**Язык программирования C++**

*Отчет*

|  |  |
| --- | --- |
| Лабораторная работа 2 | *12.10.2019* |

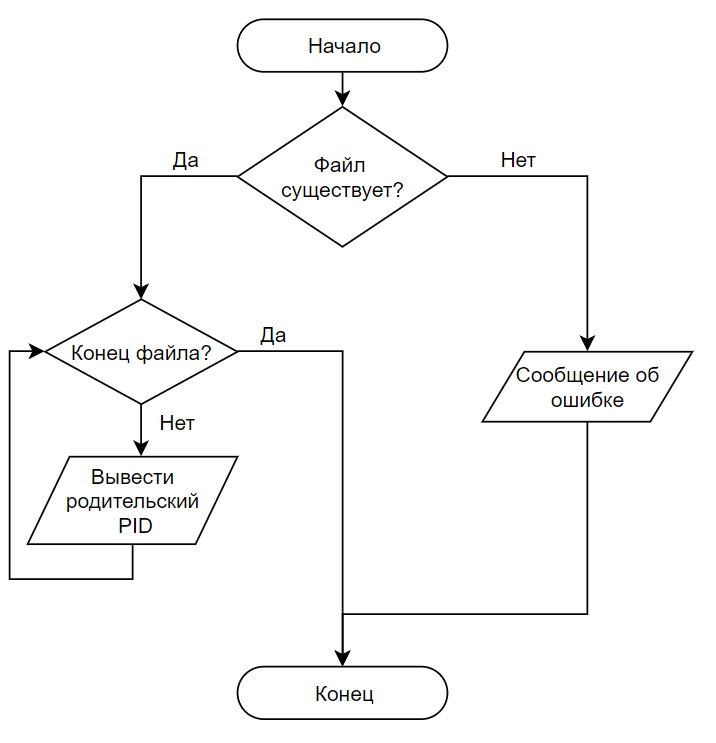
Основы программирования для Linux

*Задача №1*

**1. Условия задачи**

*Раздел №3.5.1 из курса“Основы программирования для Linux”.* Разработать программу solution, которая осуществляет поиск родительского PID текущего процесса в файловой системе proc и выводит найденное значение на консоль. Результат, возвращаемый функций должен быть выведен отдельной строкой (должен оканчиваться символом перевода строки \n) в stdio.

**2. Алгоритм. Блок-схема**

****

**3. Исходный код программы**

#include <stdio.h>

#include <stdlib.h>

#include <errno.h>

#include <string.h>

#include <sys/procfs.h>

#include <sys/types.h>

#include <ctype.h>

int main()

{

FILE \*fp;

char line[100];

char \*p;

if ( (fp = fopen("/proc/self/status", "r")) == NULL) {

fprintf(stderr, "open status file error: %s\n", strerror(errno));

}

while(fgets(line, 100, fp)) {

if(strncmp(line, "PPid:", 5) != 0)

continue;

p = line + 6;

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

printf("%s", p);

break;

}

fclose(fp);

return 0;

}

**4. Формат входных и выходных данных**

Формат входных данных

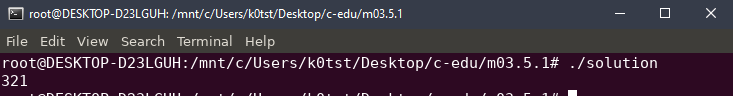
Программа ничего не принимает на входе.

Формат выходных данных

Программа выводит в стандартный поток вывода число в отдельную строку (вывод должен оканчиваться символом перевода строки \n).

