

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Тульский государственный университет»

Институт прикладной математики и компьютерных наук
Кафедра вычислительной техники

Утверждено на заседании кафедры
«Вычислительная техника»
«26» января 2022 г., протокол №7

Заведующий кафедрой
_____ А.Н. Ивутин

**МЕТОДИЧЕСКИЕ УКАЗАНИЯ
К ПРАКТИЧЕСКИМ РАБОТАМ**
по дисциплине (модулю)

«Операционные системы семейства Unix»

**основной профессиональной образовательной программы
высшего образования – программы бакалавриата**

по направлению подготовки
09.03.01 «Информатика и вычислительная техника»

с направленностью (профилем)
Электронно-вычислительные машины, комплексы, системы и сети

Форма обучения: очная, заочная

Идентификационный номер образовательной программы: 090301-02-22

Тула 2021

Разработчик(и) методических указаний

Берсенов Г.Б., доцент каф. ВТ, к.т.н.

(ФИО, должность, ученая степень, ученое звание)

(подпись)

Практическая работа № 2.

Разработка программ на языке Си для синтаксического анализа командных строк и выделения из них команд и аргументов

1. Цели работы

Целями практической работы являются:

- закрепить теоретические знания по видам и синтаксису операторов языка Си;
- получить практические навыки создания на языке Си простейшей командной оболочки ядра Linux.

2. Теоретические сведения

В листинге 1 представлена функция `main` программы, которая может быть использована в качестве простейшей оболочки ядра операционной системы Linux.

Листинг 1. (*myshell.c*) Функция `main()` простейшей оболочки ядра

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/ipc.h>
#include <sys/types.h>
#include <sys/time.h>
#include <time.h>
#include <errno.h>
#include <fcntl.h>
#include <limits.h>
#include <signal.h>
#include <sys/wait.h>
#include <sys/param.h>
#include <sys/resource.h>
#include <syslog.h>
#include <semaphore.h>
#include <sys/mman.h>

// Размерности массивов
#define MAXLINE      4096
#define MAXBUFF      256
#define MAXQUEUE     10
#define MAXPATH      120
#define MAXLEN       256

/*Разрешения по умолчанию для вновь создаваемых файлов*/
#define FILE_MODE    (S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH | O_NDELAY)

char* arg[MAXLEN];      // массив аргументов команды
char* s_path[MAXPATH];  // массив команд с путями из PATH
char* queue[MAXQUEUE];  // массив для хранения неинтерактивных команд
char* queue0[MAXQUEUE]; // массив для хранения интерактивных команд
int errno;              // код ошибки
int outfl;              // флаг вида перенаправления вывода (замещение/дополнение)

void preparse(char* cmd);
int str_parse(char *cmd);
int input(char *name);
int output(char *name);
void cmdExec(char **env);
int execat(char *search_path, char *path);

int main(int argc, char **argv, char **env)
{
    int in_out, p;
    char cmd[MAXLEN]; //строка, считанная с терминала
    int fd_1, fd_0;   //для сохранения дескрипторов потоков стандартного ввода/вывода
```

```

setbuf(stdout, NULL);      //отмена буфферизации

for(;;) {
    write(1, "$> ", 3); // вывод приглашения к вводу командной строки
    gets(cmd);          // чтение командной строки в массив cmd

    preparse(cmd);       // разбиение командной строки cmd на отдельные команды

    p=0;
    // цикл по интерактивным командам
    while (queue0[p]!=NULL) {
        //сохранение текущих дескрипторов потоков стандартного ввода/вывода
        fd_1=dup(1); fd_0=dup(0);
        // запись очередной команды в массив cmd
        snprintf(cmd, sizeof(cmd), "%s", queue0[p]);
        in_out = str_parse(cmd);      //разбор строки команды
        p++;

        // Выполнение встроенных в оболочку команд
        if (!strcmp(arg[0], "showenv")) {
            int i;
            printf("\nEnvironment:\n");
            for (i=0; env[i]; i++)
                printf("\n%s", env[i]);
            printf("\n");
        }
        if (!strcmp(arg[0], "echo"))
            printf("\n%s\n", arg[1]);

