

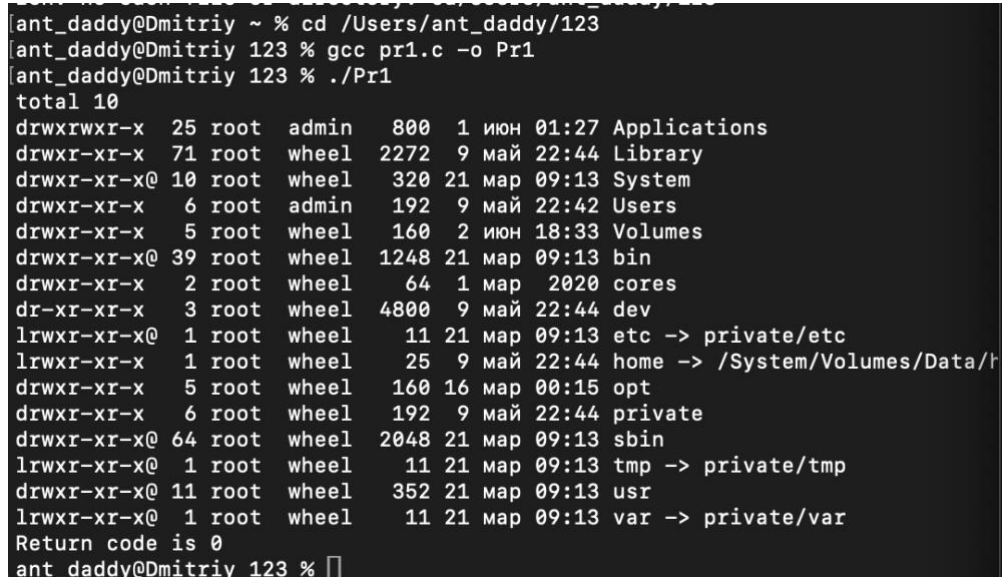
Операционные системы и сети

Лабораторная работа №3 Джугели Дмитрий ПИН-31Д

Задание 1. Выполните программу **pr1.c**, в которой запускается команда "**ls -l /**".

Поясните в отчете код возврата функции **system** при успешном и неуспешном (задайте для **ls** несуществующий каталог) выполнении. Выведите код завершения программы **pr3.c** также и из оболочки, пользуясь соответствующей переменной оболочки (см. лабораторную работу 1). Одинаковы ли эти коды завершения?

```
/* pr1.c */  
  
#include <stdio.h>  
  
#include <stdlib.h>  
  
int main()  
{ int ret_val;  
  
    ret_val = system("ls -l /");  
  
    printf("Return code is %d\n", ret_val);  
  
    return ret_val;  
  
}
```



```
ant_daddy@Dmitriy ~ % cd /Users/ant_daddy/123  
ant_daddy@Dmitriy 123 % gcc pr1.c -o Pr1  
ant_daddy@Dmitriy 123 % ./Pr1  
total 10  
drwxrwxr-x 25 root  admin   800  1 июн  01:27 Applications  
drwxr-xr-x 71 root  wheel  2272  9 май  22:44 Library  
drwxr-xr-x@ 10 root  wheel   320 21 мар  09:13 System  
drwxr-xr-x  6 root  admin   192  9 май  22:42 Users  
drwxr-xr-x  5 root  wheel   160  2 июн  18:33 Volumes  
drwxr-xr-x@ 39 root  wheel  1248 21 мар  09:13 bin  
drwxr-xr-x  2 root  wheel    64  1 мар  2020 cores  
dr-xr-xr-x  3 root  wheel  4800  9 май  22:44 dev  
lrwxr-xr-x@ 1 root  wheel    11 21 мар  09:13 etc -> private/etc  
lrwxr-xr-x  1 root  wheel    25  9 май  22:44 home -> /System/Volumes/Data/t  
drwxr-xr-x  5 root  wheel   160 16 мар  00:15 opt  
drwxr-xr-x  6 root  wheel   192  9 май  22:44 private  
drwxr-xr-x@ 64 root  wheel  2048 21 мар  09:13 sbin  
lrwxr-xr-x@ 1 root  wheel    11 21 мар  09:13 tmp -> private/tmp  
drwxr-xr-x@ 11 root  wheel   352 21 мар  09:13 usr  
lrwxr-xr-x@ 1 root  wheel    11 21 мар  09:13 var -> private/var  
Return code is 0  
ant_daddy@Dmitriy 123 %
```

