

САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО
ИНСТИТУТ КОМПЬЮТЕРНЫХ НАУК И ТЕХНОЛОГИЙ
КАФЕДРА КОМПЬЮТЕРНЫХ СИСТЕМ И ПРОГРАММНЫХ ТЕХНОЛОГИЙ

Отчёт по лабораторной работе №2
по курсу «Системное программирование»
по теме «Обработка исключений в Linux»

Выполнил студент гр. 13541/2:
Волкова М.Д.

Проверил преподаватель:
Душутина Е. В.

Санкт-Петербург
2019 г.

1 Цель работы

Познакомится с видами исключений в операционной системе Linux и со способами их обработки.

2 Характеристики системы

```
1 lorismelik@lorismelik-Aspire-Z5700:~/$ cat /proc/version
2 Linux version 4.15.0-29-generic (build@lgw01-amd64-057) (gcc version 7.3.0 (Ubuntu 7.3.0-16ubuntu3))
   #31-Ubuntu SMP Tue Jul 17 15:39:52 UTC 2018
3
4 lorismelik@lorismelik-Aspire-Z5700:~/$ gcc --version
5 gcc (Ubuntu 7.3.0-27ubuntu1~18.04) 7.3.0
6 Copyright (C) 2017 Free Software Foundation, Inc.
7 This is free software; see the source for copying conditions.  There is NO
8 warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
9
10 lorismelik@lorismelik-Aspire-Z5700:~/$ strace -V
11 strace -- version UNKNOWN
12 Copyright (c) 1991-2018 The strace developers <https://strace.io>.
13 This is free software; see the source for copying conditions.  There is NO
14 warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
15
16 Optional features enabled: stack-unwind stack-demangle m32-mpers mx32-mpers
17
18 lorismelik@lorismelik-Aspire-Z5700:~/$ ltrace -V
19 ltrace version 0.7.3.
20 Copyright (C) 1997-2009 Juan Cespedes <cespedes@debian.org>.
21 This is free software; see the GNU General Public Licence
22 version 2 or later for copying conditions.  There is NO warranty.
```

3 Ход работы

3.1 Обработка исключений посредством сигналов

В отличие от Windows, в UNIX системах не предусмотрено единого средства для обработки аппаратных и программных исключений, как структурированная обработка исключений (SEH). Для фиксации и обработки аппаратных исключений в Unix системах предусмотрено многоцелевое средство межпроцессного взаимодействия – сигналы. Сигнал представляет собой асинхронное уведомление процесса о возникновении некоторого события.

С точки зрения назначения сигналы в Linux делятся на несколько категорий: исключение, отладка, пользовательские, управление и сигналы POSIX реального времени. В рамках данной работы рассмотрим сигналы, порождаемые программными или аппаратными исключениями:

- SIGBUS – Код: 10. Неправильное обращение в физическую память, по умолчанию завершает процесс с дампом памяти.
- SIGFPE – Код: 8. Ошибочная арифметическая операция, по умолчанию завершает процесс с дампом памяти.
- SIGILL – Код: 4. Недопустимая инструкция процессора, по умолчанию завершает процесс с дампом памяти.
- SIGSEGV – Код: 11. Нарушение при обращении в память, по умолчанию завершает процесс с дампом памяти.
- SIGSYS – Код: 12. Неправильный системный вызов, по умолчанию завершает процесс с дампом памяти.
- SIGXCPU – Код: 30. Процесс превысил лимит процессорного времени, по умолчанию завершает процесс с дампом памяти.
- SIGXFSZ – Код: 31. Процесс превысил допустимый размер файла, по умолчанию завершает процесс с дампом памяти.

В данной работе рассматриваются сигналы SIGFPE (деление на ноль) и SIGBUS (ошибочное обращение в физическую память) как наиболее просто генерируемые.

3.1.1 Генерация сигналов

Рассмотрим программу, генерирующую исключения SIGFPE и SIGBUS, но не имеющую обработчика:

```
1 #include <unistd.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <signal.h>
5 #include <string.h>
6 #include <sys/mman.h>
7
8 void generateArythmeticException() {
9     const int one = 1;
10    const int zero = 0;
11    const int exception = one / zero;
12 }
13
14 void generateBusException() {
15     FILE *f = tmpfile();
16     int *m = mmap(0, 4, PROT_WRITE, MAP_PRIVATE, fileno(f), 0);
17     *m = 0;
18 }
19
20 int main(int argc, char *argv[]){
21     if (argc != 2) {
22         return 0x1;
23     }
24
25     if (strcmp(argv[1], "0") == 0)
26         generateArythmeticException();
27     else if (strcmp(argv[1], "1") == 0)
28         generateBusException();
29
30     return 0x0;
31 }
```

В зависимости от аргумента командной строки, программа генерирует исключение типа SIGFPE или SIGBUS.

```
1 lorismelik@lorismelik-Aspire-Z5700:~/ $ ./task1.o 0
2 Floating point exception (core dumped)
3 lorismelik@lorismelik-Aspire-Z5700:~/ $ ./task1.o 1
4 Bus exception (core dumped)
```

Программы завершились аварийно. Характер завершения различен в зависимости от аргумента командной строки, что можно заметить по выводу.