        // Выполнение команд - исполнимых файлов
        cmdExec(env);
        //восстановление дескрипторов stdout/stdin
        dup2(fd_1, 1);
        close(fd_1);
        dup2(fd_0, 0);
        close(fd_0);
    }
}

```

Функция main организует цикл для ввода и выполнения командных строк. Для каждой командной строки выдается промптер (приглашение к вводу), состоящее из трех символов. Пример командной строки:

```
date;pwd;ls;ls -al > fl&;echo "Метаданные" >> fl&;/bin/ls -l add.c >> fl&;cat add.c
```

Данная командная строка содержит четыре интерактивных и три неинтерактивных команды. После чтения и записи этой строки в массив cmd функции main, содержимое этого массива передается (копируется через аргумент функции) в массив cmd другой функции - функции preparse.

Разбиение командной строки на отдельные команды

Первый этап обработки командной строки, который выполняется функцией preparse (листинг 2), - это разбиение командной строки на отдельные команды и запись этих команд в массивы интерактивных (массив queue0) и неинтерактивных (массив queue) команд. Для приведенного выше примера командной строки будут сформированы следующие два массива:

Массив queue0	Массив queue
date	ls -al > fl&
pwd	echo "Метаданные" >> fl&
ls	/bin/ls -l a.c >> fl&
cat add.c	

Листинг 2. (myshell.c) Функция preparse() простейшей оболочки ядра

```

void preparse(char* cmd)
{
    int i=0, j=0, k=0, cntr=0, cntr0=0;

```

```

char tmp[80];
char* ptmp = tmp;
for (k=0; k<=79; k++) tmp[k]='\0'; // очистка массива tmp

// Очистка массивов команд
for (i=0; i<MAXQUEUE; i++) free(queue0[i]), queue0[i]=NULL; // очистка массива
queue0
for (i=0; i<MAXQUEUE; i++) free(queue[i]), queue[i] = NULL; // очистка массива
queue
i=0;
strcat(cmd, "");
// разбиение командной строки cmd на отдельные команды
do {
    if ((* (cmd+i)!=';')&&(* (cmd+i)!='\0')) {
        *(tmp+j)=*(cmd+i); j++; //запись в tmp очередного символа
    }
    else { // сохранить очередную команду
        if (strchr(ptmp, '&')) {
            queue[cntr]=strdup(tmp); // сохранение неинтерактивной команды
            cntr++;
        }
        else { // сохранение интерактивной команды
            queue0[cntr0]=strdup(tmp);
            cntr0++;
        }
        k=0; for(k=0; k<=79; k++) { tmp[k]='\0'; } // очистка массива tmp
        j = 0;
    }
    i++;
}
while(* (cmd+i)!='\0');
return;
}

```

При выполнении функции `prparse` массив `tmp` используется как буфер, в который посимвольно переписывается очередная команда. Символ `;` является символом разделения команд и, соответственно, при его обнаружении завершится цикл записи символов в массив `tmp`:

```

if ((* (cmd+i)!=';')&&(* (cmd+i)!='\0')) {
    *(tmp+j)=*(cmd+i); j++; //запись в tmp очередного символа
}

```

В данном фрагменте используются указатели и адресная арифметика языка Си. Этот фрагмент можно переписать с использованием массивов с индексами следующим образом:

```

if ((cmd[i])!=';')&&(cmd[i]!='\0')) {
    tmp[j]= cmd[i]; j++; //запись в tmp очередного символа
}

```

Цикл `do ..... while (* (cmd+i)!='\0');` завершится, когда будет обнаружен символ конца командной строки (символ `\0`).

Отметим, что для сохранения очередной команды функция `strdup` дублирует строку, на которую указывает аргумент. Память под дубликат строки выделяется из кучи с помощью функции `malloc` и по окончании работы с дубликатом должна быть очищена с помощью функции `free`.

Синтаксический анализ строк каждой команды

Данная версия оболочки ядра предназначена для выполнения только интерактивных команд. Другими словами, оболочка не будет выполнять команды из массива `queue`. Чтобы усложнить синтаксический анализ команд сделаем все команды примера командной строки интерактивными:

```
date;pwd;ls;ls -al > f1;echo "Метаданные" >> f1; /bin/ls -l add.c >> f1;cat add.c
```

После выполнения функции `prparse` эта строка будет преобразована в следующие массивы:

Массив <code>queue0</code>	Массив <code>queue</code>
<code>date</code>	
<code>pwd</code>	
<code>ls</code>	
<code>ls -al > f1</code>	
<code>echo "Метаданные" >> f1</code>	

/bin/ls -l add.c >> f1	
cat add.c	

В функции main организован цикл while (queue0[p]!=NULL) { }, в котором каждая очередная интерактивная команда вначале переписывается в массив cmd, который затем передается в функцию str_parse (листинг 3) для разбиения ее на строки аргументов и записи их в массив arg. Отметим, что в строке arg[0] будет находиться имя команды, включая путь или без него.

Пример массива arg, полученного для предпоследней команды из массива queue0:

Массив arg	Элементы массива arg
/bin/ls	arg[0]
-l	arg[1]
add.c	arg[2]
>>	arg[3]
f1	arg[4]

Листинг 3. (myshell.c) Функция str_parse () простейшей оболочки ядра

```

/*Разбор строки команды. Символы перенаправления ввода/вывода воспринимаются как
отдельные аргументы - пишутся через пробел. */
int str_parse(char *cmd)
{
    char tmp[80];
    int i=0,j=0,k,CNTR_arg=0;
    char *name;
    int in_out = -1;          //0 - поток перенаправлен
    // Очистка массива-буфера для очередного аргумента
    k=0; for (k=0; k<=79; k++) tmp[k]='\0'; // очистка массива tmp
    // Очистка массива аргументов
    for (i=0; i<MAXLEN; i++) free(arg[i]), arg[i] = NULL; // очистка массива arg

    strcat(cmd, " "); //добавление пробела в конце команды
    i=0;
    // разбиение команды на аргументы
    do {
        if ((* (cmd+i)!=' ')&&(* (cmd+i)!='\0')) {
            *(tmp+j)=*(cmd+i); j++;          //запись в tmp очередного символа
        }
        else { // сохраняем очередной аргумент
            arg[CNTR_arg]=strdup(tmp);
            CNTR_arg++;
            for(k=0; k<=79; k++) { tmp[k]='\0'; } // очистка массива tmp
            j = 0;
        }
        i++;
    }
    while(* (cmd+i)!='\0');

    // Поиск символов перенаправления, выборка имени файла и
    // модификация массива аргументов
    for (i=0; arg[i] != NULL; i++) {
        if ((strcmp(arg[i], ">")==0) {
            arg[i]=NULL; name = arg[i+1];
            arg[i+1]=NULL; in_out = 0; outfl=0; output(name); break;}
        else if ((strcmp(arg[i], ">>")==0) {
            arg[i]=NULL;name = arg[i+1];
            arg[i+1]=NULL; in_out = 0; outfl=1; output(name); break;}
        }
    }
    for (i=0; arg[i] != NULL; i++) {
        if ((strcmp(arg[i], "<")==0) {
            arg[i]=NULL; name = arg[i+1];
            arg[i+1]=NULL; in_out = 0; outfl=0; input(name); break;}
        }
    }

    return in_out;
}

```

При выполнении функции `str_parse` массив `tmp` используется как буфер, в который посимвольно переписывается очередной аргумент команды:

```
if ((* (cmd+i) != ' ') && (* (cmd+i) != '\0')) {
    * (tmp+j) = * (cmd+i); j++;           //запись в tmp очередного символа
}
```

Символы каждой опции считываются из `cmd` в `tmp`, а затем в массив `arg`. Символ пробела является символом разделения аргументов команды и, соответственно, при его обнаружении в массиве `cmd` завершится очередной цикл записи символов в массив `tmp` и будет выполнена запись массива `tmp` в очередной элемент массива `arg`.

Цикл `do ..... while (* (cmd+i) != '\0');` завершится, когда будет обнаружен символ конца строки команды (символ `\0`).

После завершения разбора строки команды и формирования массива `arg` производится просмотр аргументов команды для поиска в них символов перенаправления ввода-вывода. Тип найденного перенаправления сохраняется в переменной `outfl` (0 для перенаправления выходного потока, 1 - для входного потока), а имя файла потока переписывается в переменную `name`, значение которой передается в функции `input` или `output` (листинг 4) для соответствующей замены файлов стандартного потока ввода или вывода. После этого аргументы, содержащие символы перенаправления и имя потока, очищаются. Кроме этого, записывается 0 в переменную `in_out`, значение которой будет значением, возвращаемым функцией `str_parse`. Отметим, что при отсутствии перенаправления ввода-вывода в команде в переменной `in_out` будет значение -1.

Листинг 4. (myshell.c) Функции `input ()` и `output ()` простейшей оболочки ядра

```
//Перенаправление ввода
int input(char *name)
{
    int fd;
    if ((fd = open(name, O_RDONLY)) < 0) return 0;
    dup2(fd, 0); close(fd); return 1;
}

//Перенаправление вывода
int output(char *name)
{
    int fd;
    if ((fd = open(name, O_WRONLY)) < 0) {
        if (errno == ENOENT) //если файла нет, создадим
            if ((fd = creat(name, 0644)) < 0) return 0; // 0 - не успешно
        if (errno == EACCES) { //права доступа?
            write(2, "Shell: Доступ запрещен\n", 25); return 0; }
        //не имеем права писать в файл
    }
    // ">> file" указатель на конец файла
    else if (outfl == 1) lseek(fd, 1, SEEK_END);

    if (outfl == 0) {
        // "> file" пытаемся опустошить файл
        if ((fd = creat(name, 0644)) < 0) return 0; }
    //перенаправляем вывод
    dup2(fd, 1);
    close(fd); return 1; //1 = успешно
}
}
```

Функция `input` заменяет стандартный поток ввода (поток `stdin`) с файловым дескриптором 0 на поток из файла, имя которого сохранено в переменной `name`.

Функция `output` заменяет стандартный поток вывода (поток `stdout`) с файловым дескриптором 1 на поток в файл, имя которого сохранено в переменной `name`.

Контрольный пример для проверки работы оболочки

На рис. 1 приведен контрольный пример для проверки работы разработанной оболочки ядра операционной системы Linux.

```

user1@user1-VirtualBox:~/curs/curs01$ gcc myshell_01.c
myshell_01.c: In function 'main':
myshell_01.c:54:6: warning: implicit declaration of function 'gets'; did you mean 'fgets'? [-Wimplicit-function-declaration]
  54 |     gets(cmd); // чтение командной строки в массив cmd
      |     ^~~~~
      |     fgets
/usr/bin/ld: /tmp/ccYYifvJ.o: в функции «main»:
myshell_01.c:(.text+0x6c): предупреждение: the 'gets' function is dangerous and should not be used.
user1@user1-VirtualBox:~/curs/curs01$ ./a.out
$> date;pwd;ls;ls -al > f1;echo "Метаданные" >> f1;ls -l add.c >> f1;cat add.c
Пт 01 окт 2021 16:44:22 MSK
/home/user1/curs/curs01
add.c add.cpp a.out f1 hello.c myshell_01.c myshell_1.c myshell.c
// add.c
#include <stdio.h>
int main ()
{ int x, y;
  printf("Input 2 numburs: ");
  scanf ("%d%d", &x, &y);
  printf("Summa = %d\n", x+y);
}