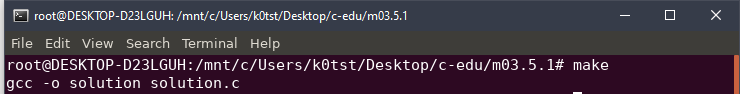
**5. Результат работы программы**





**6. Выводы и комментарии к решению задачи**

* Библиотека *sys/procfs.h* системная и используется для работы с процессами;
* Библиотека *sys/types.h* системная и используется для работы с системными типами данных;

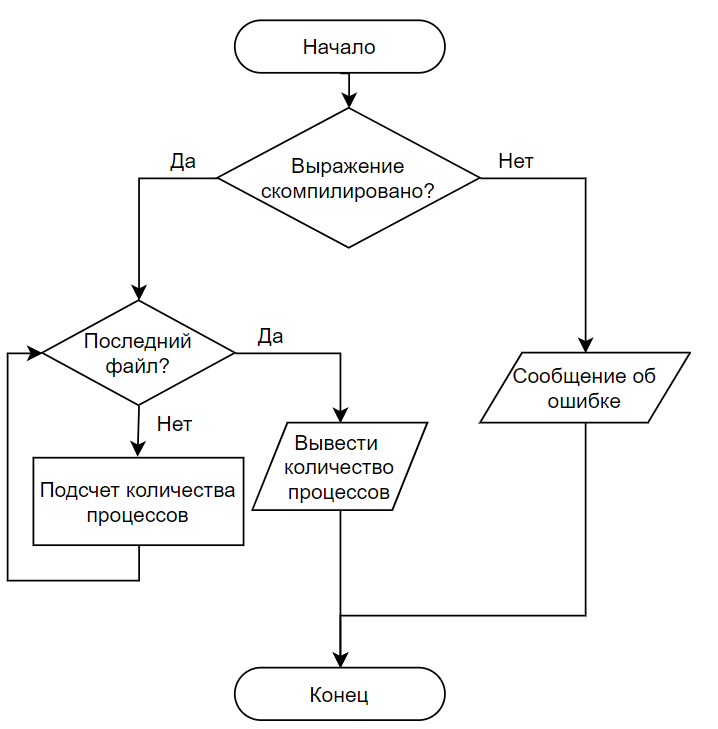


*Задача №2*

**1. Условия задачи**

*Раздел №3.5.2 из курса “Основы программирования для Linux”.* Разработать программу solution, которая осуществляет поиск и подсчет числа процессов с именем genenv в системе Linux на которой выполняется программа.

**2. Алгоритм. Блок-схема**

****

**3. Исходный код программы**

#include <stdio.h>

#include <stdlib.h>

#include <dirent.h>

#include <regex.h>

#include <sys/types.h>

#include <string.h>

#include <errno.h>

#define \_REGEX\_RE\_COMP

int main()

{

DIR \*d;

FILE \*f;

struct dirent \*e;

regex\_t regex;

char fstat[255];

int counter = 0;

int pid, ppid;

char comm[255], state;

if (regcomp(&regex, "^[[:digit:]]", 0) != 0) {

fprintf(stderr, "Could not compile regex: %s\n", strerror(errno));

exit(1);

}

d = opendir("/proc");

while ((e = readdir(d)) != NULL) {

/\* printf("%s", e->d\_name); \*/

if (regexec(&regex, e->d\_name, 0, NULL, 0) == 0) {

/\* printf("%s match\n", e->d\_name); \*/

sprintf(fstat, "/proc/%s/stat", e->d\_name);

/\* printf("fstat: %s\n", fstat); \*/

f = fopen(fstat, "r");

fscanf(f,"%d %s %c %d ", &pid, comm, &state, &ppid);

if (strcmp("(genenv)", comm) == 0)

counter = counter + 1;

}

}

closedir(d);

printf("%d\n", counter);

return 0;

}

**4. Формат входных и выходных данных**

Формат входных данных

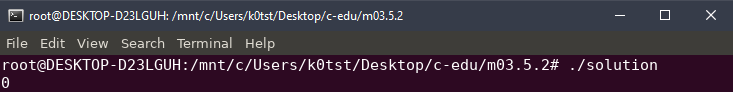
Программа ничего не принимает на входе.

Формат выходных данных

Программа выводит в стандартный поток вывода число, полученное после вызова функции, и символ новой строки \n.

