**Язык программирования C++**

*Отчет*

|  |  |
| --- | --- |
| Лабораторная работа 3 | *19.10.2019* |

Основы программирования для Linux

*Задача №1*

**1. Условия задачи**

*Раздел №4.6.1 из курса“Основы программирования для Linux”.* Некоторая утилита генерирует довольно большой вывод, а вам требуется всего-лишь посчитать количество символов '0' в этом выводе. Утилита при запуске принимает 1 параметр. Вам требуется разработать программу, которая вызывает указанную утилиту, с заданным параметром и подсчитывает количество символов '0' в ее выводе. Ваша программа принимает на вход 2 параметра -- имя утилиты, в текущем каталоге и ее параметр. Ваша программа должна после подсчета вывести найденное число '0' в отдельной строке, заканчивающейся символом конца строки.

**2. Алгоритм. Блок-схема**

**3. Исходный код программы**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <errno.h>

#include <fcntl.h>

#include <sys/stat.h>

#include <sys/wait.h>

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

{

if (argc != 3) {

fprintf(stderr, "Usage: %s programm arg\n", argv[0]);

exit(1);

}

pid\_t pid;

int status;

int pipefd[2];

int counter = 0;

pipe(pipefd);

if ((pid = fork()) < 0) {

fprintf(stderr, "Can't fork: %s\n", strerror(errno));

exit(1);

} else if (pid == 0) {

/\* child here \*/

close(pipefd[0]);

dup2(pipefd[1], 1); // send stdout to the pipe

close(pipefd[1]); // this descriptor is no longer needed

execl(argv[1], argv[1], argv[2], NULL);

} else {

close(pipefd[1]); // close the write end of the pipe in the paren

char buffer[4096];

int bytes;

while ((bytes = read(pipefd[0], buffer, sizeof(buffer))) != 0) {

/\* printf("read %d bytes\n", bytes); \*/

int i = 0;

for (i = 0; i < bytes; i++) {

if (buffer[i] == '0')

counter = counter + 1;

}

}

/\* parrent \*/

if (waitpid (pid, &status, 0) != pid) {

fprintf(stderr, "waitpid error: %s\n", strerror(errno));

exit(1);

}

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

}

return 0;

}

**4. Формат входных и выходных данных**

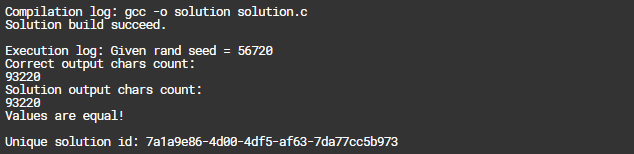
Формат входных данных

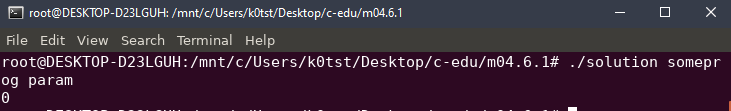
Программа принимает на входе строку.

Формат выходных данных

Программа выводит в стандартный поток вывода число в отдельную строку.

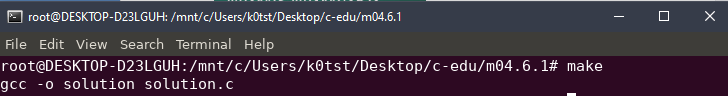
**5. Результат работы программы**





**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 №2*

**1. Условия задачи**

*Раздел №4.6.2 из курса “Основы программирования для Linux”.* В текущем каталоге есть 2 канала in1 in2, в которые в случайном порядке поступают числа, которые необходимо просуммировать и вывести окончательную сумму на экран. Сумма выводится в отдельной строке, завершающейся символом конца строки. Признаком окончания подачи символов в каналы является закрытие этих каналов посылающей стороной.

**2. Алгоритм. Блок-схема**

**3. Исходный код программы**

#include <fcntl.h>

#include <stdio.h>

#include <sys/stat.h>

#include <unistd.h>

#include <stdlib.h>

#include <errno.h>

#define MAX\_BUF 1024

int main()

{

/\* printf("PID: %d\n", getpid()); \*/

int fd1, fd2, res;

char \* myfifo1 = "./in1";

char \* myfifo2 = "./in2";

char buf[MAX\_BUF];

int result = 0;

fd\_set read\_fd\_set;

fd1 = open(myfifo1, O\_RDONLY);

if (fd1 == -1) {

printf("fd1 == -1\n");

exit(1);

}

fd2 = open(myfifo2, O\_RDONLY);

if (fd2 == -1) {

printf("fd2 == -1\n");

exit(1);

}

for (;;) {

/\* printf("fd1 = %d, fd2 = %d\n", fd1, fd2); \*/

FD\_ZERO(&read\_fd\_set);

FD\_SET(fd1, &read\_fd\_set);

FD\_SET(fd2, &read\_fd\_set);