В ситуациях, когда сигнал-исключение не был обработан программой и она завершила свою работу в аварийном режиме, ядро системы автоматически вносит соответствующие записи об ошибках в системных лог. Рассмотрим подробности завершения программы с помощью утилиты strace, а также проверим системный журнал syslog на наличие информации об исключениях:

```
1 lorismelik@lorismelik-Aspire-Z5700:~/ $ strace ./task1.o 0
2 execve("./task1.o", [ "./task1.o", "0"], 0x7ffe19d0a5e8 /* 57 vars */) = 0
3 brk(NULL) = 0x5641c8c55000
4 access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory)
5 access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
6 openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
7 fstat(3, {st_mode=S_IFREG|0644, st_size=88178, ...}) = 0
8 mmap(NULL, 88178, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fc733f2a000
9 close(3) = 0
10 access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory)
11 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
12 read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\260\34\2\0\0\0\0\0...", 832) = 832
13 fstat(3, {st_mode=S_IFREG|0755, st_size=2030544, ...}) = 0
14 mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fc733f28000
15 mmap(NULL, 4131552, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fc733928000
16 mprotect(0x7fc733b0f000, 2097152, PROT_NONE) = 0
17 mmap(0x7fc733d0f000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7000) = 0
   x7fc733d0f000
18 mmap(0x7fc733d15000, 15072, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0
   x7fc733d15000
19 close(3) = 0
20 arch_prctl(ARCH_SET_FS, 0x7fc733f294c0) = 0
21 mprotect(0x7fc733d0f000, 16384, PROT_READ) = 0
22 mprotect(0x5641c77d3000, 4096, PROT_READ) = 0
23 mprotect(0x7fc733f40000, 4096, PROT_READ) = 0
24 munmap(0x7fc733f2a000, 88178) = 0
25 --- SIGFPE {si_signo=SIGFPE, si_code=FPE_INTDIV, si_addr=0x5641c75d3730} ---
26 +++ killed by SIGFPE (core dumped) +++
27 Floating point exception (core dumped)
28
29
30
31 lorismelik@lorismelik-Aspire-Z5700:~/ $ strace ./task1.o 1
32 execve("./task1.o", [ "./task1.o", "1"], 0x7ffcc7aa9168 /* 57 vars */) = 0
33 brk(NULL) = 0x55bbf2d16000
34 access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory)
```

```

35 access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
36 openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
37 fstat(3, {st_mode=S_IFREG|0644, st_size=88178, ...}) = 0
38 mmap(NULL, 88178, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f6f1cac9000
39 close(3) = 0
40 access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory)
41 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
42 read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\260\34\2\0\0\0\0"..., 832) = 832
43 fstat(3, {st_mode=S_IFREG|0755, st_size=2030544, ...}) = 0
44 mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f6f1cac7000
45 mmap(NULL, 4131552, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f6f1c4c7000
46 mprotect(0x7f6f1c6ae000, 2097152, PROT_NONE) = 0
47 mmap(0x7f6f1c8ae000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7000) = 0
  x7f6f1c8ae000
48 mmap(0x7f6f1c8b4000, 15072, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0
  x7f6f1c8b4000
49 close(3) = 0
50 arch_prctl(ARCH_SET_FS, 0x7f6f1cac84c0) = 0
51 mprotect(0x7f6f1c8ae000, 16384, PROT_READ) = 0
52 mprotect(0x55bbb1e57000, 4096, PROT_READ) = 0
53 mprotect(0x7f6f1cadf000, 4096, PROT_READ) = 0
54 munmap(0x7f6f1cac9000, 88178) = 0
55 openat(AT_FDCWD, "/tmp", O_RDWR|O_EXCL|O_TMPFILE, 0600) = 3
56 fcntl(3, F_GETFL) = 0x418002 (flags O_RDWR|O_LARGEFILE|O_TMPFILE)
57 brk(NULL) = 0x55bbf2d16000
58 brk(0x55bbf2d37000) = 0x55bbf2d37000
59 mmap(NULL, 4, PROT_WRITE, MAP_PRIVATE, 3, 0) = 0x7f6f1cade000
60 --- SIGBUS {si_signo=SIGBUS, si_code=BUS_ADRERR, si_addr=0x7f6f1cade000} ---
61 +++ killed by SIGBUS (core dumped) +++
62 Bus exception (core dumped)
63
64
65
66 lorismelik@lorismelik-Aspire-Z5700:~/ $ cat /var/log/syslog | grep task1
67 Jan 26 15:53:04 lorismelik-Aspire-Z5700 kernel: [ 5403.068337] task1.o[21154]: segfault at 0 ip 000055
  a53d2ab679 sp 00007fffa8c94560 error 6 in task1.o[55a53d2ab000+1000]
68 Jan 26 15:53:07 lorismelik-Aspire-Z5700 kernel: [ 5406.212642] traps: task1.o[21156] trap divide error ip
  :55a7c195a660 sp:7ffd3ec76d40 error:0 in task1.o[55a7c195a000+1000]
69 Jan 26 16:51:34 lorismelik-Aspire-Z5700 kernel: [ 8913.509359] traps: task1.o[22152] trap divide error ip
  :556ab2200730 sp:7ffc1d278480 error:0 in task1.o[556ab2200000+1000]

```

Вывод утилиты `strace` подтверждает завершение процесса с помощью необработанных исключений `SIGFPE` и `SIGBUS`.

3.1.2 Обработка сигналов функцией signal

Для назначения обработчика ненадежных сигналов в Unix-подобных системах используется функция `signal` со следующей сигнатурой:

```
sighandler_t signal(int signum, sighandler_t handler);
```

Прототип имеет два аргумента, первый номер сигнала и второй указатель на функцию обработчика сигналов. Функция обработчика сигналов возвращает `void` и принимает единственный целочисленный аргумент, который представляет номер сигнала, который был отправлен. Таким образом, имеется возможность использовать одну и ту же функцию обработчика сигнала для нескольких разных сигналов.