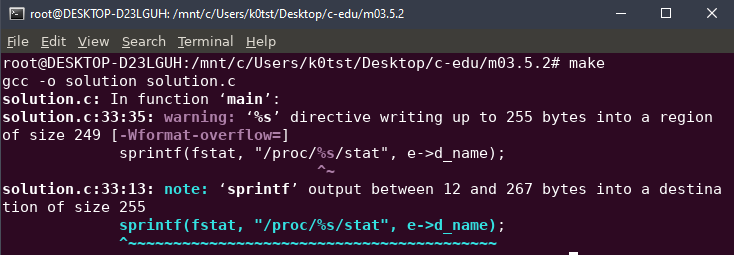
**5. Результат работы программы**





**6. Выводы и комментарии к решению задачи**

* Результаты компиляции и работы программы отличаются от данных с сайта, поскольку для получения результата полученного на сайте, необходимо использовать Docker.



*Задача №3*

**1. Условия задачи**

*Раздел №3.5.3 из курса“Основы программирования для Linux”.* Разработать программу solution, которая по заданному pid, осуществляет поиск пути в дереве процессов до процесса с идентификатором 1 (init). Для каждого найденного процесса печатается в отдельной строке его идентификатор.

**2. Алгоритм. Блок-схема**

**3. Исходный код программы**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

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

{

char fstat[255];

FILE \*f;

int pid, ppid, nowpid;

char comm[255], state;

if (argc != 2) {

printf("Usage: solution PID\n");

exit(1);

}

nowpid = atoi(argv[1]);

printf("%d\n", nowpid);

while (nowpid != 1) {

sprintf(fstat, "/proc/%d/stat", nowpid);

f = fopen(fstat, "r");

fscanf(f, "%d %s %c %d ", &pid, comm, &state, &ppid);

printf("%d\n", ppid);

nowpid = ppid;

}

return 0;

}

**4. Формат входных и выходных данных**

Формат входных данных

Программа принимает на входе целое число.

Формат выходных данных

Программа выводит в стандартный поток вывода числа (каждое число в отдельной строке).

**5. Результат работы программы**



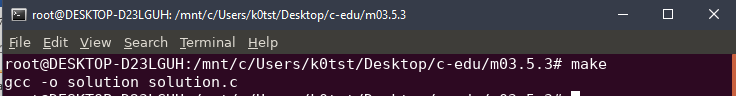






**6. Выводы и комментарии к решению задачи**

* результаты работы программы отличаются от данных с сайта, поскольку для получения результата полученного на сайте, необходимо использовать Docker,
* результатом работы программы на компьютере стала ответ о том что ядро было сброшено.

**

*Задача №4*

**1. Условия задачи**

*Раздел №3.5.4 из курса“Основы программирования для Linux”.* Разработать программу solution, которая по заданному pid, осуществляет поиск всех потомков процесса с этим идентификатором и выводит их количество (включая процесс с заданным pid).

**2. Алгоритм. Блок-схема**

**3. Исходный код программы**

#include <stdio.h>

#include <stdlib.h>

#include <dirent.h>

#include <regex.h>

#include <sys/types.h>

#include <string.h>

#include <errno.h>

#define \_REGEX\_RE\_COMP

void find\_in\_dir(int proc\_pid);

DIR \*d;

FILE \*f;

struct dirent \*e;

regex\_t regex;

char fstat[255];

int counter = 1;

int pid, ppid, arg;

char comm[255], state;

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

{

if (argc != 2) {

printf("Usage: solution PID\n");

exit(1);

}

if (regcomp(&regex, "^[[:digit:]]", 0) != 0) {

fprintf(stderr, "Could not compile regex: %s\n", strerror(errno));

exit(1);

}

arg = atoi(argv[1]);

find\_in\_dir(arg);

printf("%d\n", counter);

return 0;

}

/\* Find process with PPID == proc\_pid

\* and all him child use recursion.

