

Отчет по лабораторным

(позже здесь появится форматирование, титульник, оглавление и т.д. Сейчас же просто скриншоты и общее описание)

Лабораторная работа №1

```
dumpling@thinkpad-l380: ~  
Файл Правка Вид Поиск Терминал Справка  
dumpling@thinkpad-l380:~$ pwd  
/home/dumpling  
dumpling@thinkpad-l380:~$ cd ChatProject  
dumpling@thinkpad-l380:~/ChatProject$ ls  
c_cpp_properties.json  database.c      old_versions    temp.c  
client                database.db     protocol.c      tempDir  
clientepoll           header.h       protocol.o      test1.db  
clientepoll.c         list.c         README.md      users.db  
clientpipe            list.o         server          'Безымянный документ 1'  
clientpipe.c          MakefileServer server.c        'Безымянный документ 2'  
database              ncursesTest    server.o  
dumpling@thinkpad-l380:~/ChatProject$ nano server.c  
dumpling@thinkpad-l380:~/ChatProject$ cd ..  
dumpling@thinkpad-l380:~$ cd Unix-Labs/  
dumpling@thinkpad-l380:~/Unix-Labs$ vim lab1.txt  
dumpling@thinkpad-l380:~/Unix-Labs$ mkdir temp1  
dumpling@thinkpad-l380:~/Unix-Labs$ cd temp1  
dumpling@thinkpad-l380:~/Unix-Labs/temp1$ mkdir temp2  
dumpling@thinkpad-l380:~/Unix-Labs/temp1$ cd temp2  
dumpling@thinkpad-l380:~/Unix-Labs/temp1/temp2$ touch text1.txt  
dumpling@thinkpad-l380:~/Unix-Labs/temp1/temp2$ touch text2.txt  
dumpling@thinkpad-l380:~/Unix-Labs/temp1/temp2$ ll  
итого 8  
drwxr-xr-x 2 dumpling dumpling 4096 сен 19 23:13 ./  
drwxr-xr-x 3 dumpling dumpling 4096 сен 19 23:13 ../  
-rw-r--r-- 1 dumpling dumpling  0 сен 19 23:13 text1.txt  
-rw-r--r-- 1 dumpling dumpling  0 сен 19 23:13 text2.txt  
dumpling@thinkpad-l380:~/Unix-Labs/temp1/temp2$ cd ~  
dumpling@thinkpad-l380:~$
```

nano server.c

```
dumpling@thinkpad-l380: ~/ChatProject
Файл Правка Вид Поиск Терминал Справка
GNU nano 2.9.3 server.c

#include "header.h"

static void *get_in_addr(struct sockaddr *sa){
    if(sa->sa_family == AF_INET){
        return &(((struct sockaddr_in*)sa)->sin_addr);
    }
    return &(((struct sockaddr_in6*)sa)->sin6_addr);
}

static void setnonblocking(int fd) {
    int flag = fcntl(fd, F_GETFL, 0);
    fcntl(fd, F_SETFL, flag | O_NONBLOCK);
}

/*
 * Binding, setting as non-blocking and listening sockfd
 * Returning: sockfd in correct case, -1 in error case
 */
[ Read 237 lines ]
^G Помощь ^O Записать ^W Поиск ^K Вырезать ^J Выровнять ^C ТекПозиц
^X Выход ^R ЧитФайл ^\ Замена ^U Отмен. выр ^T Словарь ^_ К строке
```

vim lab1.txt

```
dumpling@thinkpad-l380: ~/Unix-Labs
Файл Правка Вид Поиск Терминал Справка
1 This is the temporary text file
2 for GNU/Linux course
3
4
```

Лабораторная работа №2

Цель работы: Знакомство с характерной для Linux схемой порождения и завершения процессов, с отношениями типа потомок – родитель, со способами передачи информации о событии завершения процесса.

Ход работы:

Были скомпилированы и выполнены примеры программ forkdemo.cpp , tinymenu.cpp , tinyexit.cpp , procgroup.cpp , wait_parent.cpp

```
dumpling@thinkpad-l380: ~/Unix-Labs/fork_exec_wait
Файл Правка Вид Поиск Терминал Справка
dumpling@thinkpad-l380:~/Unix-Labs/fork_exec_wait$ ./tinymenu
0=who, 1=ls, 2=date:1
forkdemo      procgroup.cpp  tinymenu      wait_child.cpp
forkdemo.cpp  tinyexit      tinymenu.cpp  wait_parent
procgroup     tinyexit.cpp  wait_child    wait_parent.cpp
dumpling@thinkpad-l380:~/Unix-Labs/fork_exec_wait$ ./tinyexit
0=who, 1=ls, 2=date:0
dumpling :0      2019-09-20 12:08 (:0)
0=who, 1=ls, 2=date:2
Пт сен 20 12:18:48 MSK 2019
0=who, 1=ls, 2=date:4
dumpling@thinkpad-l380:~/Unix-Labs/fork_exec_wait$ ./procgroup

Initial process      PID    5917      PPID    3655      GID    5917

New process          PID    5918      PPID    5917      GID    5917
New process          PID    5919      PPID    5917      GID    5917
New process          PID    5920      PPID    5917      GID    5917
New process          PID    5921      PPID    5918      GID    5917
New process          PID    5922      PPID    5919      GID    5917
New process          PID    5924      PPID    5921      GID    5917
New process          PID    5923      PPID    5918      GID    5917
dumpling@thinkpad-l380:~/Unix-Labs/fork_exec_wait$
```

```
dumpling@thinkpad-l380: ~/Unix-Labs/fork_exec_wait
Файл Правка Вид Поиск Терминал Справка
dumpling@thinkpad-l380:~/Unix-Labs/fork_exec_wait$ ./wait_parent
Forked child 5969
Forked child 5970
Forked child 5971
Child 5970 is terminating with signal 0009
Wait on PID: 5970 returns status of: 0009
Child 5969 is terminating with exit (0051)
Wait on PID: 5969 returns status of: 5100
Child 5971 is terminating with exit (0053)
Wait on PID: 5971 returns status of: 5300
dumpling@thinkpad-l380:~/Unix-Labs/fork_exec_wait$
```

Был изменен файл forkdemo.cpp, программа была запущена в фоновом режиме, номера PID процессов (родителя и потомка) были получены с помощью команд ps, top. Так как программа не дает существенной нагрузки на CPU и память (и поэтому «теряется» в массе других процессов), команду top следует вызывать с ключом -p[PID]

```
forkdemo.cpp
1  #include<stdio.h>
2  #include<sys/types.h>
3  #include<unistd.h>
4  main() {
5      int i;
6      if (fork()) {
7          //printf("\nParent process: PID %d \n", getpid());
8          sleep(600);
9      }
10     else {
11         //printf("\nChild process: PPID %d \n", getpid());
12         sleep(600);
13     }
14 }
```

ps (process status)

```
dumpling@thinkpad-l380: ~/Unix-Labs/fork_exec_wait
Файл Правка Вид Поиск Терминал Справка
dumpling@thinkpad-l380:~/Unix-Labs/fork_exec_wait$ ./forkdemo &
[1] 6929
dumpling@thinkpad-l380:~/Unix-Labs/fork_exec_wait$ ps
  PID TTY          TIME CMD
 3655 pts/1    00:00:00 bash
 6929 pts/1    00:00:00 forkdemo
 6930 pts/1    00:00:00 forkdemo
 6931 pts/1    00:00:00 ps
dumpling@thinkpad-l380:~/Unix-Labs/fork_exec_wait$
```

top -p6929 -p6930

```
dumpling@thinkpad-l380: ~/Unix-Labs/fork_exec_wait
Файл Правка Вид Поиск Терминал Справка
top - 13:09:12 up 1:00, 1 user, load average: 0,17, 0,21, 0,23
Tasks: 2 total, 0 running, 2 sleeping, 0 stopped, 0 zombie
%Cpu(s): 1,9 us, 0,6 sy, 0,0 ni, 97,3 id, 0,1 wa, 0,0 hi, 0,1 si, 0,0 st
КиБ Mem : 8060364 total, 3611960 free, 2023920 used, 2424484 buff/cache
КиБ Swap: 2097148 total, 2097148 free, 0 used. 5435844 avail Mem

  PID USER      PR  NI   VIRT   RES   SHR  S  %CPU  %MEM    TIME+  COMMAND
 6929 dumpling  20   0   4376    756    692  S   0,0   0,0   0:00.00  forkdemo
 6930 dumpling  20   0   4376     72     0  S   0,0   0,0   0:00.00  forkdemo
```