Return code is 0 при успешном завершении программы

Зададим для **ls** несуществующий путь (`ret_val = system("ls -l /invalid_direct");`)

```

1 error generated.
ant_daddy@Dmitriy 123 % gcc pr1.c -o Pr1_1
ant_daddy@Dmitriy 123 % ./Pr1_1
ls: /invalid_direct: No such file or directory
Return code is 256
ant_daddy@Dmitriy 123 %

```

Return code is 256 – означает, что директория не найдена

```

ant_daddy@Dmitriy 123 % gcc pr3.c -o Pr3
ant_daddy@Dmitriy 123 % ./Pr3
ant_daddy@Dmitriy 123 % total 10
drwxrwxr-x 25 root admin 800 1 июн 01:27 Applications
drwxr-xr-x 71 root wheel 2272 9 май 22:44 Library
drwxr-xr-x@ 10 root wheel 320 21 мар 09:13 System
drwxr-xr-x 6 root admin 192 9 май 22:42 Users
drwxr-xr-x 5 root wheel 160 2 июн 18:33 Volumes
drwxr-xr-x@ 39 root wheel 1248 21 мар 09:13 bin
drwxr-xr-x 2 root wheel 64 1 мар 2020 cores
dr-xr-xr-x 3 root wheel 4800 9 май 22:44 dev
lrwxr-xr-x@ 1 root wheel 11 21 мар 09:13 etc -> private/etc
lrwxr-xr-x 1 root wheel 25 9 май 22:44 home -> /System/Volumes/Data/home
drwxr-xr-x 5 root wheel 160 16 мар 00:15 opt
drwxr-xr-x 6 root wheel 192 9 май 22:44 private
drwxr-xr-x@ 64 root wheel 2048 21 мар 09:13 sbin
lrwxr-xr-x@ 1 root wheel 11 21 мар 09:13 tmp -> private/tmp
drwxr-xr-x@ 11 root wheel 352 21 мар 09:13 usr
lrwxr-xr-x@ 1 root wheel 11 21 мар 09:13 var -> private/var
echo $?
0

```

Return code is 0

Коды завершения одинаковые

Задание 2.

2.1. Выполните программу **pr2.c**. Поясните в отчете, сколько процессов будет выполнено? Сколько сообщений будет напечатано? Нарисуйте дерево процессов. Почему приглашение оболочки **\$** появляется раньше, чем программа завершает работу?

```

/* pr2.c */

#include <stdio.h>

#include <unistd.h>

int main()

{
    fork(); printf("A\n");

    fork(); printf("B\n");

    return 0;

}

```

```

0
ant_daddy@Dmitriy 123 % ./Pr2
A
A
B
B
B
B
ant_daddy@Dmitriy 123 %

```

Будет выполнено 3 процесса.

Будет напечатано 6 сообщений.

Дерево: - Процесс 1

- Процесс 2

- Процесс 3

2.2. В программе **pr2.c** приостановите все процессы, например, при помощи **getchar()**.

```

pr2.c
#include <stdio.h>
#include <unistd.h>
int main() {
    fork(); printf("A\n");
    fork(); printf("B\n");

    getchar();
    return 0;
}

```

Введите команду **ps** со следующими опциями:

\$ps -e -o pid,ppid,start_time,command

```

68453      1 /System/Library/PrivateFrameworks/BridgeOSInstallReporting.framework
41038  1004 login -pf ant_daddy
41039  41038 -zsh
41051  41039 ./Pr2
41052  41051 ./Pr2
41053  41051 ./Pr2
41054  41052 ./Pr2
41063  1004 login -pf ant_daddy
41064  41063 -zsh
41078  41064 -ps -e -o pid,ppid,start_time,command
ant_daddy@Dmitriy ~ %

```

Эта команда выводит PID процессов, их родительские PID, время запуска и команду запуска.

