Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работ №2 по курсу**

**«Операционные системы»**

**Управление процессами в ОС**

Студент: Ефимов Александр Владимирович

Группа: М80 – 201Б-18

Вариант: 6

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2019

1. **Постановка задачи**

На вход программе подается команда интерпретатора команд. Программа должна произвести вывод команды заменяя знаки табуляции и space на знаки «\_\_\_» и «\_» соответственно.

1. **Описание программы**

На глобальном уровне программы объявляются две переменные: *BuffSize* – для задания буфера команды, и *CharSize* – размер одного символа.

При запуске *main* создаются две трубы: *fd1* для передачи команды из родителя в ребенка, и *fd2* – для передачи из ребенка в родителя. После чего происходит *fork()* – создание детского процесса. Родительский процесс получает на ввод строку, которую преобразует в формат для команды консоли, в то время как детский процесс копирует файловый дескриптор для вывода в трубу (с помощью *dup2*) и заменяет себя на процесс консоли (с помощью execv), который возвращает результат команды. Этот результат передается в родительский процесс с помощью *fd2*, где он преобразуется в соответствии с заданием.

1. **Набор testcases**

*Test.txt*

|  |
| --- |
| This is space delimited.  This is tab delimited. |

1. **Результаты выполнения тестов.**

rookstar@Refrigerator:~/Git/OS-Labs/os\_lab\_02/src$ make

gcc reformer.c -o reformer

rookstar@Refrigerator:~/Git/OS-Labs/os\_lab\_02/src$ hexdump -bc test.txt

0000000 124 150 151 163 040 151 163 040 163 160 141 143 145 040 144 145

0000000 T h i s i s s p a c e d e

0000010 154 151 155 151 164 145 144 056 012 124 150 151 163 011 151 163

0000010 l i m i t e d . \n T h i s \t i s

0000020 011 164 141 142 011 144 145 154 151 155 151 164 145 144 056 012

0000020 \t t a b \t d e l i m i t e d . \n

0000030

rookstar@Refrigerator:~/Git/OS-Labs/os\_lab\_02/src$ ./reformer

cat test.txt

This\_is\_space\_delimited.

This\_\_\_is\_\_\_tab\_\_\_delimited.

rookstar@Refrigerator:~/Git/OS-Labs/os\_lab\_02/src$ ./reformer

wc test.txt

\_2\_\_8\_48\_test.txt

rookstar@Refrigerator:~/Git/OS-Labs/os\_lab\_02/src$

1. **Листинг программы**

#include <unistd.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/wait.h>

const size\_t BUFF\_SIZE = 100;

const size\_t CharSize = sizeof(char);

int main()

{

int fd1[2]; //From parent to child

int fd2[2]; //From child to parent

if (pipe(fd1) == -1)

{

write(STDOUT\_FILENO, "Pipe creation failure.\n", sizeof("Pipe creation failure.\n")); //sizeof string always includes terminator. We don't need it.

exit(EXIT\_FAILURE);

}

if (pipe(fd2) == -1)

{

write(STDOUT\_FILENO, "Pipe creation failure.\n", sizeof("Pipe creation failure.\n"));

exit(EXIT\_FAILURE);

}

pid\_t pid = fork();

if (pid == -1)

{

write(STDOUT\_FILENO, "Failed to execute.\n", sizeof("Failed to execute.\n"));

exit(EXIT\_FAILURE);

}

if (pid == 0)

{

int size;

char com[BUFF\_SIZE];

//Get the command from parrent

read(fd1[0], &size, sizeof(int));

read(fd1[0], com, size);

close(fd2[0]);

dup2(fd2[1], STDOUT\_FILENO);

dup2(fd2[1], STDERR\_FILENO);

//Use the command and sent result to parent

char \*result[] = {"sh", "-c", com, NULL};

execv("/bin/sh", result);

close(fd2[1]);

}

else

{

close(fd1[0]); //We don't need to read from parent->child pipe

close(fd2[1]); //We don't need to write to child->parent pipe

int ByteRead = 0, i = 0;

char ComIn[BUFF\_SIZE];

while( (ByteRead = read(STDIN\_FILENO, &ComIn[i], CharSize)) > 0 ) // Get the command

{

if (ComIn[i] == '\n') close(STDIN\_FILENO);

++i;

}

ComIn[i++] = '\0'; // and close it

//Write the command to child

write(fd1[1], &i, sizeof(int));

write(fd1[1], ComIn, i);

close(fd1[1]);

//Wait for child

waitpid(pid, NULL, 0);

//Parsing process

char x = 0;

while( (ByteRead = read(fd2[0], &x, CharSize)) > 0)

{

if (x == ' ') write(STDOUT\_FILENO, "\_", sizeof("\_"));

else if (x == '\t') write(STDOUT\_FILENO, "\_\_\_", sizeof("\_\_\_"));

else write(STDOUT\_FILENO, &x, sizeof(char));

}

close(fd2[0]);

}

return EXIT\_SUCCESS;

}

1. **Вывод**

*Pipe* является основой для обмена данными между процессами, что позволяет разбить одну задачу на несколько процессов и занять неиспользуемую процессом память, что повышает эффективность.