Министерство образования и науки Российской Федерации

Новосибирский государственный технический университет

Кафедра программных систем и баз данных

Лабораторная работа №6 по дисциплине

«Управление ресурсами в вычислительных системах»

Факультет: ПМИ

Группа: ПМ-04

Студентки: Анисимова К.Н.

Сулейманова К.А.

Вариант: 4

Преподаватель: Быханов К.В.

Куликов И.М.

Новосибирск

2013

1. ***Цель работы***

Практическое освоение средств управления ресурсами ОС UNIX на основе разработки программы, моделирующей работу интерпретатора в плане создания процессов, реализующих команды в командной строке, их синхронизации и взаимодействию по данным.

1. ***Задание***

Составить программу, моделирующую работу Shell-интерпретатора при обработке командной строки

**tr -d "[p-z]" < a.txt | wc -c & wc -c < a.txt.**

При реализации программы путём выдачи сообщений информировать обо всех этапах ее работы (создан процесс, выполнение команды закончено и т.д.).

1. ***Решение***

*Алгоритм работы*

Программа, моделируя работу интерпретатора, выполняет следующие действия:

* + - 1. Преобразует и копирует информацию в канал из файла a.txt, удаляя при этом литеры [p-z].
      2. Подсчитывает количество символов в канале и файле a.txt, и выводит информацию на экран.

1. ***Средства программирования***

В данной работе были использованы *функции:*

int fork(); - системный вызов fork() используется для порождения нового процесса.

int wait(int \*status); - ожидание завершения процесса-потомка родительским процессом.

int exit(int status); - завершение функционирования процесса

Аргумент status является статусом завершения, который передается отцу процесса, если он выполнял системный вызов wait().

char \*name, \*arg0, \*arg1, ... , \*argn;

execl(name, arg0, arg1, ... , argn, 0) – осуществляется смена программы, определяющей функционирование данного процесса.

int fcntl(int fd, char command, int argument); - системный вызов fcntl () выполняет действия по разделению пользовательских дескрипторов в зависимости от пяти значений аргумента command, специфицированных в файле fcntl.h.

1. ***Текст программы***

#include <stdio.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

int main ()

{

int status, status1; //"Статус" для завершения

int fd; //Файловый дескриптор

int f1, f2, f3; //Идентификаторы процессов

int pd[2], p; //Канал для записи дескриптора

//Создаём канал

p = pipe(pd);

if (p == -1)

{

printf("ERROR!!!"); //Ошибка при создании канала

exit(1);

};

printf("\nRun1 Порождение процесса tr -d ''[p-z]'' < a.txt");

f1 = fork(); //Порождение процесса tr –d "[p-z]" <a.txt

if (f1 == 0) //Потомок

{

printf("\nI'm in f1");

printf("\n");

fd = open("a.txt",O\_RDONLY); //Открытие файла данных a.txt

close(0); //Закрытие стандартного файла ввода

fcntl(fd,F\_DUPFD,0); //Перенаправление ввода из файла

close(1); //Закрытие стандартного файла вывода

fcntl(pd[1],F\_DUPFD,1); //Перенаправление вывода в канал

//Выполнение tr –d "[p-z]" <a.txt

execl("/usr/bin/tr", "tr", "-d", "\"[p-z]\"", 0);

exit(0);

}

else  
 if (f1 == -1)

printf("\nERROR!");

else

printf("\nI'm not in f1!");

printf("\nEnd1 Завершение выполнение потомка tr -d ''[p-z]'' < a.txt");

wait(&status); //Ожидание окончания работы потомка

if (status)

return status;

close(pd[1]);

f2 = fork(); //Порождение процесса wc -c

if (f2 == 0) //Потомок

{

printf("\nI'm in f2");

close(0); //Закрытие стандартного файла ввода

fcntl(pd[0],F\_DUPFD,0); //Перенаправление ввода из канала

//Выполнение wc –c

printf("\nRun2 Результат выполнения tr -d ''[p-z]'' < a.txt | wc -c:");

printf("\n");

execl("/usr/bin/wc", "wc", "-c", 0);

exit(0);

}

else

if (f2 == -1)

printf("\nERROR!");

else

printf("\nI'm not in f2!");

f3 = fork(); //Порождение процесса wc -c <a.txt

if (f3 == 0) //Потомок

{

printf("\nI'm in f3");

fd = open("a.txt",O\_RDONLY);

close(0); //Закрытие стандартного файла ввода

fcntl(fd,F\_DUPFD,0); //Пренаправление ввода из файла

//Выполнение wc –c <a.txt

printf("\nRun3 Результат выполнения tr -d ''[p-z]'' < a.txt | wc -c <a.txt:");

printf("\n");

execl("/usr/bin/wc", "wc", "-c", 0);

exit(0);

}

else

if (f3 == -1)

printf("\nERROR!");

else

printf("\nI'm not in f3!");

//Ожидание окончания работы потомков

printf("\nEnd2 End3 Завершение выполнение потомков tr -d ''[p-z]'' < a.txt | wc -c и tr -d ''[p-z]'' < a.txt | wc -c <a.txt");

wait(&status);

if (status)

return status;

wait(&status1);

if (status1)

return status1;

close(pd[0]);

close(pd[1]);

printf("\nTHE END Завершение родительского процесса\n");

return status;

}

1. ***Тесты***

*a.txt:* a b c d e f g h i j k l m n o p q r s t u v w x y z

Run1 Порождение процесса tr -d ''[p-z]'' < a.txt

I'm not in f1!

Run1 Порождение процесса tr -d ''[p-z]'' < a.txt

I'm in f1

End1 Завершение выполнение потомка tr -d ''[p-z]'' < a.txt

I'm not in f2!

I'm not in f3!

I'm not in f2!

I'm in f3

Run3 Результат выполнения tr -d ''[p-z]'' < a.txt | wc -c <a.txt:

51

End1 Завершение выполнение потомка tr -d ''[p-z]'' < a.txt

I'm in f2

Run2 Результат выполнения tr -d ''[p-z]'' < a.txt | wc -c:

40

End2 End3 Завершение выполнение потомков tr -d ''[p-z]'' < a.txt | wc -c и tr -d ''[p-z]'' < a.txt | wc -c <a.txt

THE END Завершение родительского процесса