\$ps -f -A

```
ant_daddy — zsh — 80x24
nts/MacOS/AXVisualSupportAgent launchd -s
501 60231 1 0 15май24 ?? 0:10.20 /System/Library/CoreServices/ControlCenter.app/Contents/XPCServices/ControlCenterHelper.xpc/Contents/MacOS/ControlCenterHelper
0 60232 1 0 15май24 ?? 0:00.15 /usr/sbin/BlueTool -R
0 64094 1 0 16май24 ?? 0:00.65 /System/Library/PrivateFrameworks/MobileSoftwareUpdate.framework/Versions/A/XPCServices/com.apple.MobileSoftwareUpdate.CleanupPreparePathService.xpc/Contents/MacOS/com.apple.MobileSoftwareUpdate.CleanupPreparePathService
501 68205 1 0 20май24 ?? 0:00.30 /System/Library/PrivateFrameworks/IASUtilities.framework/Versions/A/Resources/installerauthagent
0 68207 1 0 20май24 ?? 14:57.61 /private/var/db/com.apple.xpc.roleaccountd.staging/exec/16777221.49952073.xpc/Contents/MacOS/com.apple.MobileSoftwareUpdate.UpdateBrainService
0 68208 1 0 20май24 ?? 0:00.21 /usr/libexec/xpcroleaccountd -l
launchd
0 68453 1 0 21май24 ?? 0:00.29 /System/Library/PrivateFrameworks/BridgeOSInstallReporting.framework/Resources/bosreporter
0 41038 1004 0 6:00 ttys000 0:00.03 login -pf ant_daddy
501 41039 41038 0 6:00 ttys000 0:00.08 -zsh
0 41063 1004 0 6:03 ttys001 0:00.03 login -pf ant_daddy
501 41064 41063 0 6:03 ttys001 0:00.04 -zsh
0 41083 41064 0 6:05 ttys001 0:00.01 ps -f -A
ant_daddy@Dmitriy ~ %
```

Эта команда выводит информацию обо всех процессах на компьютере.

\$ps -la (\$ps -mAl)

```
ant_daddy@Dmitriy ~ % ps -la
  UID  PID  PPID  F CPU PRI NI  SZ  RSS WCHAN  S  ADDR  TTY  TIME CMD
    0  41038  1004  4106  0  31  0 34161192 4000 - Ss  0  ttys000 0:00.03 login -pf ant_da
501  41039 41038 4006  0  31  0 34186088 2404 - S+  0  ttys000 0:00.08 -zsh
    0  41063  1004  4106  0  31  0 34186792 4240 - Ss  0  ttys001 0:00.03 login -pf ant_da
501  41064 41063 4006  0  31  0 34187112 2388 - S  0  ttys001 0:00.04 -zsh
    0  41087 41064 4106  0  31  0 34131312 1208 - R+  0  ttys001 0:00.01 ps -la
ant_daddy@Dmitriy ~ %
```

Эта команда выводит информацию в формате long listing

значений колонок:

- UID идентификатор пользователя
- PID идентификатор процесса
- PPID идентификатор родительского процесса
- C использование CPU
- STIME время запуска
- TTY терминал

- TIME общее время работы процесса

- CMD команда

Законспектируйте информацию об использованных Вами опциях команды **ps**.

Для последней команды расшифруйте значения колонок листинга.

Определите, какой процесс является родителем процесса **pr2**, и какой - родителем его родителя и т.д.

