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: Косогоров Владислав Валерьевич
Группа: М8О-206Б-18
Вариант: 5
Преподаватель: Соколов Андрей Алексеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2019

Содержание

Постановка задачи.

На вход программе подаётся команда интерпретатора команд. Программа должна произвести вывод команды в верхнем регистре.

Общие сведения о программе

Программа компилируется из файла main.c. В программе используются следующие системные вызовы:

1. **read** – для чтения данных из pipe.
2. **write** – для записи данных в pipe.
3. **pipe** – для создания однонаправленного канала, через который из дочернего процесса данные передаются родительскому. Возвращает два дескриптора файлов: для чтения и для записи.
4. **fork** – для создания дочернего процесса.
5. **close** – для закрытия pipe после окончания считывания результата выполнения команды.

Общий метод и алгоритм решения.

С помощью fork() создаётся дочерний процесс. Для обмена данными между дочерним и родительским процессами создаём pipe. Если pid равен нулю, то идёт выполнение дочернего процесса. В неё мы используем dup2(), где перенаправляем стандартный поток вывода в pipe. Далее с помощью execvp() выполняется команда интерпретатора команд, а результат её работы попадает в pipe. Родительский процесс использует waitpid(), чтобы начать своё выполнение после дочернего. Далее из pipe() по одному перебираются все символы. Если это буква в нижзем регистре, то она переводится в верхний регистр. После этого результат работы выводится в стандартный поток вывода.

Файлы программы.

main.c

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

int main(int argc, char** argv) {
    pid_t pid;
    int rv;
    int fd[2];

    if (pipe(fd) == -1) {
        perror("pipe error");
        exit(1);
```

```
}

if ((pid = fork()) < 0) {
    perror("fork error");
    exit(1);

} else if (pid == 0) {              // потомок
    close(fd[0]);                   // потомок не читает
    dup2(fd[1], STDOUT_FILENO);     // перенаправление stdout
    rv = execvp(argv[1], argv + 1);
    if (rv) {
        perror("exec error");
    }
    _exit(rv);

} else {                            // родитель
    waitpid(pid, &rv, 0);

    close(fd[1]);                   // родитель не записывает
    char buf[1];

    while(read(fd[0], buf, 1) > 0) {
      if (islower(buf[0])) {
          buf[0] = toupper(buf[0]); //замена строчных букв на заглавные
      }
       write(1, buf, 1);            // в stdout
    }
    exit(WEXITSTATUS(rv));
```

```
    }
}
```

Демонстрация работы программы.

```
…/prog_3_sem/os/lab2 ż ./a.out cat main.c | head -4
#INCLUDE <SYS/TYPES.H>
#INCLUDE <UNISTD.H>
#INCLUDE <STDIO.H>
#INCLUDE <SYS/WAIT.H>

…/prog_3_sem/os/lab2 ż ./a.out echo heyyyeyeyeyeaa
HEYYYEYEYEYEAA

…/prog_3_sem/os/lab2 ż ./a.out man exec | tail -30 | head -5

       THE BEHAVIOR OF EXECLP() AND EXECVP() WHEN ERRORS OCCUR  WHILE
       ATTEMPTING  TO  EXECUTE THE FILE IS HISTORIC PRACTICE, BUT HAS
       NOT TRADITIONALLY BEEN DOCUMENTED AND IS NOT SPECIFIED BY  THE
       POSIX  STANDARD.  BSD (AND POSSIBLY OTHER SYSTEMS) DO AN AUTO-
```

Вывод

Я научился создавать процессы, используя системный вызов fork(). Также я обрел навыки взаимодействия между процессами с помощью pipe(), получил новые знания о файловых дескрипторах и выполнении команд интерпретатора команд с помощью семейства системных вызовов exec().