Для записи в системный журнал используется 3 функции:

```
void openlog(const char *ident, int option, int facility);
void syslog(int priority, const char *format, ...);
void closelog(void);
```

- `openlog()` устанавливает связь с программой, ведущей системный журнал. Строка `ident` добавляется к каждому сообщению и обычно представляет собой название программы. Аргумент `option` указывает флаг управляющий работой `openlog()` и соответствующих вызовов `syslog()`.
- `syslog()` создает сообщение для журнала, которое передается `syslogd`.
- `closelog()` закрывает описатель, используемый для записи данных в журнал. Использование `closelog()` необязательно.

Рассмотрим программу, которая при возникновении исключения вызывает собственный обработчик, в котором составляет сообщение для системного журнала и завершает процесс:

```
1 #include <unistd.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <signal.h>
5 #include <string.h>
6 #include <syslog.h>
```

```

7 #include <sys/mman.h>
8
9 void generateArythmeticException() {
10     const int one = 1;
11     const int zero = 0;
12     const int exception = one / zero;
13 }
14
15 void generateBusException() {
16     FILE *f = tmpfile();
17     int *m = mmap(0, 4, PROT_WRITE, MAP_PRIVATE, fileno(f), 0);
18     *m = 0;
19 }
20
21 void signalArythmeticHandler(const int signalCode) {
22     const char* message = "Arythmetic exception occurred! Code: 0x";
23     printf("%s%x\n", message, signalCode);
24
25     openlog("task2", 0, LOG_USER);
26     syslog(LOG_NOTICE, "%s%x\n", message, signalCode);
27     closelog();
28
29     exit(EXIT_FAILURE);
30 }
31
32 void signalBusHandler(const int signalCode) {
33     const char* message = "Bus exception occurred! Code: 0x";
34     printf("%s%x\n", message, signalCode);
35
36     openlog("task2", 0, LOG_USER);
37     syslog(LOG_NOTICE, "%s%x\n", message, signalCode);
38     closelog();
39
40     exit(EXIT_FAILURE);
41 }
42
43 int main(int argc, char *argv[]) {
44     if (argc != 2) {
45         return 0x1;
46     }
47
48     if (signal(SIGFPE, signalArythmeticHandler) == SIG_ERR)
49         return 0x2;
50
51     if (signal(SIGSEGV, signalBusHandler) == SIG_ERR)
52         return 0x3;
53
54     if (strcmp(argv[1], "0") == 0)
55         generateArythmeticException();
56     else if (strcmp(argv[1], "1") == 0)
57         generateBusException();
58
59     return 0x0;
60 }

```

```

1 lorismelik@lorismelik-Aspire-Z5700:~/ $ ./task2.o 0
2 Arythmetic exception occurred! Code: 0x8
3 lorismelik@lorismelik-Aspire-Z5700:~/ $ ./task2.o 1
4 Bus exception occurred! Code: 0x7

```

Вывод программы свидетельствует о том, что исключения действительно были обработаны. Рассмотрим поведение программы с точки зрения системы:

```

1 lorismelik@lorismelik-Aspire-Z5700:~/ $ strace ./task2.o 0
2 execve("./task2.o", ["/task2.o", "0"], 0x7ffead83a2a8 /* 57 vars */) = 0
3 brk(NULL) = 0x555ba0a30000
4 access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory)
5 access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
6 openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
7 fstat(3, {st_mode=S_IFREG|0644, st_size=88178, ...}) = 0
8 mmap(NULL, 88178, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f71a5533000
9 close(3) = 0
10 access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory)
11 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
12 read(3, "\177ELF\2\1\3\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\260\34\2\0\0\0\0"..., 832) = 832
13 fstat(3, {st_mode=S_IFREG|0755, st_size=2030544, ...}) = 0
14 mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f71a5531000
15 mmap(NULL, 4131552, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f71a4f31000
16 mprotect(0x7f71a518000, 2097152, PROT_NONE) = 0
17 mmap(0x7f71a5318000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7000) = 0
   x7f71a5318000
18 mmap(0x7f71a531e000, 15072, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0
   x7f71a531e000
19 close(3) = 0
20 arch_prctl(ARCH_SET_FS, 0x7f71a55324c0) = 0
21 mprotect(0x7f71a5318000, 16384, PROT_READ) = 0
22 mprotect(0x555b9f407000, 4096, PROT_READ) = 0
23 mprotect(0x7f71a5549000, 4096, PROT_READ) = 0
24 munmap(0x7f71a5533000, 88178) = 0

```

[illegible]

На выводе утилиты `strace` виден момент установки обработчиков сигналов, момент возникновения исключения, а также процесс его обработки.

Записи в журнале `syslog`, которые были добавлены непосредственно внутри собственных обработчиков, действительно были добавлены.

3.1.3 Генерация сигналов функцией raise

Исключение можно генерировать с помощью функции `raise`, имеющей следующую сигнатуру:

```
int raise(int sig);
```

Функция `raise` принимает единственный аргумент, являющийся кодом сигнала. Стоит отметить, что `raise` эквивалентна функции `kill(getpid(), sig)`. Обработку сигналов можно осуществить с помощью функции `sigaction`, имеющей следующую сигнатуру:

```
int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);
```

Системный вызов `sigaction` используется для изменения действий процесса при получении соответствующего сигнала. `Sigaction` намного более опциональна, чем `signal`, а также позволяет задавать обработчики надежных сигналов. Лучше избегать использования функции `signal` и использовать вместо нее `sigaction`.