Введите команду **\$top**.

```
ant_daddy — top — 136x24
Processes: 714 total, 2 running, 712 sleeping, 2767 threads                                06:09:39
Load Avg: 2.25, 2.59, 2.67  CPU usage: 8.19% user, 6.28% sys, 85.52% idle  SharedLibs: 557M resident, 69M data, 45M linkedit.
MemRegions: 317590 total, 3360M resident, 160M private, 1325M shared. PhysMem: 13G used (3425M wired, 1352M compressor), 3013M unused.
VM: 37T vsize, 4499M framework vsize, 103591827(0) swapins, 110271179(0) swapouts.
Networks: packets: 122546095/144G in, 40916900/5836M out. Disks: 39905265/1320G read, 29052514/1050G written.

PID  COMMAND      %CPU  TIME    #TH   #WQ  #PORT  MEM    PURG   CMPRS  PGRP  PPID  STATE  BOOSTS      %CPU_ME  %CPU_OTHS  UID
173  WindowServer 63.8  33:31:20 20    10   7401+ 1202M- 99M+   327M- 173   1    sleeping *0[1]      0.08085  0.05325  88
0    kernel_task  11.0  13:12:55 243/8 0      0      206M+  0B      0B      0      0      running  0[0]      0.00000  0.00000  0
41094 top          9.6   00:02:00 1/1    0      28     7852K  0B      0B      41094 41064 running  *0[1]      0.00000  0.00000  0
41095 screencaptur 7.4   00:00:31 6       3     246   5092K+ 620K    0B      1011  1011 sleeping *0[163+]  0.20778  0.00000  501
34360 Google Chrom 4.3   03:12:88 22      1     334   586M-  0B      32M     10695 10695 sleeping *0[7]      0.00000  0.00000  501
462   com.apple.Ap 3.9   47:43:38 4       3     374   3236K  0B      2236K   462   1    sleeping 0[1]      0.00000  0.00000  270
230   coreaudioid 2.7   10:33:29 16      8     3670  21M     0B      8848K   230   1    sleeping *0[1]      0.00000  0.00000  202
166   bluetoothhd 2.6   45:27:00 8       3     356   7240K  384K    2000K   166   1    sleeping *0[1]      0.00000  0.00000  0
8166 Telegram      2.4   04:45:18 42      7     3347  958M   8064K   661M    8166  1    sleeping *65568[60798] 0.01981  0.00000  501
1006 Microsoft Wo 1.0   26:00:75 44      6     1430  565M- 113M    297M    1006  1    sleeping *0[54608]  0.00000  0.00000  501
10724 Google Chrom 0.9   19:43:55 15      2     267   17M     0B      14M     10695 10695 sleeping *0[6]      0.00000  0.00000  501
41081 screencaptur 0.9   00:05:15 5       3     227   24M     308K    0B      41081 1    sleeping *0[443+]  0.00000  0.20778  501
1004 Terminal      0.7   02:58:52 14      7     565   75M+   2176K+ 18M     1004  1    sleeping *0[43557]  0.03071  0.00000  501
10695 Google Chrom 0.2   01:46:19 46      2     1970  308M   156K    98M     10695 1    sleeping *1[1294]  0.00000  0.00000  501
683   searchpartyu 0.2   03:26:56 5       3     190   8384K  0B      4112K   683   1    sleeping 0[5654]  0.04194  0.00000  501
1     launchd      0.2   00:46:73 4       3     3940  24M     0B      7592K   1      0    sleeping 0[0]      0.00000  0.09447  0
124   remoted      0.1   02:28:36 4       3     838+  1820K+ 0B      660K    124   1    sleeping *0[1]      0.00000  0.09715  0
```

Активные процессы и использование ресурсов.

Введите команду **\$pstree** и зарисуйте часть дерева процессов для **pr2**. Рядом с процессами укажите их PID (для этого введите **pstree** с соответствующим флагом).

```
ant_daddy — -zsh — 136x24
41077  1 /System/Library/PrivateFrameworks/UniversalAccess.framework/Version
60231  1 /System/Library/CoreServices/ControlCenter.app/Contents/XPCServices
60232  1 /usr/sbin/BlueTool -R
64094  1 /System/Library/PrivateFrameworks/MobileSoftwareUpdate.framework/Ve
68205  1 /System/Library/PrivateFrameworks/IASUtilities.framework/Versions/A
68207  1 /private/var/db/com.apple.xpc.roleaccountd.staging/exec/16777221.49
68208  1 /usr/libexec/xpcroleaccountd -launchd
68453  1 /System/Library/PrivateFrameworks/BridgeOSInstallReporting.framework
41038  1004 login -pf ant_daddy
41039 41038 -zsh
41051 41039 ./Pr2
```

PID Pr2 = 41051

PPID Pr2 = 41039

Родительский процесс для Pr2 – процесс PID = 41039

Введем `ps tree -p 41051`

```
ant_daddy@Dmitriy ~ % ps tree -p 41039
--= 00001 root /sbin/launchd
\--= 01004 ant_daddy /System/Applications/Utilities/Terminal.app/Contents/MacOS/Terminal
\--= 41038 root login -pf ant_daddy
\--= 41039 ant_daddy -zsh
```

2.3. Модифицируйте программу **pr2.c**, удалив один **fork**. После оставшегося **fork** разделите код для родительского и дочернего процессов и выведите следующую информацию:

Для родительского процесса

"Родительский процесс с PID=... PID дочернего процесса=..."