Результат вызова pstree (часть дерева, ветка systemd, показывающая два вызванных процесса forkdemo)

```
gnome-terminal- bash- forkdemo- forkdemo
                  |    |
                  |    +-- pstree
                  |
                  +-- 3*[{gnome-terminal-}]
```

Было проверено, изменяет ли процесс-родитель директорию, в которой он был запущен, при смене каталога у процесса-потомка (нет, не изменяет).

```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
#include <string.h>

int main() {
    char cwd[256];
    if (fork()) {
        printf("\nParent first path\n");
        if(getcwd(cwd, sizeof(cwd)) == 0) {
            perror("getcwd() error");
        }
        printf("%s\n",cwd);
        cwd[0] = '\0';
        sleep(3);
        printf("\nParent second path\n");
        if(getcwd(cwd, sizeof(cwd)) == 0) {
            perror("getcwd() error");
        }
        printf("%s\n",cwd);
        sleep(1);
    }
    else {
        if(chdir("./temp") != 0){
            perror("chdir() error");
        }
        printf("\nChild path\n");
        if(getcwd(cwd, sizeof(cwd)) == NULL) {
            perror("getcwd() error");
        }
        printf("%s\n",cwd);
        sleep(1);
    }
}
```

Результат работы программы:

```
dumpling@thinkpad-l380:~/Unix-Labs/fork_exec_wait$ ./forkdemo_cwd

Parent first path
/home/dumpling/Unix-Labs/fork_exec_wait

Child path
/home/dumpling/Unix-Labs/fork_exec_wait/temp

Parent second path
/home/dumpling/Unix-Labs/fork_exec_wait
```

Была написана программа, иллюстрирующая, как процесс-родитель и процесс-потомок работают с одним текстовым файлом и обрабатывают смещение. Код программы:

```
forkdemo.cpp
1  #include<stdio.h>
2  #include<sys/types.h>
3  #include<unistd.h>
4  #include <string.h>
5
6  main() {
7      int i;
8      FILE *fp;
9      if((fp = fopen("textfile.txt", "r")) == NULL) {
10         perror("fopen() error");
11     }
12     if (fork()) {
13         sleep(2);
14         char buf[16] = {"\0"};
15         fread(buf, 10, 1, fp);
16         printf("Parent read 10 bytes: %s\n", buf);
17         fclose(fp);
18     }
19     else {
20         char buf[16] = {"\0"};
21         fread(buf, 10, 1, fp);
22         printf("Child read 10 bytes: %s\n", buf);
23     }
24 }
```

Текстовый документ и результат выполнения программы:

```
GNU nano 2.9.3 textfile.txt
This is a text file
```

```
dumpling@thinkpad-l380:~/Unix-Labs/fork_exec_wait$ ./forkdemo
Child read 10 bytes: This is a
Parent read 10 bytes: text file
```

Таким образом, при открытии файла в процессе родителя сам родитель и его потомок работают с общим смещением текстового файла.

Вывод:

Лабораторная работа №3

Цель работы: Изучение конвейеров (pipes, программных каналов), как простейшего средства коммуникации запущенных процессов. Исследование различных способов организации каналов и их сопоставление.

Ход работы:

1. Была скомпилирована и выполнена программа whosortpipe.cpp, сопоставлены результаты выполнения программы и команды `who | sort`.

Конструкция вида `who | sort`, запускаемая в shell, создает два конкурентных процесса, соединенных конвейером так, что поток вывода первого из них попадает в поток ввода второго.

В программе whosortpipe реализуется типовой подход к созданию однонаправленного канала передачи данных между процессами потомками. Родительский процесс создает конвейер и двух потомков, потомки закрывают ненужный им «конец» конвейера, а оставшийся перенаправляют в свои стандартные вводы/выводы

```
whosortpipe.cpp > ...
1  /* The program whosortpipe.cpp */
2  #include <stdio.h>
3  #include <unistd.h>
4  #include <stdlib.h>
5  #include <sys/wait.h>
6  main() {
7      int fds[2];
8      pipe(fds); /* Create the pipe */
9      /* First child reconnects stdin to downstream end of pipe
10       and closes the upstream end */
11      if (fork() == 0) {
12          dup2(fds[0], 0);
13          close(fds[1]);
14          execlp("sort", "sort", NULL);
15      }
16      /* Second child reconnects stdout to upstream end of pipe and
17       closes the downstream end */
18      else if (fork() == 0) {
19          dup2(fds[1], 1);
20          close(fds[0]);
21          execlp("who", "who", NULL);
22      }
23      /* Parent closes both ends of pipe and waits for both
24       children to finish */
25      else {
26          close(fds[0]);
27          close(fds[1]);
28          wait(0);
29          wait(0);
30      }
31  }
```

```
dumpling@thinkpad-l380: ~/Unix-Labs/pipe
Файл Правка Вид Поиск Терминал Справка
dumpling@thinkpad-l380:~/Unix-Labs/pipe$ g++ whosortpipe.cpp -o whosortpipe
dumpling@thinkpad-l380:~/Unix-Labs/pipe$ ./whosortpipe
dumpling :0                2019-09-20 17:23 (:0)
dumpling@thinkpad-l380:~/Unix-Labs/pipe$ who | sort
dumpling :0                2019-09-20 17:23 (:0)
dumpling@thinkpad-l380:~/Unix-Labs/pipe$
```