Структура sigaction имеет следующий формат:

```
struct sigaction {
    void (*sa_handler)(int);
    void (*sa_sigaction)(int, siginfo_t *, void *);
    sigset_t sa_mask;
    int sa_flags;
    void (*sa_restorer)(void);
}
```

Рассмотрим модифицированную программу, использующую функции `raise` и `sigaction`:

```

1  #include <unistd.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <signal.h>
5  #include <string.h>
6  #include <syslog.h>
7  #include <sys/mman.h>
8
9  void signalArythmeticHandler(const int signalCode) {
10     const char* message = "Arythmetic exception ocurred! Code: 0x";
11     printf("%s%x\n", message, signalCode);
12
13     openlog("task3", 0, LOG_USER);
14     syslog(LOG_NOTICE, "%s%x\n", message, signalCode);
15     closelog();
16
17     exit(EXIT_FAILURE);

```


[illegible]

Результат аналогичен предыдущей программе, за исключением системного вызова `tgkill` на строке 28, которую `raise` использует для генерации исключения.

3.1.4 Обработка вложенных исключений ненадежными сигналами

Особенность ненадежных сигналов состоит в том, что если во время обработки такого сигнала возникнет еще один, то он обработан не будет. Именно из-за этого недостатка ненадежные сигналы не рекомендуется использовать в реальных программах.

Проиллюстрируем работу обработчика вложенных исключений, работающего на основе ненадежных сигналов:

```

1 #include <unistd.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <signal.h>
5 #include <string.h>
6 #include <syslog.h>
7
8 void signalArithmeticHandler(const int signalCode) {
9     const char* message = "Arithmetic exception occurred! Code: 0x";
10    printf("%s%x\n", message, signalCode);
11    raise(SIGBUS);
12
13    printf("Unreachable code, signals not in queue.\n");
14    exit(EXIT_FAILURE);
15 }
16
17 void signalBusHandler(const int signalCode) {
18     const char* message = "Bus exception occurred! Code: 0x";
19    printf("%s%x\n", message, signalCode);
20    exit(EXIT_FAILURE);
21 }
22
23 int main(int argc, char *argv[]) {
24     struct sigaction arithmeticAction, busAction;
25
26     arithmeticAction.sa_handler = signalArithmeticHandler;
27     sigemptyset(&arithmeticAction.sa_mask);
28     arithmeticAction.sa_flags;
29
30     busAction.sa_handler = signalBusHandler;
31     sigemptyset(&busAction.sa_mask);
32     busAction.sa_flags;
33
34     sigaction(SIGFPE, &arithmeticAction, NULL);
35     sigaction(SIGBUS, &busAction, NULL);
36
37     raise(SIGFPE);
38     return 0x0;
39 }

```

В функции `main` был вызван сигнал `SIGFPE`, который инициировал обработчик `signalArithmeticHandler`. Внутри обработчика арифметического исключения был вызван сигнал `SIGBUS`, который сразу же инициировал обработчик `signalBusHandler`, который завершил процесс. Данный пример иллюстрирует принцип работы ненадежных сигналов: они работают полностью асинхронно и не организуются в очередь.

```
1 | lorismelik@lorismelik-Aspire-Z5700:~/ $ gcc task4.c -o task4
2 | lorismelik@lorismelik-Aspire-Z5700:~/ $ ./task4
3 | Arithmetic exception occurred! Code: 0x8
4 | Bus exception occurred! Code: 0x7
```

Можно заметить, что вывод фразы "Unreachable code, signals not in queue." не произошел, потому что приложение завершилось до этого.

Рассмотрим системные вызовы данного процесса:

```
1  lorismelik@lorismelik-Aspire-Z5700:~/ $ ./task4
2  Ayrthmetic exception occured! Code: 0x8
3  Bus exception occured! Code: 0x7
4
5  lorismelik@lorismelik-Aspire-Z5700:~/ $ strace ./task4.o
6  execve("./task4.o", ["/task4.o"], 0x7ffedc85bd00 /* 57 vars */) = 0
7  brk(NULL) = 0x55e6f3e4d000
8  access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory)
9  access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
10 openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
11 fstat(3, {st_mode=S_IFREG|0644, st_size=88178, ...}) = 0
12 mmap(NULL, 88178, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f6859849000
13 close(3) = 0
14 access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory)
15 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
16 read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\260\34\2\0\0\0\0"... , 832) = 832
17 fstat(3, {st_mode=S_IFREG|0755, st_size=2030544, ...}) = 0
18 mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f6859847000
19 mmap(NULL, 4131552, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f6859247000
20 mprotect(0x7f685942e000, 2097152, PROT_NONE) = 0
21 mmap(0x7f685962e000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7000) = 0
   x7f685962e000
22 mmap(0x7f6859634000, 15072, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0
   x7f6859634000
23 close(3) = 0
24 arch_prctl(ARCH_SET_FS, 0x7f68598484c0) = 0
25 mprotect(0x7f685962e000, 16384, PROT_READ) = 0
26 mprotect(0x55e6f34c4000, 4096, PROT_READ) = 0
27 mprotect(0x7f685985f000, 4096, PROT_READ) = 0
28 munmap(0x7f6859849000, 88178) = 0
29 rt_sigaction(SIGFPE, {sa_handler=0x55e6f32c481a, sa_mask=[], sa_flags=SA_RESTORER, sa_restorer=0
   x7f6859285f20}, NULL, 8) = 0
30 rt_sigaction(SIGBUS, {sa_handler=0x55e6f32c486b, sa_mask=[], sa_flags=SA_RESTORER|SA_RESTART|SA_INTERRUPT
   |SA_NODEFER|SA_RESETHAND|0x32c4710, sa_restorer=0x7f6859285f20}, NULL, 8) = 0
31 rt_sigprocmask(SIG_BLOCK, ~[RTMIN RT_1], [], 8) = 0
32 getpid() = 22277
33 gettid() = 22277
34 tgkill(22277, 22277, SIGFPE) = 0
35 rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
36 --- SIGFPE {si_signo=SIGFPE, si_code=SI_TKILL, si_pid=22277, si_uid=1000} ---
37 fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 1), ...}) = 0
38 brk(NULL) = 0x55e6f3e4d000
39 brk(0x55e6f3e6e000) = 0x55e6f3e6e000
40 write(1, "Ayrthmetic exception occured! Co"... , 40Ayrthmetic exception occured! Code: 0x8
41 ) = 40
42 rt_sigprocmask(SIG_BLOCK, ~[RTMIN RT_1], [FPE], 8) = 0
43 getpid() = 22277
44 gettid() = 22277
45 tgkill(22277, 22277, SIGBUS) = 0
46 rt_sigprocmask(SIG_SETMASK, [FPE], NULL, 8) = 0
47 --- SIGBUS {si_signo=SIGBUS, si_code=SI_TKILL, si_pid=22277, si_uid=1000} ---
48 write(1, "Bus exception occured! Code: 0x7"... , 33Bus exception occured! Code: 0x7
49 ) = 33
50 exit_group(1) = ?
51 +++ exited with 1 +++
```

