

next[0] = '\0';

++i;

}

args[i] = NULL;

}

**Output of the code is:**