2. Была скомпилирована и выполнена программа cmdpipe.cpp с аргументами last more, сопоставлены результаты выполнения программы и команды last | more.

```
dumpling@thinkpad-l380: ~/Unix-Labs/pipe
Файл Правка Вид Поиск Терминал Справка
dumpling@thinkpad-l380:~/Unix-Labs/pipe$ ./cmdpipe last more
dumpling :0                :0                Fri Sep 20 17:23    still logged in
reboot   system boot      4.15.0-62-generi Fri Sep 20 17:22    still running
dumpling :0                :0                Fri Sep 20 12:08 - down    (02:30)
reboot   system boot      4.15.0-62-generi Fri Sep 20 12:08 - 14:39  (02:30)
dumpling :0                :0                Fri Sep 20 10:03 - down    (00:23)
reboot   system boot      4.15.0-62-generi Fri Sep 20 10:03 - 10:26  (00:23)
dumpling :0                :0                Thu Sep 19 22:06 - down    (02:04)
reboot   system boot      4.15.0-62-generi Thu Sep 19 22:06 - 00:10  (02:04)
dumpling :0                :0                Thu Sep 19 00:29 - down    (00:23)
reboot   system boot      4.15.0-62-generi Thu Sep 19 00:29 - 00:52  (00:23)
dumpling :0                :0                Wed Sep 18 19:23 - down    (00:13)
reboot   system boot      4.15.0-62-generi Wed Sep 18 19:23 - 19:37  (00:14)
dumpling :0                :0                Wed Sep 18 14:01 - down    (02:59)
reboot   system boot      4.15.0-62-generi Wed Sep 18 14:01 - 17:00  (02:59)
dumpling :0                :0                Wed Sep 18 10:31 - down    (02:59)
reboot   system boot      4.15.0-60-generi Wed Sep 18 10:30 - 13:31  (03:00)
dumpling :0                :0                Wed Sep 18 00:07 - down    (00:22)
reboot   system boot      4.15.0-60-generi Wed Sep 18 00:06 - 00:30  (00:23)
dumpling :0                :0                Mon Sep 16 18:28 - down    (05:29)
reboot   system boot      4.15.0-60-generi Mon Sep 16 18:27 - 23:58  (05:30)
dumpling :0                :0                Mon Sep 16 10:43 - 15:38  (04:55)
reboot   system boot      4.15.0-60-generi Mon Sep 16 10:43 - 15:38  (04:55)
dumpling :0                :0                Sun Sep 15 16:55 - 01:07  (08:12)
```

```
dumpling@thinkpad-l380: ~/Unix-Labs/pipe
Файл Правка Вид Поиск Терминал Справка
dumpling@thinkpad-l380:~/Unix-Labs/pipe$ last | more
dumpling :0                :0                Fri Sep 20 17:23    still logged in
reboot   system boot      4.15.0-62-generi Fri Sep 20 17:22    still running
dumpling :0                :0                Fri Sep 20 12:08 - down    (02:30)
reboot   system boot      4.15.0-62-generi Fri Sep 20 12:08 - 14:39  (02:30)
dumpling :0                :0                Fri Sep 20 10:03 - down    (00:23)
reboot   system boot      4.15.0-62-generi Fri Sep 20 10:03 - 10:26  (00:23)
dumpling :0                :0                Thu Sep 19 22:06 - down    (02:04)
reboot   system boot      4.15.0-62-generi Thu Sep 19 22:06 - 00:10  (02:04)
dumpling :0                :0                Thu Sep 19 00:29 - down    (00:23)
reboot   system boot      4.15.0-62-generi Thu Sep 19 00:29 - 00:52  (00:23)
dumpling :0                :0                Wed Sep 18 19:23 - down    (00:13)
reboot   system boot      4.15.0-62-generi Wed Sep 18 19:23 - 19:37  (00:14)
dumpling :0                :0                Wed Sep 18 14:01 - down    (02:59)
reboot   system boot      4.15.0-62-generi Wed Sep 18 14:01 - 17:00  (02:59)
dumpling :0                :0                Wed Sep 18 10:31 - down    (02:59)
reboot   system boot      4.15.0-60-generi Wed Sep 18 10:30 - 13:31  (03:00)
dumpling :0                :0                Wed Sep 18 00:07 - down    (00:22)
reboot   system boot      4.15.0-60-generi Wed Sep 18 00:06 - 00:30  (00:23)
dumpling :0                :0                Mon Sep 16 18:28 - down    (05:29)
reboot   system boot      4.15.0-60-generi Mon Sep 16 18:27 - 23:58  (05:30)
dumpling :0                :0                Mon Sep 16 10:43 - 15:38  (04:55)
reboot   system boot      4.15.0-60-generi Mon Sep 16 10:43 - 15:38  (04:55)
dumpling :0                :0                Sun Sep 15 16:55 - 01:07  (08:12)
```


С помощью вызова `ropen()` нельзя создать программу, организующую конвейер из трех команд `shell` , передаваемых ей в качестве параметров командной строки при запуске. Команда `ropen()` реализует:

- вызов `pipe()`;
- вызов `fork()` (создание процесса-потомка);
- закрытие ненужных дескрипторов в родителе и потомке;
- замену процесса-потомка с помощью `exec`.

То есть вызов сразу выполняет команду, при этом каждый отдельный вызов не может одновременно читать и писать. Таким образом, в каждый момент времени мы можем либо писать (класть) команду в канал, либо читать (не сохраняя "состояние" и не передавая его дальше на чтение следующему каналу).

3.

=====!!!!!!!=====

Лабораторная работа №4

Цель работы: Знакомство с важным атрибутом любой операционной системы - переменными среды (или переменными окружения) и с возможностями их использования в Linux. Освоение языка для составления командных сценариев и написание набора полезных для системного администрирования скриптов.

Ход работы:

1. Был написан командный файл first.sh, выполняющий разные операции со строками. Код скрипта:

```
str1="Hello"
str2="World"
echo "str1: $str1"
echo "str2: $str2"

concatStr="${str1}${str2}"
echo "concat strings: $concatStr"

subStr=$(echo $concatStr | cut -c 2-5)
echo "substring (characters from 2 to 5): $subStr"

newStr=${concatStr/Hello/Hi}
echo "changed string: $newStr"
```

Результат работы:

```
dumpling@thinkpad-l380:~/Unix-Labs/envvar$ ./first.sh
str1: Hello
str2: World
concat strings: HelloWorld
substring (characters from 2 to 5): ello
changed string: HiWorld
```

2. Был написан командный файл second.sh, выполняющий разные операции с числами. Код скрипта:

```
var1=`expr 1 + 2`
echo "adding: 1+2 by expr. Result: $var1"

let var2=2*2
echo "multiplying: 2*2 by let. Result: $var2"

var3=$((8-6))
echo "subtracting: 8-6 by double brackets. Result: $var3"

var4="scale=2; 5/4"
echo "dividing: 5/4 using bc. Result"
echo $var4 | bc

var5=$((2+2*2))
echo "calculate expression: 2+2*2 by double brackets. Result: $var5 "
```

Результат работы:

```
dumpling@thinkpad-l380:~/Unix-Labs/envvar$ bash second.sh
adding: 1+2 by expr. Result: 3
multiplying: 2*2 by let. Result: 4
subtracting: 8-6 by double brackets. Result: 2
dividing: 5/4 using bc. Result
1.25
calculate expression: 2+2*2 by double brackets. Result: 6
```

Лабораторная работа №5

Цель работы: Манипуляции с правами доступа при создании в системе учетных записей и исследование влияния прав на файловые операции. Изучение специфик фонового (background) и диалогового (foreground) режимов исполнения процессов и способов переключений между этими режимами.

Ход работы:

1. Some bullshit

2. Были написаны две небольшие программы, выводящие сообщения с задержкой. Первая программа была запущенна в фоновом режиме, вторая - в обычном режиме. Вывод этих двух процессов на консоль перемежается.

```
dumpling@thinkpad-l380:~/Unix-Labs/process_groups$ bash bg_prog.sh &
[1] 27435
dumpling@thinkpad-l380:~/Unix-Labs/process_groups$ bash fg_prog.sh
I'm from fg and I'm writing this every second
I'm from fg and I'm writing this every second
I'm from fg and I'm writing this every second
Program from bg prints this text every 5 seconds.
I'm from fg and I'm writing this every second
I'm from fg and I'm writing this every second
I'm from fg and I'm writing this every second
I'm from fg and I'm writing this every second
I'm from fg and I'm writing this every second
Program from bg prints this text every 5 seconds.
I'm from fg and I'm writing this every second
I'm from fg and I'm writing this every second
I'm from fg and I'm writing this every second
I'm from fg and I'm writing this every second
```

Вывод первой программы был перенаправлен в файл output.txt. После этого вывод двух процессов был разделен:

```
dumpling@thinkpad-l380:~/Unix-Labs/process_groups$ bash bg_prog.sh > output.txt &
[1] 27566
dumpling@thinkpad-l380:~/Unix-Labs/process_groups$ bash fg_prog.sh
I'm from fg and I'm writing this every second
I'm from fg and I'm writing this every second
I'm from fg and I'm writing this every second
I'm from fg and I'm writing this every second
I'm from fg and I'm writing this every second
I'm from fg and I'm writing this every second
I'm from fg and I'm writing this every second
I'm from fg and I'm writing this every second
^C
dumpling@thinkpad-l380:~/Unix-Labs/process_groups$ fg
bash bg_prog.sh > output.txt
^C
dumpling@thinkpad-l380:~/Unix-Labs/process_groups$ cat output.txt
Program from bg prints this text every 5 seconds.
Program from bg prints this text every 5 seconds.
Program from bg prints this text every 5 seconds.
```

3. Был написан скрипт, который ждет в обычном режиме одну строку, а дальше продолжает бесконечно выводить строки. Для перехода в бэкграунд после ввода нужно воспользоваться Ctrl+Z