Действительно, оба сигнала были порождены асинхронно, о чем свидетельствует вывод утилиты strace. Как только из первого обработчика был инициирован сигнал, сразу же был вызван второй обработчик.

3.1.5 Обработка вложенных исключений надежными сигналами

В отличие от ненадежных сигналов, надежные сигналы ставятся в очередь и не теряются, если один из сигналов находится в обработке. Надежные сигналы организуются при помощи структуры `sigset_t`, а также следующих функций:

```
int sigemptyset(sigset_t *sig_m);
int sigaddset(sigset_t *sig_m, int signr);
int sigprocmask(int mode, const sigset_t *sig_m, sigset_t *alt_sig_m);
```

С помощью этой функции можно блокировать процесс до тех пор, пока не придет нужный сигнал. Рассмотрим программу для обработки вложенных исключений, организованную посредством надежных сигналов:

```
1  #include <unistd.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <signal.h>
5  #include <string.h>
6  #include <syslog.h>
7
8  void signalAyrthmeticHandler(const int signalCode) {
9      printf("Ayrthmetic exception occured! Code: 0x%x\n", signalCode);
```

```

10
11     sigset_t sigset;
12     sigemptyset(&sigset);
13     sigaddset(&sigset, SIGSEGV);
14     sigprocmask(SIG_SETMASK, &sigset, NULL);
15
16     raise(SIGSEGV);
17
18     printf("Next signal in queue.\n");
19 }
20
21 void signalBusHandler(const int signalCode) {
22     printf("Bus exception occurred! Code: 0x%x\n", signalCode);
23     exit(EXIT_FAILURE);
24 }
25
26 int main(int argc, char *argv[]) {
27     if (signal(SIGFPE, signalArithmeticHandler) == SIG_ERR)
28         return 0x1;
29
30     if (signal(SIGBUS, signalBusHandler) == SIG_ERR)
31         return 0x2;
32
33     raise(SIGFPE);
34     return 0x0;
35 }

```

```
1 lorismelik@lorismelik-Aspire-Z5700:~/ $ ./task5
2 Arthmetic exception occured! Code: 0x8
3 Next signal in queue.
4 Bus exception occured! Code: 0x7
```

```

1  lorismelik@lorismelik-Aspire-Z5700:~/task5
2  Arithmetic exception occurred! Code: 0x8
3  Next signal in queue.
4  Memory exception occurred! Code: 0x7
5
6  lorismelik@lorismelik-Aspire-Z5700:~/task5.o
7  execve("./task5.o", ["/task5.o"], 0x7fff2aa7a1e0 /* 57 vars */) = 0
8  brk(NULL) = 0x55e76de47000
9  access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory)
10 access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
11 openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
12 fstat(3, {st_mode=S_IFREG|0644, st_size=88178, ...}) = 0
13 mmap(NULL, 88178, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fa691c45000
14 close(3) = 0
15 access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory)
16 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
17 read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\3\0\0\0\1\0\0\0\260\34\2\0\0\0\0\0"... , 832) = 832
18 fstat(3, {st_mode=S_IFREG|0755, st_size=2030544, ...}) = 0
19 mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fa691c43000
20 mmap(NULL, 4131552, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fa691643000
21 mprotect(0x7fa69182a000, 2097152, PROT_NONE) = 0
22 mmap(0x7fa691a2a000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7000) = 0
   x7fa691a2a000
23 mmap(0x7fa691a30000, 15072, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0
   x7fa691a30000
24 close(3) = 0
25 arch_prctl(ARCH_SET_FS, 0x7fa691c444c0) = 0
26 mprotect(0x7fa691a2a000, 16384, PROT_READ) = 0
27 mprotect(0x55e76cc39000, 4096, PROT_READ) = 0
28 mprotect(0x7fa691c5b000, 4096, PROT_READ) = 0
29 munmap(0x7fa691c45000, 88178) = 0
30 rt_sigaction(SIGFPE, {sa_handler=0x55e76ca398ba, sa_mask=[FPE], sa_flags=SA_RESTORER|SA_RESTART,
   sa_restorer=0x7fa691681f20}, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=0}, 8) = 0
31 rt_sigaction(SIGBUS, {sa_handler=0x55e76ca3995c, sa_mask=[BUS], sa_flags=SA_RESTORER|SA_RESTART,
   sa_restorer=0x7fa691681f20}, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=0}, 8) = 0
32 rt_sigprocmask(SIG_BLOCK, ~[RTMIN RT_1], [], 8) = 0
33 getpid() = 22314
34 gettid() = 22314
35 tgkill(22314, 22314, SIGFPE) = 0
36 rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
37 --- SIGFPE {si_signo=SIGFPE, si_code=SI_TKILL, si_pid=22314, si_uid=1000} ---
38 fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 1), ...}) = 0
39 brk(NULL) = 0x55e76de47000
40 brk(0x55e76de68000) = 0x55e76de68000
41 write(1, "Arithmetic exception occurred! Co"... , 40Arithmetic exception occurred! Code: 0x8
42 ) = 40
43 rt_sigprocmask(SIG_SETMASK, [BUS], NULL, 8) = 0
44 rt_sigprocmask(SIG_BLOCK, ~[RTMIN RT_1], [BUS], 8) = 0
45 getpid() = 22314
46 gettid() = 22314
47 tgkill(22314, 22314, SIGBUS) = 0