Для дочернего процесса

"Дочерний процесс с PID=... PID родительского процесса=..."

Затем после **fork** задержите выполнение родительского процесса на 3 единицы времени (при помощи библиотечной функции **sleep**), а дочернего - на 10 единиц. Распечатайте в дочернем процессе PID его родителя дважды: до и после **sleep**. Объясните в отчете Ваши результаты.

```
pr2.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main() {
    pid_t pid;

    // Оставляем только один fork
    pid = fork();

    if (pid < 0) {
        fprintf(stderr, "Ошибка при вызове fork\n");
        return 1;
    } else if (pid == 0) {
        // Дочерний процесс
        printf("Дочерний процесс с PID=%d PID родительского процесса=%d\n", getpid(), getppid());

        // Выводим PID родителя до и после sleep
        printf("PID родителя до sleep=%d\n", getpid());
        sleep(10);
        printf("PID родителя после sleep=%d\n", getppid());
    } else {
        // Родительский процесс
        printf("Родительский процесс с PID=%d PID дочернего процесса=%d\n", getpid(), pid);

        // Задержка выполнения родительского процесса на 3 единицы времени
        sleep(3);
    }

    return 0;
}
```



```
ant_daddy@Dmitriy 123 % gcc pr2.c -o Pr2
ant_daddy@Dmitriy 123 % ./Pr2
Родительский процесс с PID=42712  PID дочернего процесса=42714
Дочерний процесс с PID=42714  PID родительского процесса=42712
PID родителя до sleep=42712
ant_daddy@Dmitriy 123 % PID родителя после sleep=1
```

Переключение процесса на выполнение другой программы. Семейство функций **exec** содержит набор функций, которые переключают процесс на выполнение другой программы, т.е. изменяют образ процесса. При этом прекращается выполнение текущей программы и начинается выполнение в этом же процессе другой программы, начиная с ее точки входа. Функции семейства **exec** различаются набором аргументов:

- Функции, которые содержат в имени букву **p** (**execvp** и **execlp**), принимают имя программы и ищут программу по имени во всех путях, содержащихся в PATH; в функции, не содержащие в имени букву **p**, должно быть передано полное имя пути программы.
- Функции, содержащие в имени букву **v** (**execv**, **execvp** и **execve**), принимают список параметров для новой программы как массив указателей на строки с последним нулевым элементом. Каждая строка соответствует одному параметру. В функции, содержащие в имени букву **l** (**execl**, **execlp**, **execle**), параметры передаются как параметры соответствующей функции **exec**.
- Функции, которые содержат в имени букву **e** (**execve** и **execle**), принимают дополнительный параметр - массив переменных окружения. Этот параметр должен быть массивом указателей на строки, с последним нулевым элементом. Каждая строка должна иметь вид "ПЕРЕМЕННАЯ=значение".

Т.к. **exec** заменяет вызывающую программу на другую, то **exec** никогда не возвращает управление в вызывающую программу в случае успеха. Таким образом, код, находящийся после **exec**, будет выполнен только в случае ошибки в вызове **exec**. Список параметров, передаваемых в вызываемую программу,

является аналогом параметров командной строки. Первым параметром (***argv[0]**) в нем должно быть имя программы.

Один из вариантов запуска новых программ - в основной программе создать дочерний процесс (**fork**) и в нем переключиться на другую программу (**exec**), как в задании 3:

Задание 3.