=====!!!=====

4. Была написана полезная программа, которая архивирует директорию в нужную папку раз в n секунд.

```
name=$1
path=$2
timeout=$3
counter=0

while [ ${timeout} -gt 1 ]
do
    tar -czf "${path}/${name}${counter}".tar.gz ${name}
    let counter++
    echo "Archieved ${counter} times"
    sleep ${timeout}
done
```

Результат работы:

```
dumpling@thinkpad-l380:~/Unix-Labs/process_groups$ cd archive/
dumpling@thinkpad-l380:~/Unix-Labs/process_groups/archive$ ls -l
итого 0
dumpling@thinkpad-l380:~/Unix-Labs/process_groups/archive$ cd ..
dumpling@thinkpad-l380:~/Unix-Labs/process_groups$ bash archiveByTimeout.sh temp ~/Unix-Labs/process_groups/archive 5
Archieved 1 times
Archieved 2 times
Archieved 3 times
^C
dumpling@thinkpad-l380:~/Unix-Labs/process_groups$ cd archive/
dumpling@thinkpad-l380:~/Unix-Labs/process_groups/archive$ ls -l
итого 12
-rw-r--r-- 1 dumpling dumpling 164 ноя 24 02:22 temp0.tar.gz
-rw-r--r-- 1 dumpling dumpling 164 ноя 24 02:22 temp1.tar.gz
-rw-r--r-- 1 dumpling dumpling 164 ноя 24 02:22 temp2.tar.gz
dumpling@thinkpad-l380:~/Unix-Labs/process_groups/archive$
```

Вывод:

Лабораторная работа №6

Цель работы: Освоение простейшего средства управления процессами, позволяющего процессам передавать информацию о каких-либо событиях, отрабатывать реакции на различные события и взаимодействовать друг с другом.

Ход работы:

1. Дважды была запущена программа sigint.cpp, в первый раз завершена сигналом прерывания SIGINT (Ctrl-C) с терминала, второй - завершение с дампом памяти сигналом «Quit» с терминала (Ctrl-\\).

```
dumpling@thinkpad-l380:~/Unix-Labs/signal$ gcc sigint.cpp -o sigint
dumpling@thinkpad-l380:~/Unix-Labs/signal$ ./sigint
Enter a string:
^CAhhh! SIGINT!
fgets: Interrupted system call
dumpling@thinkpad-l380:~/Unix-Labs/signal$ ./sigint
Enter a string:
^\\Выход (стек памяти сброшен на диск)
```

2. Там непонятная дичка какая-то с SIGCONT
- 3.

Цель работы: Освоение семафоров (semaphores) как эффективных средств синхронизации доступа процессов к разделяемым ресурсам операционной системы, а также синхронизации доступа потоков к разделяемым ресурсам процесса.

Ход работы:

1. Была скомпилирована и запущена программа gener_sem.cpp, после каждого вызова выполнена команда `ipcs -s`.

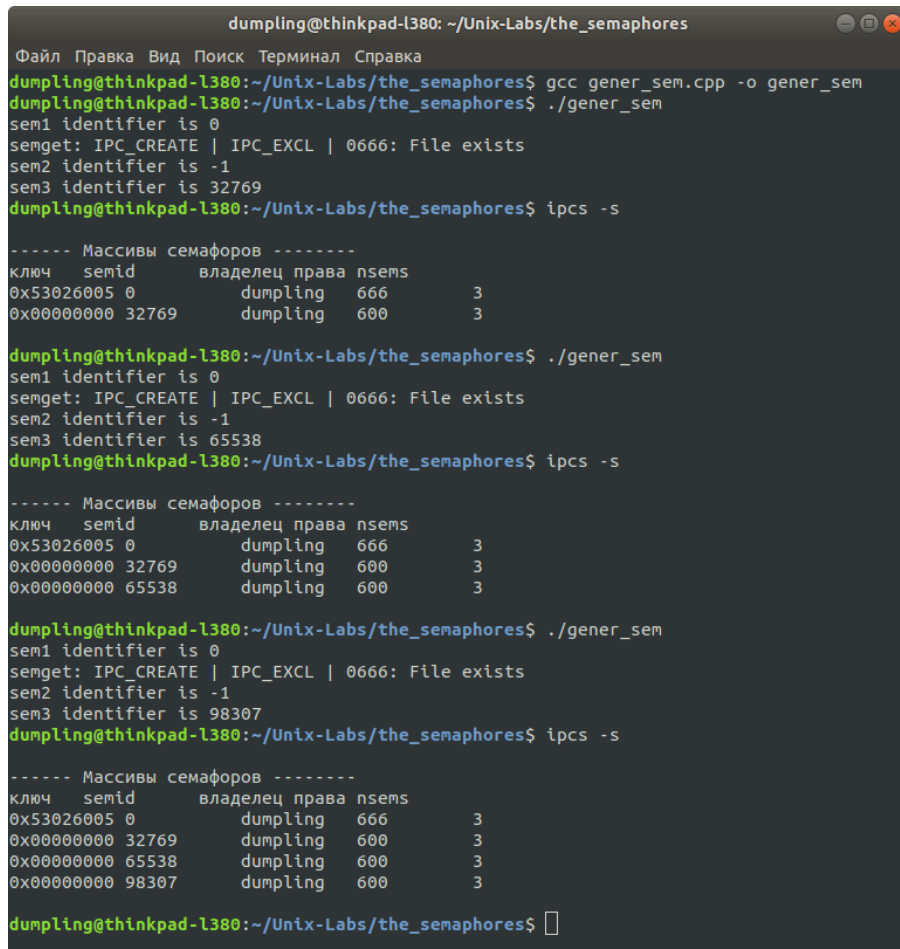
Системный вызов `semget(key_t key, int nsems, int semflg);` возвращает идентификатор набора семафоров, связанный с аргументом `key`. Создается новый набор из семафоров `nsems`, если значение `key` равно `IPC_PRIVATE` или с ключом `key` не связано ни одного существующего набора семафора, а выражение `semflg & IPC_CREAT` истинно.

Если `semflg` имеет поля `IPC_CREAT` и `IPC_EXCL`, а набор семафоров уже существует для `key`, то `semget()` не выполняется с кодом ошибки `EEXIST`

Младшие 9 битов аргумента `semflg` определяют права доступа (для владельца, группы и остальных) к набору семафоров.