```

```

48 rt_sigprocmask(SIG_SETMASK, [BUS], NULL, 8) = 0
49 write(1, "Next signal in queue.\n", 22Next signal in queue.
50 ) = 22
51 rt_sigreturn({mask=[]}) = 0
52 --- SIGBUS {si_signo=SIGBUS, si_code=SI_TKILL, si_pid=22314, si_uid=1000} ---
53 write(1, "Bus exception occurred! Code: 0x7"..., 33Bus exception occurred! Code: 0x7
54 ) = 33
55 exit_group(1) = ?
56 +++ exited with 1 +++

```

Можно заметить, что был вызван системный вызов `rt_sigprocmask` (строки 25, 29, 37, 38, 42), который устанавливает маску сигналов, а также вызов `rt_sigreturn` (строка 45).

3.2 Обработка программных исключений C++

В языке C++ была реализована программная обработка исключений. Конструкции языка `try`, `catch`, `finally` для программных исключений работают аналогично SEH в Windows.

3.2.1 Генерация и обработка исключений

Рассмотрим программу, вызывающую и обрабатывающую исключение `out_of_range` средствами C++:

```

1 #include <stdexcept>
2 #include <iostream>
3
4 int main(int argc, char *argv[]) {
5     try {
6         throw std::out_of_range("Some C++ exception.");
7     } catch (const std::out_of_range& exception) {
8         std::cout << exception.what() << std::endl;
9     }
10
11     return 0x0;
12 }

```

Результат работы программы

```

1 lorismelik@lorismelik-Aspire-Z5700:~/ $ g++ task6.cpp -o task6
2 lorismelik@lorismelik-Aspire-Z5700:~/ $ ./task6
3 Some C++ exception.

```

С помощью утилиты `ltrace` рассмотрим вызовы процесса к стандартной библиотеке C++:

```

1 lorismelik@lorismelik-Aspire-Z5700:~/ $ ltrace -e "*exception***catch***wind*" ./task6
2 task6->__cxa_allocate_exception(16, 0x7ffd92c69028, 0x7ffd92c69038, 160)
   = 0x5650e118eef0
3 libstdc++.so.6->__cxa_init_primary_exception(0x5650e118eef0, 0x5650e03eed28, 0x7f6d34bc5fc0, 0
   x5650e118ef28) = 0x5650e118ee70
4 libstdc++.so.6->_Unwind_RaiseException(0x5650e118eed0, 0x5650e03eed28, 0x7f6d34bc5fc0, 0x5650e118ef28 <
   unfinished ...>
5 libgcc_s.so.1->_Unwind_Find_FDE(0x7f6d3491683d, 0x7ffd92c68bd8, 0x7f6d3491683e, 0)
   = 0x7f6d3491b9c8
6 libgcc_s.so.1->_Unwind_Find_FDE(0x7f6d34bb0c66, 0x7ffd92c68cc8, 0x7f6d34bb0c67, 0)
   = 0x7f6d34c6b510
7 libgcc_s.so.1->_Unwind_Find_FDE(0x5650e01edcb2, 0x7ffd92c68cc8, 0x5650e01edcb3, 0)
   = 0x5650e01edf40
8 libstdc++.so.6->_Unwind_GetLanguageSpecificData(0x7ffd92c68c20, 1, 0x474e5543432b2b00, 0x5650e118eed0)
   = 0x5650e01ee010
9 libstdc++.so.6->_Unwind_GetRegionStart(0x7ffd92c68c20, 0x5650e01ee010, 0x7ffd92c68ab0, 0x5650e118eed0)
   = 0x5650e01edc6a
10 libstdc++.so.6->_Unwind_GetIPInfo(0x7ffd92c68c20, 0x7ffd92c68a8c, 23, 7)
   = 0x5650e01edcb3
11 libgcc_s.so.1->_Unwind_GetCFA(0x7ffd92c68c20, 0x5650e01ee010, 0x5650e118eef0, 0)
   = 0x7ffd92c68f00
12 libgcc_s.so.1->_Unwind_Find_FDE(0x7f6d34bb0c66, 0x7ffd92c68cc8, 0x7f6d34bb0c67, 0)
   = 0x7f6d34c6b510
13 libgcc_s.so.1->_Unwind_GetCFA(0x7ffd92c68c20, 4, 4, 7)
   = 0x7ffd92c68ee0
14 libgcc_s.so.1->_Unwind_Find_FDE(0x5650e01edcb2, 0x7ffd92c68cc8, 0x5650e01edcb3, 0)
   = 0x5650e01edf40
15 libgcc_s.so.1->_Unwind_GetCFA(0x7ffd92c68c20, 5, 0x5650e01edd47, 0)
   = 0x7ffd92c68f00
16 libstdc++.so.6->_Unwind_SetGR(0x7ffd92c68c20, 0, 0x5650e118eed0, 0x5650e118eed0)
   = 8
17 libstdc++.so.6->_Unwind_SetGR(0x7ffd92c68c20, 1, 1, 0x7ffd92c68e98)
   = 8
18 libstdc++.so.6->_Unwind_SetIP(0x7ffd92c68c20, 0x5650e01edcc9, 1, 0x7ffd92c68ea0)
   = 8
19 task6->__cxa_begin_catch(0x5650e118eed0, 0x5650e01edcc9, 1, 0x7ffd92c68ef8)
   = 0x5650e118eef0
20 Some C++ exception.
21 task6->__cxa_end_catch(0x7f6d34901760, 0x7f6d349028c0, 0, 2880 <unfinished ...>
22 libstdc++.so.6->_Unwind_DeleteException(0x5650e118eed0, 0x7f6d349028c0, 0, 2880 <unfinished ...>
23 libstdc++.so.6->_ZNSt9exceptionD2Ev(0x5650e118eef0, 0x5650e117d020, 0x5650e117d010, 1)
   = 0