3.1. Выполните программу **pr3.c**, которая в дочернем процессе вызывает программу **ls** с теми же параметрами, что и в задании 2.1. Определите, какой оператор(ы) не выполняется при успешном выполнении **exec**.

```
/* pr3.c */
#include <stdio.h>
#include <unistd.h>
int main()
{char* arg_list[] = {"ls", "-l", "/", NULL};
  if (fork() == 0)
  {
    execvp("ls", arg_list);
    printf("Return after exec\n");
  }
  return 0;
}
```



```

[ant_daddy@Dmitriy 123 % gcc pr3_1.c -o Pr3_1
[ant_daddy@Dmitriy 123 % ./Pr3_1
ant_daddy@Dmitriy 123 % total 9
drwxrwxr-x  25 root  admin   800 12 июн 01:03 Applications
drwxr-xr-x  71 root  wheel  2272 9 май 22:44 Library
drwxr-xr-x@ 10 root  wheel   320 21 мар 09:13 System
drwxr-xr-x   6 root  admin   192 9 май 22:42 Users
drwxr-xr-x   3 root  wheel    96 12 июн 04:12 Volumes
drwxr-xr-x@ 39 root  wheel  1248 21 мар 09:13 bin
drwxr-xr-x   2 root  wheel    64 1 мар 2020 cores
dr-xr-xr-x   3 root  wheel  4444 12 июн 04:12 dev
lrwxr-xr-x@ 1 root  wheel    11 21 мар 09:13 etc -> private/etc
lrwxr-xr-x   1 root  wheel    25 12 июн 04:12 home -> /System/Volume
drwxr-xr-x   5 root  wheel   160 16 мар 00:15 opt
drwxr-xr-x   6 root  wheel   192 12 июн 04:12 private
drwxr-xr-x@ 64 root  wheel  2048 21 мар 09:13 sbin
lrwxr-xr-x@ 1 root  wheel    11 21 мар 09:13 tmp -> private/tmp
drwxr-xr-x@ 11 root  wheel   352 21 мар 09:13 usr
lrwxr-xr-x@ 1 root  wheel    11 21 мар 09:13 var -> private/var

```

3.2. Модифицируйте программу: замените `execvp` на `execclp`:

```
execclp("ls", "ls", "-l", "/", NULL);
```

```

clang: error: no input files
[ant_daddy@Dmitriy 123 % gcc pr3_2.c -o Pr3_2
[ant_daddy@Dmitriy 123 % ./Pr3_2
ant_daddy@Dmitriy 123 % total 9
drwxrwxr-x  25 root  admin   800 12 июн 01:03 Applications
drwxr-xr-x  71 root  wheel  2272 9 май 22:44 Library
drwxr-xr-x@ 10 root  wheel   320 21 мар 09:13 System
drwxr-xr-x   6 root  admin   192 9 май 22:42 Users
drwxr-xr-x   3 root  wheel    96 12 июн 04:12 Volumes
drwxr-xr-x@ 39 root  wheel  1248 21 мар 09:13 bin
drwxr-xr-x   2 root  wheel    64 1 мар 2020 cores
dr-xr-xr-x   3 root  wheel  4444 12 июн 04:12 dev
lrwxr-xr-x@ 1 root  wheel    11 21 мар 09:13 etc -> private/etc
lrwxr-xr-x   1 root  wheel    25 12 июн 04:12 home -> /System/Volume
drwxr-xr-x   5 root  wheel   160 16 мар 00:15 opt
drwxr-xr-x   6 root  wheel   192 12 июн 04:12 private
drwxr-xr-x@ 64 root  wheel  2048 21 мар 09:13 sbin
lrwxr-xr-x@ 1 root  wheel    11 21 мар 09:13 tmp -> private/tmp
drwxr-xr-x@ 11 root  wheel   352 21 мар 09:13 usr
lrwxr-xr-x@ 1 root  wheel    11 21 мар 09:13 var -> private/var

```

3.3. Сделайте ошибку в вызове **exec**: а) укажите вместо **"ls"** имя несуществующей программы; б) задайте несуществующий каталог для **ls**. Различаются ли при выполнении случаи а) и б) ?