При запуске данной программы многократно будет видно, что

- набор `sem1` будет создан единожды, а затем каждая новая попытка будет всего лишь открывать доступ к уже существующему ресурсу;
- попытки создания набора `sem2` на том же ключе всегда будут приводить к ошибке из-за наличия флагов `IPC_CREATE | IPC_EXCL`, не допускающих открытия ресурса вместо его создания;
- набор `sem3` будет создаваться при каждом новом запуске программы. Причем каждый раз с новым уникальным идентификатором.



```
dumpling@thinkpad-l380: ~/Unix-Labs/the_semaphores
Файл Правка Вид Поиск Терминал Справка
dumpling@thinkpad-l380:~/Unix-Labs/the_semaphores$ gcc gener_sem.cpp -o gener_sem
dumpling@thinkpad-l380:~/Unix-Labs/the_semaphores$ ./gener_sem
sem1 identifier is 0
semget: IPC_CREATE | IPC_EXCL | 0666: File exists
sem2 identifier is -1
sem3 identifier is 32769
dumpling@thinkpad-l380:~/Unix-Labs/the_semaphores$ ipcs -s

----- Массивы семафоров -----
ключ  semid      владелец права nsems
0x53026005 0          dumpling  666        3
0x00000000 32769      dumpling  600        3

dumpling@thinkpad-l380:~/Unix-Labs/the_semaphores$ ./gener_sem
sem1 identifier is 0
semget: IPC_CREATE | IPC_EXCL | 0666: File exists
sem2 identifier is -1
sem3 identifier is 65538
dumpling@thinkpad-l380:~/Unix-Labs/the_semaphores$ ipcs -s

----- Массивы семафоров -----
ключ  semid      владелец права nsems
0x53026005 0          dumpling  666        3
0x00000000 32769      dumpling  600        3
0x00000000 65538      dumpling  600        3

dumpling@thinkpad-l380:~/Unix-Labs/the_semaphores$ ./gener_sem
sem1 identifier is 0
semget: IPC_CREATE | IPC_EXCL | 0666: File exists
sem2 identifier is -1
sem3 identifier is 98307
dumpling@thinkpad-l380:~/Unix-Labs/the_semaphores$ ipcs -s

----- Массивы семафоров -----
ключ  semid      владелец права nsems
0x53026005 0          dumpling  666        3
0x00000000 32769      dumpling  600        3
0x00000000 65538      dumpling  600        3
0x00000000 98307      dumpling  600        3

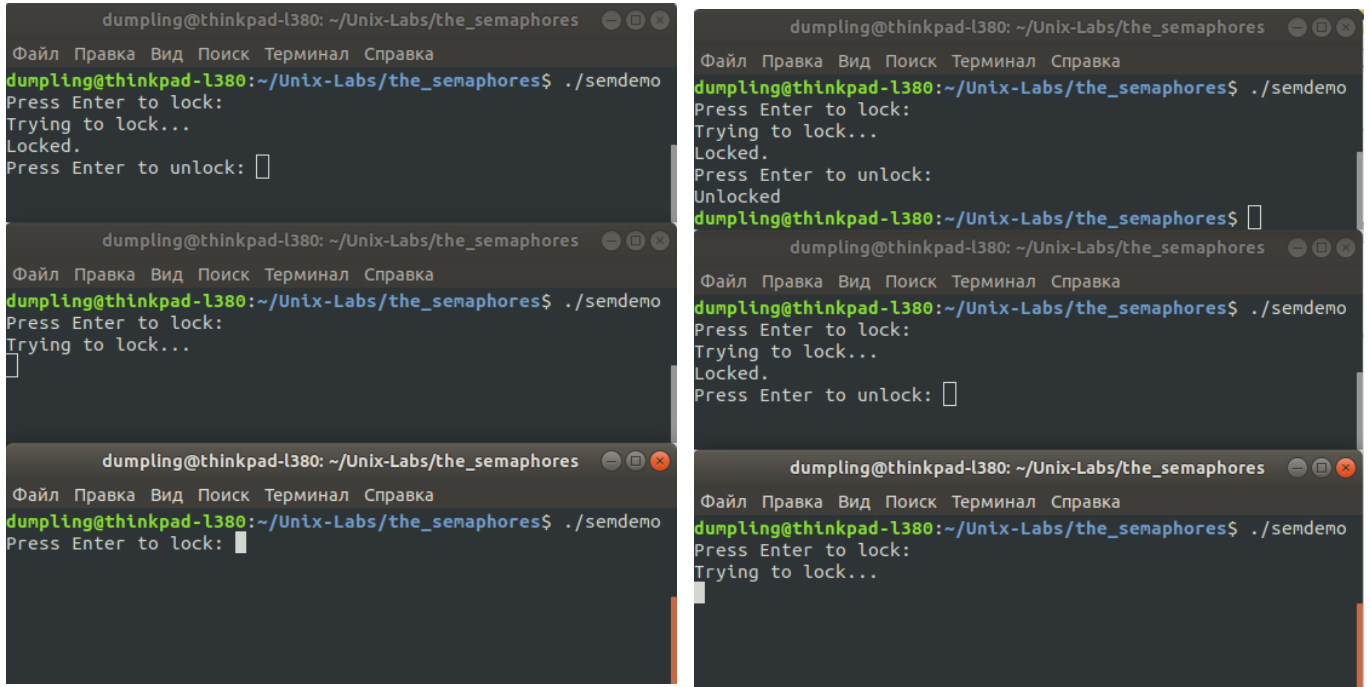
dumpling@thinkpad-l380:~/Unix-Labs/the_semaphores$
```

2. Созданные семафоры были удалены с помощью команды `ipcrm -s semid`

3. Был скомпилирован файл `semdemo.cpp`, организующий разделение доступа к общему ресурсу между несколькими процессами с помощью технологии семафоров. Программа была запущена в нескольких терминалах одновременно.

Ситуация 1: верхний терминал заблокировал доступ к ресурсу остальным терминалам, во втором терминале ждут, пока ресурс освободился

Ситуация 2: верхний терминал разлочил доступ, второй захватил ресурс, третий находится в ожидании



4. Была скомпилирована программа `semrm.cpp`, которая удаляет семафор, созданный на прошлом шаге. Программа может удалить данный семафор, т. к. она задает тот же ключ `key = ftok(".", u_char)` (где `u_char = 'J'`; ключи семафоров уникальны)

```
dumpling@thinkpad-l380:~/Unix-Labs/the_semaphores$ ipcs -s

----- Массивы семафоров -----
ключ  semid      владелец права nsems
0x53026005 0          dumpling  666      3
0x4a026005 131073      dumpling  666      1

dumpling@thinkpad-l380:~/Unix-Labs/the_semaphores$ gcc semrm.cpp -o semrm
dumpling@thinkpad-l380:~/Unix-Labs/the_semaphores$ ./semrm
dumpling@thinkpad-l380:~/Unix-Labs/the_semaphores$ ipcs -s

----- Массивы семафоров -----
ключ  semid      владелец права nsems
0x53026005 0          dumpling  666      3
```

5. Если запустить программу `semrm` во время исполнения `semdemo`, то последняя упадет с ошибкой, например «semop: Invalid argument» (код ошибки зависит от момента, в каком состоянии `semdemo` был удален семафор)


```
dumpling@thinkpad-l380: ~/Unix-Labs/the_semaphores
Файл Правка Вид Поиск Терминал Справка
dumpling@thinkpad-l380:~/Unix-Labs/the_semaphores$ ./semdemo
Press Enter to lock:
Trying to lock...
semop: Invalid argument
dumpling@thinkpad-l380:~/Unix-Labs/the_semaphores$

dumpling@thinkpad-l380: ~/Unix-Labs/the_semaphores
Файл Правка Вид Поиск Терминал Справка
dumpling@thinkpad-l380:~/Unix-Labs/the_semaphores$ ipcs -s

----- Массивы семафоров -----
ключ  semid  владелец права nsems
0x53026005 0      dumpling  666      3
0x4a026005 163841  dumpling  666      1

dumpling@thinkpad-l380:~/Unix-Labs/the_semaphores$ ./semrm
dumpling@thinkpad-l380:~/Unix-Labs/the_semaphores$ ipcs -s

----- Массивы семафоров -----
ключ  semid  владелец права nsems
0x53026005 0      dumpling  666      3

dumpling@thinkpad-l380:~/Unix-Labs/the_semaphores$
```

6. Программа semdemo.cpp была улучшена (был добавлен специальный флаг isWorking), теперь процессу предоставляется возможность после освобождения ресурса становиться снова в очередь на повторное его занятие (а не завершаться), при этом завершение процесса происходит по вводу символа Q.

```
dumpling@thinkpad-l380:~/Unix-Labs/the_semaphores$ gcc semdemo.cpp -o semdemo
dumpling@thinkpad-l380:~/Unix-Labs/the_semaphores$ ./semdemo
press Enter

Press Enter to lock:
Trying to lock...
Locked.
Press Enter to unlock or press Q to quit:
Unlocked
Press Enter to lock:
Trying to lock...
Locked.
Press Enter to unlock or press Q to quit: Q
Unlocked
dumpling@thinkpad-l380:~/Unix-Labs/the_semaphores$
```