\*/

void find\_in\_dir(int proc\_pid)

{

DIR \*d;

/\* open proc \*/

d = opendir("/proc");

/\* read direnr \*/

while ((e = readdir(d)) != NULL) {

/\* Check what d\_name match pattern \*/

if (regexec(&regex, e->d\_name, 0, NULL, 0) == 0) {

/\* find\_in\_dir(atoi(e->d\_name)); \*/

sprintf(fstat, "/proc/%s/stat", e->d\_name);

f = fopen(fstat, "r");

fscanf(f,"%d %s %c %d ", &pid, comm, &state, &ppid);

fclose(f);

if (proc\_pid == ppid) {

counter++; /\* increase counter \*/

find\_in\_dir(pid); /\* recursion \*/

}

}

}

closedir(d);

}

**4. Формат входных и выходных данных**

Формат входных данных

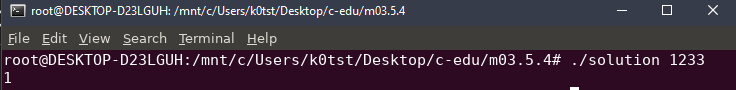
Программа принимает на входе целое число, номер PID.

Формат выходных данных

Программа выводит в стандартный поток вывода число в отдельной строке( число должно оканчиваться символом перевода строки \n).

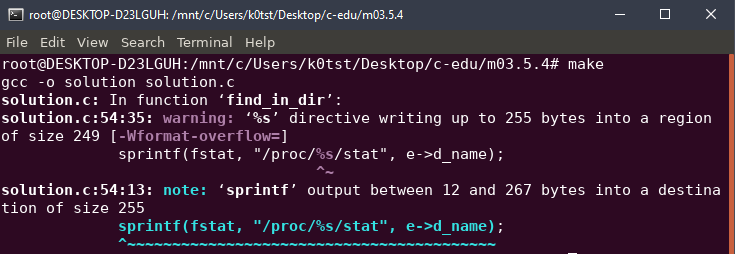
**5. Результат работы программы**





**6. Выводы и комментарии к решению задачи**

* Результаты компиляции и работы программы отличаются от данных с сайта, поскольку для получения результата полученного на сайте, необходимо использовать Docker.



*Задача №5*

**1. Условия задачи**

*Раздел №3.5.5 из курса“Основы программирования для Linux”.* Разработать программу solution, которая при запуске себя "демонизирует" и остается в памяти. Перед закрытием стандартного потока вывода stdout унаследованного от родителя, программа должна вывести в него Pid процесса-демона..

**2. Алгоритм. Блок-схема**

**3. Исходный код программы**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

int main()

{

pid\_t pid;

pid = fork();

if (!pid) {

chdir("/");

setsid();

close(STDIN\_FILENO);

close(STDOUT\_FILENO);

close(STDERR\_FILENO);

while(1) {

sleep(1);

}

}

else {

printf("%d\n", pid);

return 0;

}

}

**4. Формат входных и выходных данных**

Формат входных данных

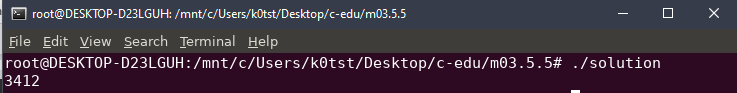
Программа не принимает на входе ни каких аргументов.

Формат выходных данных

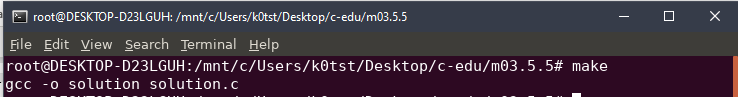
Программа выводит в стандартный поток вывода число в отдельной строке (число должно оканчиваться символом перевода строки \n) перед его закрытием.

**5. Результат работы программы**





**6. Выводы и комментарии к решению задачи**



*Выводы по лабораторной работе 3*

**1.** Решены 5 задач по теме «*Основы программирования для Linux*»:

1) Поиск ppid c использованием procfs,

2) Подсчет числа процессов с заданным именем,

3) Поиск пути к init по заданному pid,

4) Подсчет процессов в поддереве,

5) Создание демона.

СПИСОК ИСТОЧНИКОВ И ЛИТЕРАТУРЫ

1. Основы программирования для Linux. – Режим доступа: <https://stepik.org/lesson/26302/step/1?unit=8180>, свободный. Загл. с экрана. – Дата обращения: 10.09.2019