/\* printf("set done\n"); \*/

select(5, &read\_fd\_set, NULL, NULL, NULL);

if (FD\_ISSET(fd1, &read\_fd\_set)) {

res = read(fd1, &buf, MAX\_BUF);

if (res > 0) {

/\* printf("read %d bytes from in1: %d\n", res, atoi(buf)); \*/

result = result + atoi(buf);

}

if (res == 0) {

/\* printf("in1 res == 0\n"); \*/

break;

}

}

if (FD\_ISSET(fd2, &read\_fd\_set)) {

res = read(fd2, &buf, MAX\_BUF);

if (res > 0) {

/\* printf("read %d bytes from in2: %d\n", res, atoi(buf)); \*/

result = result + atoi(buf);

}

if (res == 0) {

/\* printf("in2 res == 0\n"); \*/

break;

}

}

}

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

close(fd1);

close(fd2);

return 0;

}

**4. Формат входных и выходных данных**

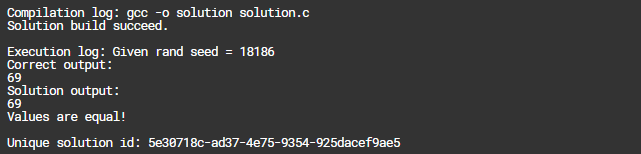
Формат входных данных

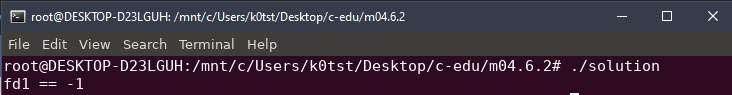
Программа ничего принимает на входе.

Формат выходных данных

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

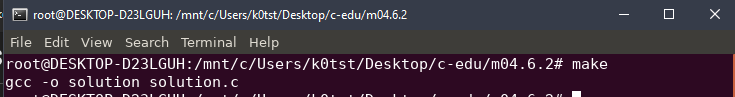
**5. Результат работы программы**





**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 №4.6.3 из курса“Основы программирования для Linux”.* Разработать приложение, умеющее обрабатывать сигналы SIGUSR1 ,SIGUSR2, SIGTERM. После старта Ваше приложение должно по приходу одного из сигналов SIGUSR1, SIGUSR2 выполнять суммирование числа срабатываний каждого из сигналов, а после прихода сигнала SIGTERM, требуется вывести в стандартный поток вывода 2 числа, разделенных пробелом, соответствующих количеству обработанных сигналов SIGUSR1, SIGUSR2, и завершить программу. Вывод оканчивается символом конца строки.

**2. Алгоритм. Блок-схема**

**3. Исходный код программы**

#include <stdio.h>

#include <stdlib.h>

#include <signal.h>

int usr1 = 0;

int usr2 = 0;

void usr1\_handler(int signo)

{

if (signo == SIGUSR1)

usr1 = usr1 + 1;

}

void usr2\_handler(int signo)

{

if (signo == SIGUSR2)

usr2 = usr2 + 1;

}

void term\_handler(int signo)

{

if (signo == SIGTERM) {

printf("%d %d\n", usr1, usr2);

exit(0);

}

}

int main()

{

signal(SIGUSR1, usr1\_handler);

signal(SIGUSR2, usr2\_handler);

signal(SIGTERM, term\_handler);

while(1) {}

return 0;

}

**4. Формат входных и выходных данных**

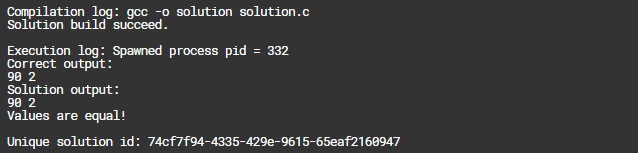
Формат входных данных

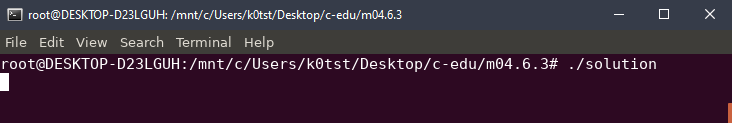
Программа принимает на входе целое число.

Формат выходных данных

Программа выводит в стандартный поток вывода 2 числа, разделенных пробелом (Строка завершается символом конца строки)

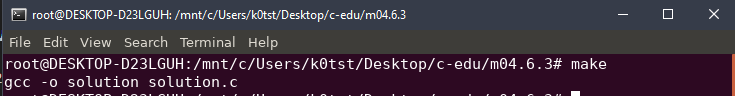
**5. Результат работы программы**





**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 стал бесконечный цикл, без вывода данных.

**

*Задача №4*

**1. Условия задачи**

*Раздел №4.6.4 из курса“Основы программирования для Linux”.* В задании требуется доработать демон, разработанный ранее в задании 6 модуля 3.5. Задача -- снабдить демон обработчиком сигнала SIGURG, по приходу которого демон должен завершать свою работу.