А)

```
pr3_3.c
#include <stdio.h>
#include <unistd.h>
int main()
{char* arg_list[] = {"ls", "-l", "/", NULL};
  if (fork() == 0)
  {
    execlp("lssl", "lssl", NULL);
    printf("Return after exec\n");
  }

  return 0;
}
```

```
nt_daddy@Dmitriy 123 % gcc pr3_3.c -o Pr3_3
nt_daddy@Dmitriy 123 % gcc pr3_3.c -o Pr3_3
nt_daddy@Dmitriy 123 % ./Pr3_3
nt_daddy@Dmitriy 123 % Return after exec
```

Б)

```
pr3_3.c
#include <stdio.h>
#include <unistd.h>
int main()
{char* arg_list[] = {"ls", "-l", "/", NULL};
  if (fork() == 0)
  {
    execlp("ls", "ls", "/nesushestv_directory", NULL);
    printf("Return after exec\n");
  }

  return 0;
}
```

```
[gcc pr3_3.c -o Pr3_3
ant_daddy@Dmitriy 123 % ./Pr3_3
ant_daddy@Dmitriy 123 % ls: /nesushestv_directory: No such file or direc
```

3.4. Модифицируйте программу: загрузите в дочерний процесс (используйте **execl**) любую вашу программу (например, печатающую *"Hello, world"* из лабораторной работы 2.

```
pr3_4.c
#include <stdio.h>
#include <unistd.h>
int main()
{char* arg_list[] = {"/hello_world", NULL};
  if (fork() == 0)
  {
    execlp("/hello_world", "/hello_world", NULL);
    printf("Return after exec\n");
  }

  return 0;
}
```

```
ant_daddy@Dmitriy 123 % gcc pr3_4.c -o Pr3_4
ant_daddy@Dmitriy 123 % ./Pr3_4
ant_daddy@Dmitriy 123 % Hello, world
```

4.1. Подготовьте произвольный текстовый файл для копирования размером 500-1000 байтов. Можно взять исходный текст данной программы.

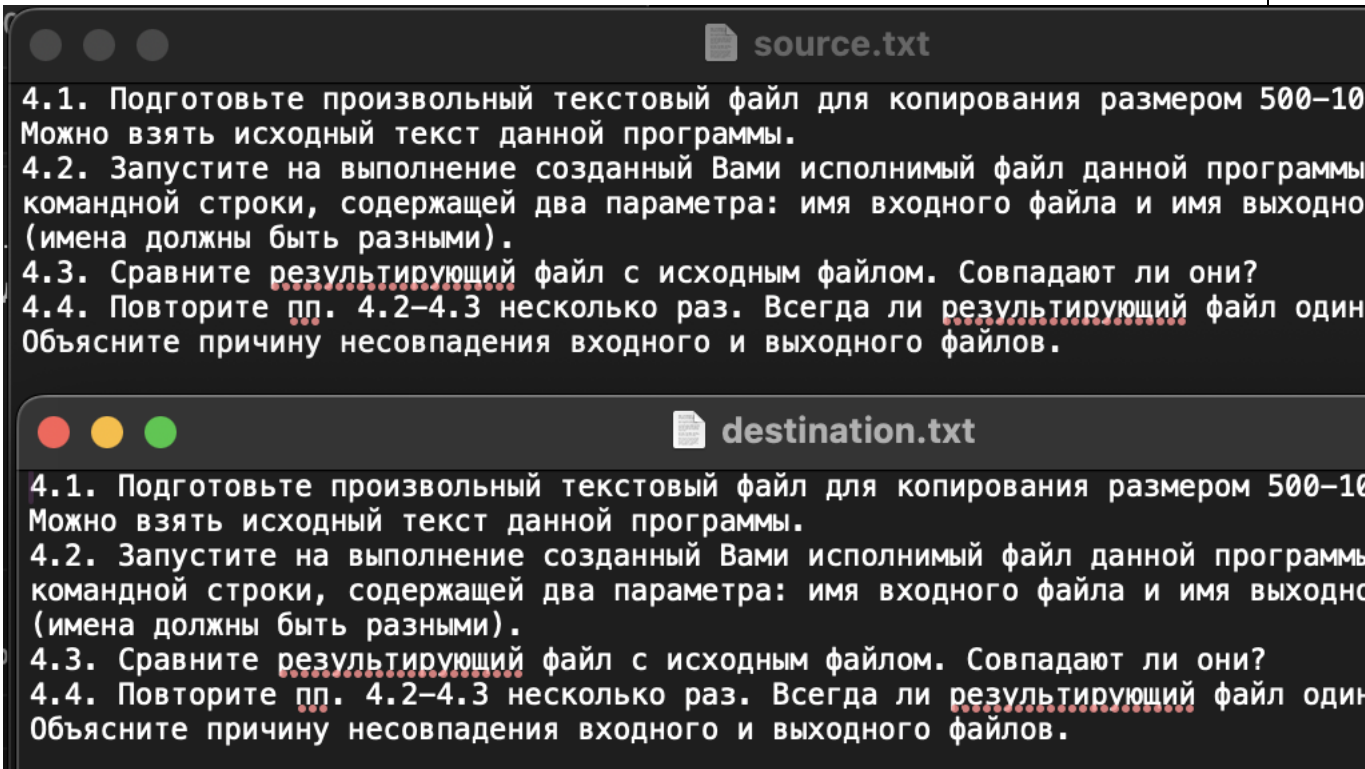
```
source.txt
4.1. Подготовьте произвольный текстовый файл для копирования размером 500–1000 байтов. Можно взять исходный текст данной программы.
4.2. Запустите на выполнение созданный Вами исполнимый файл данной программы при помощи командной строки, содержащей два параметра: имя входного файла и имя выходного файла (имена должны быть разными).
4.3. Сравните результирующий файл с исходным файлом. Совпадают ли они?
4.4. Повторите пп. 4.2–4.3 несколько раз. Всегда ли результирующий файл один и тот же? Объясните причину несовпадения входного и выходного файлов.
```