```

```

24 | libstdc++.so.6->__cxa_free_exception(0x5650e118eef0, 0x5650e117d020, 0x5650e117d010, 1)
    |                                     = 0
25 | <... _Unwind_DeleteException resumed> )
    |                                     = 0
26 | <... __cxa_end_catch resumed> )
    |                                     = 0
27 | +++ exited (status 0) +++
    |                                     = 0

```

В первую очередь была вызвана библиотечная функция `Unwind_RaiseException`, которая выбрасывает исключение. Далее происходит длительный процесс раскрутки стека, о чем свидетельствуют вызовы с префиксом `Unwind`. Затем вызывается обработчик блока `catch` функцией `sxa_begin_catch` и завершается функцией `sxa_end_catch`. После чего приложение нормально завершается.

3.2.2 Обработка вложенных исключений

В C++ обработка вложенных исключений выглядит просто и логично. При генерации исключения во внутреннем блоке ищется обработчик catch в направлении от внутренних к внешним. Рассмотрим программу, иллюстрирующую работу вложенных исключений:

```
1 #include <stdexcept>
2 #include <iostream>
3
4 int main(int argc, char *argv[]) {
5     try {
6         try {
7             try {
8                 throw std::out_of_range("");
9             } catch (const std::length_error& exception) {
10                 // Unreachable code.
11                 std::cout << "Length error exception." << std::endl;
12             }
13         } catch (const std::out_of_range& exception) {
14             std::cout << "Out of range exception." << std::endl;
15             throw std::overflow_error("");
16         }
17     } catch (const std::overflow_error& exception) {
18         std::cout << "Overflow exception." << std::endl;
19     }
20
21     return 0x0;
22 }
```

Результат работы программы

```
1 lorismelik@lorismelik-Aspire-Z5700:~/ $ g++ task7.cpp -o task7
2 lorismelik@lorismelik-Aspire-Z5700:~/ $ ./task7
3 Out of range exception.
4 Overflow exception.
```