**2. Алгоритм. Блок-схема**

**3. Исходный код программы**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <signal.h>

void sigurg\_handler(int signo)

{

if (signo == SIGURG) {

exit(0);

}

}

int main()

{

pid\_t pid;

pid = fork();

if (!pid) {

chdir("/");

setsid();

close(STDIN\_FILENO);

close(STDOUT\_FILENO);

close(STDERR\_FILENO);

signal(SIGURG, sigurg\_handler);

while(1) {

sleep(1);

}

}

else {

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

return 0;

}

}

**4. Формат входных и выходных данных**

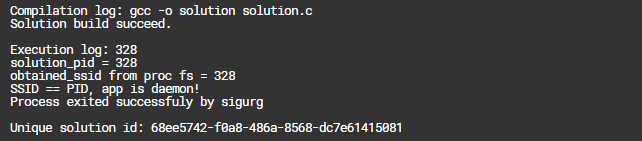
Формат входных данных

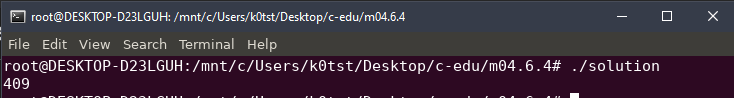
Программа ничего не принимает на входе.

Формат выходных данных

Программа выводит в стандартный поток PID созданного демона. (Строка завершается символом конца строки).

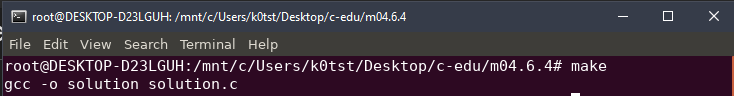
**5. Результат работы программы**





**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 №4.6.5 из курса“Основы программирования для Linux”.* В системе существуют 2 региона разделяемой памяти, заполненной некоторыми числами (типа int). Каждый из регионов имеет размер 1000 байт. Вам требуется разработать приложение, которое попарно суммирует первые 100 чисел в этих регионах и помещает суммы в новый (созданный вашим приложением) регион памяти размером 1000 байт. Таким образом, после завершения работы Вашего приложения в памяти должен существовать регион разделяемой памяти размером 1000 байт, содержащий в начале 100 сумм. Перед завершением работы приложение выводит в стандартный поток ввода-вывода ключ созданного региона, завершающийся символом конца строки. На вход ваше приложение принимает ключи существующих регионов памяти.

**2. Алгоритм. Блок-схема**

**3. Исходный код программы**

#include <stdio.h>

#include <stdlib.h>

#include <sys/shm.h>

#include <sys/ipc.h>

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

{

if (argc != 3) {

exit(1);

}

key\_t key\_one, key\_two;

key\_one = atoi(argv[1]);

key\_two = atoi(argv[2]);

int i = 0;

int shmid\_one, shmid\_two, shmid\_sum;

int \*shared\_one;

int \*shared\_two;

int \*shared\_sum;

shmid\_one = shmget(key\_one, 1000, 0600);

shared\_one = (int \*)shmat(shmid\_one, NULL, 0);

shmid\_two = shmget(key\_two, 1000, 0600);

shared\_two = (int \*)shmat(shmid\_two,NULL,0);

/\* Create key\_t \*/

key\_t sum\_mem\_key = ftok("solution.c", 0);

shmid\_sum = shmget(sum\_mem\_key, 1000, IPC\_CREAT | 0666);

shared\_sum = (int \*)shmat(shmid\_sum, (void \*) 0, 0);

for(i=0; i<100; i++) {

shared\_sum[i] = shared\_one[i] + shared\_two[i];

}

printf("%d\n", sum\_mem\_key);

return 0;

}

**4. Формат входных и выходных данных**

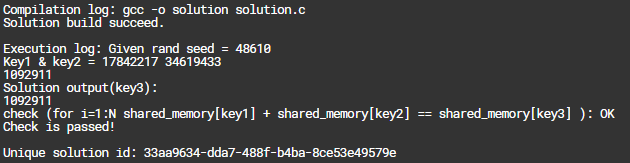
Формат входных данных

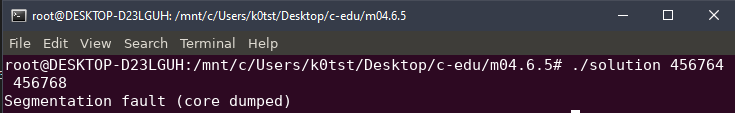
Программа принимает на входе два целочисленных аргумента.

Формат выходных данных

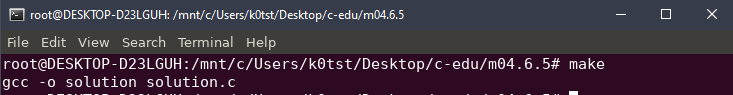
Программа выводит в стандартный поток вывода ключ созданного региона памяти (Строка завершается символом конца строки).

**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) Чтение консольного вывода,

2) Мультиплексированный ввод,

3) Подсчет сигналов,

4) "Правильный" демон,

5) Разделяемая память.

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