$> cat f1
итого 80
drwxrwxr-x 2 user1 user1 4096 окт 1 16:43 .
drwxrwxr-x 8 user1 user1 4096 окт 1 10:42 ..
-rw-r--r-- 1 user1 user1 147 авг 21 2020 add.c
-rw-r--r-- 1 user1 user1 154 авг 21 2020 add.cpp
-rwxrwxr-x 1 user1 user1 22200 окт 1 16:41 a.out
-rw-rw-r-- 1 user1 user1 0 окт 1 16:44 f1
-rw-r--r-- 1 user1 user1 89 авг 20 2020 hello.c
-rw-rw-rw- 1 user1 user1 9320 окт 1 16:41 myshell_01.c
-rw-rw-rw- 1 user1 user1 9037 окт 1 14:37 myshell_1.c
-rw-rw-rw- 1 user1 user1 8348 окт 1 10:45 myshell.c
"Метаданные"
-rw-r--r-- 1 user1 user1 147 авг 21 2020 add.c
$>

```

Рис. 1. Результаты компиляции и выполнения оболочки ядра

3. Задание на работу

3.1. Ознакомиться с исходным кодом программы простейшей оболочки ядра операционной системы Linux. Откомпилировать и выполнить отладку программы.

3.2. Добавить неименованный канал между родительским и дочерним процессами для передачи команд из массива `queue0` для исполнения дочерним процессом. Откомпилировать и выполнить отладку программы.

5. Оформление отчета

Отчет должен содержать тексты программ, командные строки и результаты их выполнения.

6. Контрольные вопросы

- 6.1. Какие объекты IPC используются в различных системах?
- 6.2. Какие команды оболочки shell используются для работы с файловой системой?
- 6.3. Приведите примеры использования вызова `dup2()`.
- 6.4. Напишите программу реализации программного канала между двумя дочерними процессами.
- 6.5. Какие команды оболочки shell выполняются без создания процесса-потомка? Почему?
- 6.6. Приведите прототипы функций `sprintf()` и `snprintf()` и укажите их назначение.
- 6.7. Как разбить командную строку на отдельные команды?
- 6.8. Как производится разбиение команды на строки аргументов?

- 6.9. Какие дополнительные переменные используются для сохранения информации о наличии символов перенаправления ввода-вывода?
- 6.10. Как можно создать массив команд на базе путей из PATH?

Библиографический список

1. Робачевский, А.М. Операционная система UNIX : учеб.пособие / А.М.Робачевский, С.А.Немнюгин, О.Л.Стесик .— 2-е изд.,перераб.и доп. — СПб. : БХВ-Петербург, 2005 .— 635с.
2. Магда, Ю.С. UNIX для студента / Ю.С.Магда .— СПб. : БХВ-Петербург, 2007 .— 480с.
3. Рочкинд, М.Д. Программирование для UNIX / М.Д.Рочкинд;пер.с англ.под общ.ред.В.В.Вшивцева .— 2-е изд. — СПб. : БХВ-Петербург, 2005 .— 704с.
4. Моли, Molay B. Unix/Linux:теория и практика программирования : пер.с англ. / Б.Моли .— М. : КУДИЦ-ОБРАЗ, 2004 .— 576с.
5. Галатенко, В.А. Программирование в стандарте POSIX : Курс лекций:Учеб.пособие / В.А.Галатенко; Под ред.В.Б.Бетелина .— М. : Интернет-Университет Информационных Технологий, 2004 .— 560с.
6. Хорвиц, Д. Unix-системы.От проектирования до сопровождения : пер.с англ. / Д.Хорвиц .— М. : DiaSoft, 2004 .— 608с.
7. Реймонд, Э.С. Искусство программирования для Unix / Э.С.Реймонд;пер.с англ.и ред.В.А.Швеца .— М.и др. : Вильямс, 2005 .— 543с.
8. Стивенс У. UNIX: взаимодействие процессов. – СПб. : Питер, 2002. – 576 с.
9. Митчелл М., Оулдем Д., Самьюэл А. Программирование для Linux. Профессиональный подход. - М.: "Вильямс", 2002. - 288 с.