С помощью утилиты ltrace рассмотрим вызовы процесса к стандартной библиотеке C++:

```
1 lorismelik@lorismelik-Aspire-Z5700:~/ $ ltrace -e "exception***catch***wind*@libstdc*.so.6+task7" ./task7
2 task7->__cxa_allocate_exception(16, 0x7ffe9047ed08, 0x7ffe9047ed18, 160)
3     = 0x555e46a94ef0
4 libstdc++.so.6->__cxa_init_primary_exception(0x555e46a94ef0, 0x555e46932d18, 0x7f4b12042fc0, 117)
5     = 0x555e46a94e70
6 libstdc++.so.6->_Unwind_RaiseException(0x555e46a94ed0, 0x555e46932d18, 0x7f4b12042fc0, 117 <unfinished
7     ...>
8 libstdc++.so.6->_Unwind_GetLanguageSpecificData(0x7ffe9047e8f0, 1, 0x474e5543432b2b00, 0x555e46a94ed0)
9     = 0x555e46732280
10 libstdc++.so.6->_Unwind_GetRegionStart(0x7ffe9047e8f0, 0x555e46732280, 0x7ffe9047e780, 0x555e46a94ed0)
11     = 0x555e46731dca
12 libstdc++.so.6->_Unwind_GetIPInfo(0x7ffe9047e8f0, 0x7ffe9047e75c, 48, 7)
13     = 0x555e46731e13
14 libstdc++.so.6->_Unwind_SetGR(0x7ffe9047e8f0, 0, 0x555e46a94ed0, 0x555e46a94ed0)
15     = 8
16 libstdc++.so.6->_Unwind_SetGR(0x7ffe9047e8f0, 1, 2, 0x7ffe9047eb68)
17     = 8
18 libstdc++.so.6->_Unwind_SetIP(0x7ffe9047e8f0, 0x555e46731e33, 2, 0x7ffe9047eb70)
19     = 8
20 task7->__cxa_begin_catch(0x555e46a94ed0, 0x555e46731e33, 2, 2)
21     = 0x555e46a94ef0
22 Out of range exception.
23 task7->__cxa_allocate_exception(16, 0x7f4b11d7f8c0, 0, 2880)
24     = 0x555e46a953a0
25 libstdc++.so.6->__cxa_init_primary_exception(0x555e46a953a0, 0x555e46932d00, 0x7f4b120430f0, 117)
26     = 0x555e46a95320
27 libstdc++.so.6->_Unwind_RaiseException(0x555e46a95380, 0x555e46932d00, 0x7f4b120430f0, 117 <unfinished
28     ...>
29 libstdc++.so.6->_Unwind_GetLanguageSpecificData(0x7ffe9047e8f0, 1, 0x474e5543432b2b00, 0x555e46a95380)
30     = 0x555e46732280
31 libstdc++.so.6->_Unwind_GetRegionStart(0x7ffe9047e8f0, 0x555e46732280, 0x7ffe9047e780, 0x555e46a95380)
32     = 0x555e46731dca
33 libstdc++.so.6->_Unwind_GetIPInfo(0x7ffe9047e8f0, 0x7ffe9047e75c, 48, 7)
34     = 0x555e46731f01
35 libstdc++.so.6->_Unwind_SetGR(0x7ffe9047e8f0, 0, 0x555e46a95380, 0x555e46a95380)
36     = 8
37 libstdc++.so.6->_Unwind_SetGR(0x7ffe9047e8f0, 1, 1, 0x7ffe9047eb68)
38     = 8
39 libstdc++.so.6->_Unwind_SetIP(0x7ffe9047e8f0, 0x555e46731f14, 1, 0x7ffe9047eb70)
40     = 8
41 task7->__cxa_end_catch(0x7ffe9047ebd0, 0x555e46731f14, 1, 0x7ffe9047ebc8 <unfinished ...>
42 libstdc++.so.6->_Unwind_DeleteException(0x555e46a94ed0, 0x555e46731f14, 0, 0x7ffe9047ebc8 <unfinished
43     ...>
44 libstdc++.so.6->_ZNSt9exceptionD2Ev(0x555e46a94ef0, 0x555e46a94ed0, 0, 0x7ffe9047ebc8)
45     = 0x7f4b12322d78
46 libstdc++.so.6->__cxa_free_exception(0x555e46a94ef0, 0x555e46a94ed0, 0, 0x7ffe9047ebc8)
47     = 0
48 <... _Unwind_DeleteException resumed> )
49     = 0
50 <... __cxa_end_catch resumed> )
51     = 0
```

```

28 task7->__cxa_begin_catch(0x555e46a95380, 0x555e46a83050, 1, 1)
29                                     = 0x555e46a953a0
29 Overflow exception.
30 task7->__cxa_end_catch(0x7f4b11d7e760, 0x7f4b11d7f8c0, 0, 2880 <unfinished ...>
31 libstdc++.so.6->_Unwind_DeleteException(0x555e46a95380, 0x7f4b11d7f8c0, 0, 2880 <unfinished ...>
32 libstdc++.so.6->_ZNSt9exceptionD2Ev(0x555e46a953a0, 0x555e46a95380, 0, 2880)
33                                     = 0x7f4b12322d78
33 libstdc++.so.6->__cxa_free_exception(0x555e46a953a0, 0x555e46a95380, 0, 2880)
34                                     = 0
34 <... _Unwind_DeleteException resumed> )
35                                     = 0
35 <... __cxa_end_catch resumed> )
36                                     = 0
36 +++ exited (status 0) +++

```

Можно заметить, что после каждого из двух выброшенных исключений произошла раскрутка стека и был найден соответствующий обработчик.

3.2.3 Выход из охраняемого кода инструкцией goto

Рассмотрим программу, которая выходит из охраняемого кода инструкцией goto:

```

1  #include <stdexcept>
2  #include <iostream>
3
4  int main(int argc, char *argv[]) {
5      try {
6          try {
7              try {
8                  goto out;
9              } catch (const std::length_error& exception) {
10                 std::cout << "Length error exception." << std::endl;
11             }
12             } catch (const std::out_of_range& exception) {
13                 std::cout << "Out of range exception." << std::endl;
14             }
15             } catch (const std::overflow_error& exception) {
16                 std::cout << "Overflow exception." << std::endl;
17             }
18
19         return 0x0;
20
21     out:
22         std::cout << "Exit with goto instruction." << std::endl;
23         return 0x0;
24 }

```

Результат работы программы свидетельствует о том, что выход из блока был успешно осуществлен:

```

1  lorismelik@lorismelik-Aspire-Z5700:~/ $ g++ task8.cpp -o task8
2  lorismelik@lorismelik-Aspire-Z5700:~/ $ ./task8
3  Exit with goto instruction.

```

С помощью утилиты ltrace рассмотрим вызовы процесса к стандартной библиотеке C++:

```

1  lorismelik@lorismelik-Aspire-Z5700:~/ $ ltrace -e "*exception***catch***wind*libstdc*.so.6+task8" ./task8
2  Exit with goto instruction.
3  +++ exited (status 0) +++

```

Можно заметить, что раскрутки стека осуществлено не было.

4 Вывод

В ходе выполнения данной работы был проведен анализ таких механизмов обработки аппаратных и программных исключений, доступных в unix-подобных системах, как системные сигналы и стандартные средства языка C++.

Метод обработки исключений посредством сигналов в Unix-подобных системах является намного менее удобным для программиста, чем SEH в Windows. Кроме того всегда необходимо задумываться о ненадежности сигналов или организовывать надежные. Систему вложенных исключений посредством сигналов также очень неудобно организовывать.

Программные исключения C++ являются полноценным универсальным средством для обработки программных исключений. Программисту предоставляется множество возможностей для разработки собственной системы исключений, для вложенных исключений, для очистки ресурсов (finally). Однако, исключения в C++ не умеют обрабатывать аппаратные прерывания, а также работают не так быстро из-за раскрутки стека.