тип: документ
Размер: 988 Б (1000 байт)

4.2. Запустите на выполнение созданный Вами исполнимый файл данной программы при помощи командной строки, содержащей два параметра: имя входного файла и имя выходного файла (имена должны быть разными).

```
ant_daddy@Dmitriy 123 % gcc pr4.c -o Pr4
ant_daddy@Dmitriy 123 % ./Pr4
Copying completed in 0.001114 seconds.
ant_daddy@Dmitriy 123 %
```

4.3. Сравните результирующий файл с исходным файлом. Совпадают ли они?



Содержимое файлов совпадает

4.4. Повторите пп. 4.2-4.3 несколько раз. Всегда ли результирующий файл один и тот же? Объясните причину несовпадения входного и выходного файлов. Примите во внимание, что процессы читают и записывают по одному байту каждой операцией **read** или **write**.

```
[ant_daddy@Dmitriy 123 % gcc pr4.c -o Pr4
[ant_daddy@Dmitriy 123 % ./Pr4
Copying completed in 0.001114 seconds.
[ant_daddy@Dmitriy 123 % ./Pr4
Copying completed in 0.000526 seconds.
[ant_daddy@Dmitriy 123 % ./Pr4
Copying completed in 0.000587 seconds.
[ant_daddy@Dmitriy 123 % ./Pr4
Copying completed in 0.000578 seconds.
[ant_daddy@Dmitriy 123 % ./Pr4
Copying completed in 0.000529 seconds.
[ant_daddy@Dmitriy 123 % ./Pr4
Copying completed in 0.000601 seconds.
[ant_daddy@Dmitriy 123 % ./Pr4
Copying completed in 0.000559 seconds.
[ant_daddy@Dmitriy 123 % ./Pr4
Copying completed in 0.000733 seconds.
[ant_daddy@Dmitriy 123 % ./Pr4
Copying completed in 0.000593 seconds.
[ant_daddy@Dmitriy 123 % ./Pr4
Copying completed in 0.000495 seconds.
[ant_daddy@Dmitriy 123 % ./Pr4
Copying completed in 0.000463 seconds.
ant_daddy@Dmitriy 123 %
```

За все итерации выходной файл destination.txt не изменился.

4.5. Добавьте в Вашу программу возможность измерять время (в микросекундах), необходимое для копирования файла. (Используйте функцию **gettimeofday**. Ее описание получите с помощью команды **man**.). Замерьте время копирования файла двумя процессами. Затем для сравнения исключите один процесс из операции копирования и замерьте время, необходимое для копирования файла одним процессом. Запишите в отчет Ваши результаты и выводы.