7. Была составлена программа semmonitor.cpp, позволяющая мониторить количество процессов (типа semdemo), находящихся в состоянии ожидания освобождения ресурса (Trying to lock...) в каждый момент времени. Код программы и результат выполнения (при разных запусках и завершениях программ semdemo)

```
dumpling@thinkpad-l380:~/Unix-Labs/the_semaphores$ gcc semmonitor.cpp -o semmoni
tor
dumpling@thinkpad-l380:~/Unix-Labs/the_semaphores$ ./semmonitor
Number of waiting processes: 1
Number of waiting processes: 1
Number of waiting processes: 2
Number of waiting processes: 2
Number of waiting processes: 2
Number of waiting processes: 2
Number of waiting processes: 1
Number of waiting processes: 1
Number of waiting processes: 0
Number of waiting processes: 0
```

```

semmonitor.cpp > main(void)
1  /*
2  ** semmonitor.cpp -- checks the number of processes which is waiting for resources unlock
3  */
4
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <errno.h>
8  #include <sys/types.h>
9  #include <sys/ipc.h>
10 #include <sys/sem.h>
11 #include <unistd.h>
12
13 int main(void) {
14     key_t key;
15     int semid;
16
17     if ((key = ftok(".", 'J')) == -1) {
18         perror("ftok");
19         exit(1);
20     }
21
22     /* grab the semaphore set created by seminit.c: */
23     if ((semid = semget(key, 1, 0)) == -1) {
24         perror("semget");
25         exit(1);
26     }
27
28     int numberOfProcesses = 0;
29     while (true) {
30         if ((numberOfProcesses = semctl(semid, 0, GETNCNT)) == -1) {
31             perror("semctl");
32             exit(1);
33         }
34         printf("Number of waiting processes: %d\n", numberOfProcesses);
35         sleep(2);
36     }
37     return 0;
38 }

```

Лабораторная работа №8

Цель работы: Знакомство с возможностями очередей сообщений (Message Queues) – мощного и гибкого средства межпроцессного взаимодействия в ОС Linux.

Ход работы:

1. Была скомпилирована и запущена программа `gener_mq.cpp`. В процессе выполнения программы создается очередь из пяти сообщений, выводится информация об очередях с помощью `ipcs -q` с помощью `msgctl()` все очереди сообщений удаляются (что видно при вызове `ipcs -q` после исполнения программы).

```
dumpling@thinkpad-l380:~/Unix-Labs/the_mess_que$ gcc gener_mq.cpp -o gener_mq
dumpling@thinkpad-l380:~/Unix-Labs/the_mess_que$ ./gener_mq

----- Очереди сообщений -----
ключ  msqid      владелец права исп.  байты сообщения
0x41026018 163840      dumpling  660      0          0
0x42026018 196609      dumpling  660      0          0
0x43026018 229378      dumpling  660      0          0
0x44026018 262147      dumpling  660      0          0
0x45026018 294916      dumpling  660      0          0

dumpling@thinkpad-l380:~/Unix-Labs/the_mess_que$ ipcs -q

----- Очереди сообщений -----
ключ  msqid      владелец права исп.  байты сообщения
```

2. Процессы могут записывать и считывать сообщения из очередей. Процесс, пославший сообщение в очередь, может не ожидать чтения этого сообщения каким-либо другим процессом. Он может закончить свое выполнение, оставив в очереди сообщение, которое будет прочитано другим процессом позже.

Программы `sender.cpp` и `receiver.cpp` передают друг другу сообщения, используя очереди сообщений:

```
dumpling@thinkpad-l380: ~/Unix-Labs/the_mess_que
Файл Правка Вид Поиск Терминал Справка
dumpling@thinkpad-l380:~/Unix-Labs/the_mess_que$ ./sender
Enter lines of text, ^D to quit:
Send first line
Send second line
^D

dumpling@thinkpad-l380: ~/Unix-Labs/the_mess_que
Файл Правка Вид Поиск Терминал Справка
dumpling@thinkpad-l380:~/Unix-Labs/the_mess_que$ ./receiver
spock: ready to receive messages, captain.
spock: "Send first line"
spock: "Send second line"
```

При этом в случае, если запущен только файл `sender.o`, то в очереди сохраняются все сообщения, еще не принятые принимающей стороной:

```
dumpling@thinkpad-l380:~/Unix-Labs/the_mess_que$ ./sender
Enter lines of text, ^D to quit:
1
2
3
4
5
```

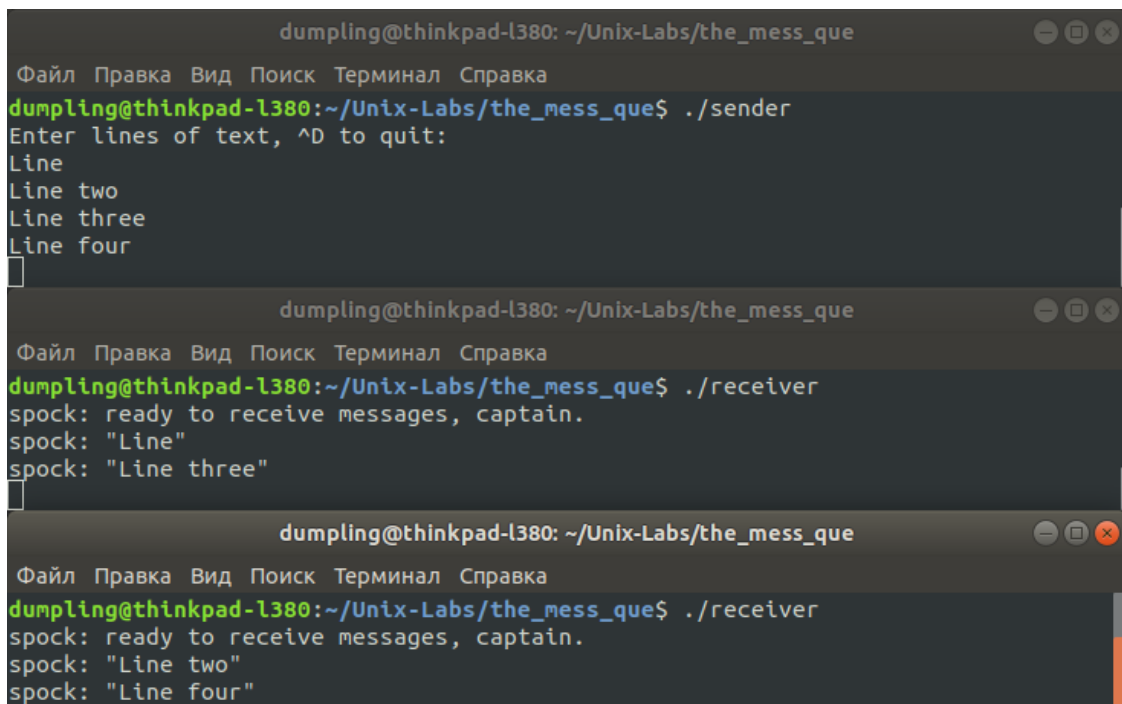
```
dumpling@thinkpad-l380:~$ ipcs -q
----- Очереди сообщений -----
ключ  msqid      владелец права исп. байты сообщения
0x42026018 32768      dumpling  644      15      5
```

При запуске ресивера, он вычитывает все сообщения, и очередь остается пустой:

```
dumpling@thinkpad-l380:~/Unix-Labs/the_mess_que$ ./receiver
spock: ready to receive messages, captain.
spock: "1"
spock: "2"
spock: "3"
spock: "4"
spock: "5"
```

```
dumpling@thinkpad-l380:~$ ipcs -q
----- Очереди сообщений -----
ключ  msqid      владелец права исп. байты сообщения
0x42026018 32768      dumpling  644      0      0
```

3. При запуске нескольких receiver очередь сообщений вычитывается каждым получателем по очереди:



The image shows three terminal windows stacked vertically, each running the same program in the directory ~/Unix-Labs/the_mess_que. The top window is running ./sender and has entered four lines of text: "Line", "Line two", "Line three", and "Line four". The middle window is running ./receiver and has received the first two messages: "Line" and "Line three". The bottom window is also running ./receiver and has received the remaining two messages: "Line two" and "Line four". This demonstrates that multiple receivers can process messages from a single queue in parallel.

```
dumpling@thinkpad-l380: ~/Unix-Labs/the_mess_que
Файл Правка Вид Поиск Терминал Справка
dumpling@thinkpad-l380:~/Unix-Labs/the_mess_que$ ./sender
Enter lines of text, ^D to quit:
Line
Line two
Line three
Line four
^D

dumpling@thinkpad-l380: ~/Unix-Labs/the_mess_que
Файл Правка Вид Поиск Терминал Справка
dumpling@thinkpad-l380:~/Unix-Labs/the_mess_que$ ./receiver
spock: ready to receive messages, captain.
spock: "Line"
spock: "Line three"
^D

dumpling@thinkpad-l380: ~/Unix-Labs/the_mess_que
Файл Правка Вид Поиск Терминал Справка
dumpling@thinkpad-l380:~/Unix-Labs/the_mess_que$ ./receiver
spock: ready to receive messages, captain.
spock: "Line two"
spock: "Line four"
```

4. Было добавлено мультиплексирование, теперь отправитель явно задает mtype очереди, в которую он хочет отправить сообщение, а каждый получатель теперь читает только из своей очереди (указывая нужный mtype в msgrcv)

```
dumpling@thinkpad-l380: ~/Unix-Labs/the_mess_que
Файл Правка Вид Поиск Терминал Справка
dumpling@thinkpad-l380:~/Unix-Labs/the_mess_que$ ./sender
Enter type of queue (1 or 2) & after that type the line of text, ^D to quit:
1 line for the first mq
2 line for the second mq
1 another for first
1 and another
^D

dumpling@thinkpad-l380: ~/Unix-Labs/the_mess_que
Файл Правка Вид Поиск Терминал Справка
dumpling@thinkpad-l380:~/Unix-Labs/the_mess_que$ ./receiver1
1: spock: ready to receive messages, captain.
spock: "1 line for the first mq"
spock: "1 another for first"
spock: "1 and another"
^D

dumpling@thinkpad-l380: ~/Unix-Labs/the_mess_que
Файл Правка Вид Поиск Терминал Справка
dumpling@thinkpad-l380:~/Unix-Labs/the_mess_que$ ./receiver2
2: spock: ready to receive messages, captain.
spock: "2 line for the second mq"
^D
```

Лабораторная работа №9

Цель работы: Использование для обмена данными разделяемой памяти (shared memory) – самого быстрого средства межпроцессного взаимодействия в Linux.

Ход работы:

1. Была скомпилирована программа gener_shm.cpp, запущена несколько раз:

```
dumpling@thinkpad-l380:~/Unix-Labs/the_shared_mem$ ./gener_shm
First memory identifiere is 12353558
Second shared memory identifiere is 12386327
dumpling@thinkpad-l380:~/Unix-Labs/the_shared_mem$ ./gener_shm
First memory identifiere is 12353558
Second shared memory identifiere is 12419096
dumpling@thinkpad-l380:~/Unix-Labs/the_shared_mem$ ./gener_shm
First memory identifiere is 12353558
Second shared memory identifiere is 12451865
```

Результат вызова `ipcs -m`. Мы можем видеть, что после исполнения программы gener_shm разделяемая память с идентификатором 12353558 появляется один раз (т. к. `shmget` вызывался три раза с одинаковым ключом: `shmid_1=shmget(key, 1000, 0644|IPC_CREAT)`). При каждом запуске программы также выделяется память с ключом `IPC_PRIVATE`, поэтому количество выделенных кусочков соответствует количеству запусков программы:

0x0000000f	12353558	dumpling	644	1000	0
0x00000000	12386327	dumpling	644	20	0
0x00000000	12419096	dumpling	644	20	0
0x00000000	12451865	dumpling	644	20	0

2. Была скомпилирована программа shmdemo.cpp, которая создает/подключается к выделенной разделяемой памяти, выводит данные, которые лежат в памяти или пишет в память строку, переданную программе в качестве второго аргумента:

```
dumpling@thinkpad-l380:~/Unix-Labs/the_shared_mem$ ./shmdemo
segment contains: ""
dumpling@thinkpad-l380:~/Unix-Labs/the_shared_mem$ ./shmdemo line
writing to segment: "line"
dumpling@thinkpad-l380:~/Unix-Labs/the_shared_mem$ ./shmdemo new_line
writing to segment: "new_line"
dumpling@thinkpad-l380:~/Unix-Labs/the_shared_mem$ ./shmdemo
segment contains: "new_line"
```

При запуске shmdemo в разных терминалах доступ к разделяемой памяти сохраняется. Вычитываются те данные, которые были положены последними (при этом не важно, каким терминалом, что очевидно):

```
dumpling@thinkpad-l380: ~/Unix-Labs/the_shared_mem
Файл Правка Вид Поиск Терминал Справка
dumpling@thinkpad-l380:~/Unix-Labs/the_shared_mem$ ./shmdemo
segment contains: ""
dumpling@thinkpad-l380:~/Unix-Labs/the_shared_mem$ ./shmdemo set_first_line
writing to segment: "set_first_line"
dumpling@thinkpad-l380:~/Unix-Labs/the_shared_mem$ ./shmdemo
segment contains: "set_second_line"
dumpling@thinkpad-l380:~/Unix-Labs/the_shared_mem$ █

dumpling@thinkpad-l380: ~/Unix-Labs/the_shared_mem
Файл Правка Вид Поиск Терминал Справка
dumpling@thinkpad-l380:~/Unix-Labs/the_shared_mem$ ./shmdemo
segment contains: "set_first_line"
dumpling@thinkpad-l380:~/Unix-Labs/the_shared_mem$ ./shmdemo set_second_line
writing to segment: "set_second_line"
dumpling@thinkpad-l380:~/Unix-Labs/the_shared_mem$ ./shmdemo
segment contains: "set_second_line"
dumpling@thinkpad-l380:~/Unix-Labs/the_shared_mem$ █
```

3. Был скомпилирован и запущен файл `attach_shm.cpp`

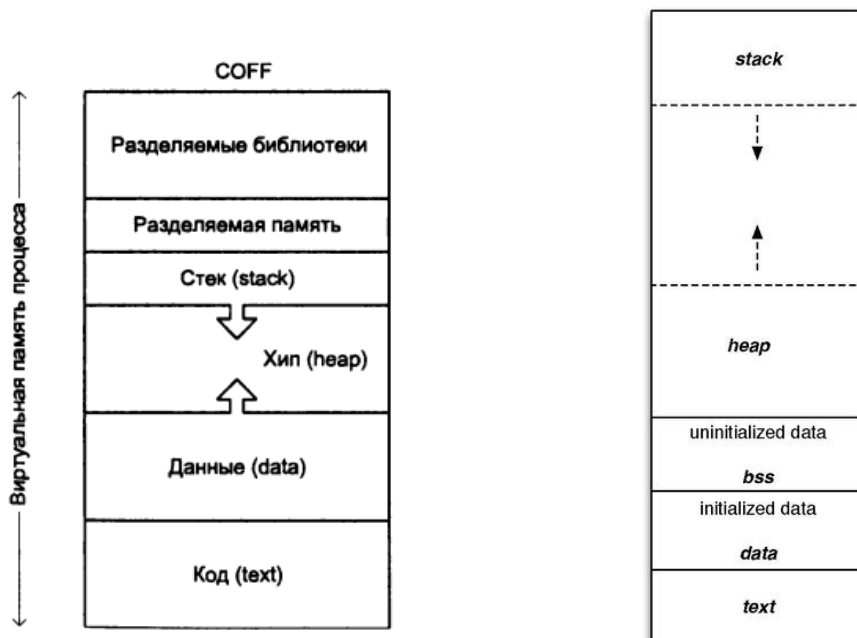
```
dumpling@thinkpad-l380:~/Unix-Labs/the_shared_mem$ ./attach_shm
Addresses in parent

shared mem: 0x7f67298d4000
program text (etext): 0x55ac88af8b2d
initialized data (edata): 0x55ac88cfa010
uninitialized data (end): 0x55ac88cfa018

In parent before fork, memory is : ABCDEFGHIJKLMNOPQRSTUVWXYZ
In child after fork, memory is : ABCDEFGHIJKLMNOPQRSTUVWXYZ

In parent after fork, memory is : abcdefghijklmnopqrstuvwxyz
Parent removing shared memory
```

Коротко рассмотрим строение виртуальной памяти процесса:



Виртуальная память процесса состоит из нескольких сегментов или областей памяти. Размер, содержимое и расположение сегментов в памяти определяется как самой программой, например, использованием библиотек, размером кода и данных, так и форматом исполняемого файла этой программы. Сегмент данных включает инициализированные данные, копируемые в память из соответствующих разделов исполняемого файла, и неинициализированные данные, которые заполняются нулями перед началом выполнения процесса. Неинициализированные данные часто называют сегментом BSS. Именно это показывает нам вывод значений переменных `extern int etext, edata, end;`

Для работы с разделяемой памятью вводятся следующие системные вызовы:

- `shmget` — создание сегмента разделяемой памяти с привязкой к целочисленному идентификатору, либо анонимного сегмента разделяемой памяти (при указании вместо идентификатора значения `IPC_PRIVATE`);
- `shmctl` — установка параметров сегмента памяти;
- `shmat` — подключение сегмента к адресному пространству процесса;
- `shmdt` — отключение сегмента от адресного пространства процесса.

В коде программы `attach_shm` последовательно вызываются `shmget`, `shmat`, `shmdt`, `shmctl`. Вызов последних двух функций позволяет родительскому процессу освобождать сегмент общей памяти из системы.

4. Была написана программа `seq_att_shm.cpp`, создающая три разделяемых сегмента памяти размером 1023 байта каждый:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define SHM_SIZE 1023

extern int etext, edata, end;

main(void) {
    int shmid[3];
    char *shm[3];
    char *str;

    for (int i = 0; i < 3; i++) {
        if ((shmid[i] = shmget(IPC_PRIVATE, SHM_SIZE, IPC_CREAT | 0666)) < 0) {
            perror("shmget fail");
            exit(1);
        }
        if ((shm[i] = (char *) shmat(shmid[i], 0, 0)) == (char *)-1) {
            perror("shmat fail");
            exit(2);
        }
        printf("Shared memory %d: %p\n", i, shm[i]);

        str = shm[i];
        /* str now references shared mem */
        for (int j = 0; j < 1023; j++) {
            *str++ = 'a' + i; /* Put some info there */
        }
        *str = 0; /* Terminate the sequence */

        printf("Info in shared memory %d: %s\n", i, shm[i]);
    }
    *(shm[0] + 1024) = 'W';
    printf("\nTrying to change 1024th byte of first seq of shared memory: %s\n", shm[0]);
    printf("(and after changes let's look on the second one): %s\n", shm[1]);

    *(shm[0] + 2) = 'W';
    printf("\nTrying to change 3rd byte of first seq of shared memory: %s\n", shm[0]);

    for (int i = 0; i < 3; i++) {
        shmdt(shm[i]);
        shmctl(shmid[i], IPC_RMID, (struct shm_ds *) 0);
    }
    exit(0);
}
```


Результат выполнения. Как можно заметить, при попытке изменения данных за пределами размера конкретного сегмента ничего не происходит:

[illegible]

Лабораторная работа №9

Цель работы: Использование для обмена данными разделяемой памяти (shared memory) – самого быстрого средства межпроцессного взаимодействия в Linux.

Ход работы:

1. Была скомпилирована программа socketpair.cpp:

```
dumpling@thinkpad-l380:~/Unix-Labs/the_sockets$ gcc socketpair.cpp -o socketpair
dumpling@thinkpad-l380:~/Unix-Labs/the_sockets$ ./socketpair
parent->child:0
child->parent: 1
parent->child:2
child->parent: 3
parent->child:4
child->parent: 5
parent->child:6
child->parent: 7
parent->child:8
child->parent: 9
```

Временные задержки (sleep()) в процессах родителя и ребенка не изменяют порядок доставки: каждый процесс в состоянии чтения ждет, когда с «другой стороны» придет сообщение, и пока этого не произойдет, исполняться код далее не будет (сокеты создавались с ключом SOCK_STREAM)

2. Были скомпилированы файлы echo_server.cpp и echo_client.cpp. Внутри программ создаются сокеты socket(AF_UNIX, SOCK_STREAM, 0), где AF_UNIX - флаг домена локального межпроцессного взаимодействия в пределах единой операционной системы UNIX, а SOCK_STREAM — флаг сокета потока

```
dumpling@thinkpad-l380: ~/Unix-Labs/the_sockets
Файл Правка Вид Поиск Терминал Справка
dumpling@thinkpad-l380:~/Unix-Labs/the_sockets$ ./echo_server
Waiting for a connection...
Connected.
█

dumpling@thinkpad-l380: ~/Unix-Labs/the_sockets
Файл Правка Вид Поиск Терминал Справка
dumpling@thinkpad-l380:~/Unix-Labs/the_sockets$ ./echo_client
Trying to connect...
Connected.
> Hi there!
echo> Hi there!
> Hello again!
echo> Hello again!
> █
```

Адрес сокета зависит от коммуникационного домена, в рамках которого он определен. Поскольку в данном домене взаимодействующие процессы выполняются под управлением одной операционной системы на одном и том же хосте, коммуникационный узел может быть

однозначно определен одним параметром — локальным процессом. В качестве адреса в домене UNIX используются имена файлов. Именно поэтому создается новый файл `echo_socket` (указываем его как путь).

3. Были скомпилированы файлы `sock_c_i_srv.cpp` и `sock_c_i_clt.cpp`. Внутри программ создаются сокеты `ssocket(AF_INET, SOCK_STREAM, 0)`, где `AF_INET` - флаг домена взаимодействия процессов удаленных систем, а `SOCK_STREAM` — флаг сокета потока

```
dumpling@thinkpad-l380:~/Unix-Labs/the_sockets$ ./sock_c_i_srv
dumpling@thinkpad-l380: ~/Unix-Labs/the_sockets
Файл Правка Вид Поиск Терминал Справка
dumpling@thinkpad-l380:~/Unix-Labs/the_sockets$ ./sock_c_i_clt localhost
> Hi
HI
> Hello
HELLO
> Don't cry on me!
DON'T CRY ON ME!
>
```

4. Код сервера был немного модифицирован (добавлена одна строка). Теперь сервер выводит сообщения, полученные от клиента:

```
dumpling@thinkpad-l380: ~/Unix-Labs/the_sockets
Файл Правка Вид Поиск Терминал Справка
dumpling@thinkpad-l380:~/Unix-Labs/the_sockets$ ./sock_c_i_srv
Client sent: FIRST
Client sent: SECOND
Client sent: FIRST LONG LONG MSG
Client sent: SECOND
dumpling@thinkpad-l380: ~/Unix-Labs/the_sockets
Файл Правка Вид Поиск Терминал Справка
dumpling@thinkpad-l380:~/Unix-Labs/the_sockets$ ./sock_c_i_clt localhost
> First
FIRST
> First long long msg
FIRST LONG LONG MSG
>
dumpling@thinkpad-l380: ~/Unix-Labs/the_sockets
Файл Правка Вид Поиск Терминал Справка
dumpling@thinkpad-l380:~/Unix-Labs/the_sockets$ ./sock_c_i_clt localhost
> Second
SECOND
> Second
SECOND
>
```

Данный сервер способен слушать до 5 клиентов (определяется в системном вызове `listen`).

В мануале приводится крайне понятное объяснение всех системных вызовов, которые необходимы серверу для установки корректного соединения:

To accept connections, the following steps are performed:

1. A socket is created with `socket(2)`.

2. The socket is bound to a local address using `bind(2)`, so that other sockets may be connect(2)ed to it.
3. A willingness to accept incoming connections and a queue limit for incoming connections are specified with `listen()`.
4. Connections are accepted with `accept(2)`