Санкт-Петербургский Политехнический Университет Петра Великого
Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

Отчёт по лабораторной работе №1 часть 2
Курс: Системное программирование
Тема: Анализ обработки исключений в Linux

Выполнил студент группы 13541/3 _____ Д.В. Круминьш
(подпись)

Преподаватель _____ Е.В. Душутина
(подпись)

Санкт-Петербург
2017 г.

Содержание

Постановка задачи

Необходимо провести анализ анализ механизмов обработки аппаратных и программных исключений, доступных в операционной системе Linux. Выполнить задание для исключения:

- ошибка деления на ноль;
- нарушение доступа к памяти.

Подготовка к выполнению работы

2.1 Сведения о системе

Все опыты проводятся на виртуальной системе(64 битной), сведения о которой приведены в листинге 3.1.

```
root@kali:~/Desktop/testFolder# cat /etc/*release*
DISTRIB_ID=Kali
DISTRIB_RELEASE=kali-rolling
DISTRIB_CODENAME=kali-rolling
DISTRIB_DESCRIPTION="Kali GNU/Linux Rolling"
PRETTY_NAME="Kali GNU/Linux Rolling"
NAME="Kali GNU/Linux"
ID=kali
VERSION="2017.2"
VERSION_ID="2017.2"
ID_LIKE=debian
ANSI_COLOR="1;31"
HOME_URL="http://www.kali.org/"
SUPPORT_URL="http://forums.kali.org/"
BUG_REPORT_URL="http://bugs.kali.org/"

root@kali:~# uname -r
4.12.0-kali1-amd64

root@kali:~# lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                4
On-line CPU(s) list:   0-3
Thread(s) per core:    1
Core(s) per socket:    2
Socket(s):             2
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model name:            Intel(R) Core(TM) i5-7300HQ CPU @ 2.50GHz
Stepping:              9
CPU MHz:               2495.998
Hypervisor vendor:     VMware
Virtualization type:   full
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              6144K
NUMA node0 CPU(s):     0-3
```

Листинг 2.1: Информация о системе

Работа происходила с использованием:

- Текстового редактора Sublime text(версии 3.0, сборка 3143);
- Компилятора gcc(версия Debian 7.2.0-4);
- Отладчика GNU Debugger(версия Debian 7.12-6).

Выполнение работы

Исключение – это аномальное поведение во время выполнения, в случае отсутствия обработки исключений их возникновение приведет к немедленному прекращению выполнения программы. В Unix-подобных системах, в отличие от систем семейства Windows, не имеется единого средства для обработки аппаратных и программных исключений, как структурированная обработка исключений (SEH). Для фиксации и обработки аппаратных исключений в *nix системах предусмотрено многоцелевое средство межпроцессного взаимодействия – сигналы. Сигнал представляет собой асинхронное уведомление процесса о возникновении некоторого события. Модель сигналов используемая в UNIX, значительно отличается от SEH, сигналы могут быть пропущены или игнорированы, и логика их работы иная.

3.1 Существующие сигналы

Количество сигналов в системе может варьироваться в зависимости от реализации системы. С помощью команды **kill -l** был выведен список сигналов.

```
root@kali:~# kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

Листинг 3.1: Информация о системе

Далее приведена таблица с сигналами, их действиями, описанием и типом. (Информация взята из источников [1] и [2])

Название	Код	Действие по умолчанию	Описание	Тип
SIGHUP	1	Завершение	Закрытие терминала	Уведомление
SIGINT	2	Завершение	Сигнал прерывания (Ctrl-C) с терминала	Управление

SIGQUIT	3	Завершение с дампом памяти	Сигнал «Quit» с терминала (Ctrl-\)	Управление
SIGILL	4	Завершение с дампом памяти	Недопустимая инструкция процессора	Исключение
SIGTRAP	5	Завершение с дампом памяти	Ловушка трассировки или брейкпоинт	Отладка
SIGABRT	6	Завершение с дампом памяти	Сигнал посылаемый функцией abort()	Управление
SIGFPE	8	Завершение с дампом памяти	Ошибочная арифметическая операция	Исключение
SIGKILL	9	Завершение	Безусловное завершение	Управление
SIGBUS	10	Завершение с дампом памяти	Неправильное обращение в физическую память	Исключение
SIGSEGV	11	Завершение с дампом памяти	Нарушение при обращении в память	Исключение
SIGSYS	12	Завершение с дампом памяти	Неправильный системный вызов	Исключение
SIGPIPE	13	Завершение	Запись в разорванное соединение (пайп, сокет)	Уведомление
SIGALRM	14	Завершение	Сигнал истечения времени, заданного alarm()	Уведомление
SIGTERM	15	Завершение	Сигнал завершения (сигнал по умолчанию для утилиты kill)	Управление
SIGUSR1	16	Завершение	Пользовательский сигнал № 1	Пользовательский
SIGUSR2	17	Завершение	Пользовательский сигнал № 2	Пользовательский
SIGCHLD	18	Игнорируется	Дочерний процесс завершен или остановлен	Уведомление
SIGTSTP	20	Остановка процесса	Сигнал остановки с терминала (Ctrl-Z).	Управление
SIGURG	21	Игнорируется	На сокете получены срочные данные	Уведомление

SIGPOLL	22	Завершение	Событие, отслеживаемое poll()	Уведомление
SIGSTOP	23	Остановка процесса	Остановка выполнения процесса	Управление
SIGCONT	25	Продолжить выполнение	Продолжить выполнение ранее остановленного процесса	Управление
SIGTTIN	26	Остановка процесса	Попытка чтения с терминала фоновым процессом	Управление
SIGTTOU	27	Остановка процесса	Попытка записи на терминал фоновым процессом	Управление
SIGVTALRM	28	Завершение	Истечение «виртуального таймера»	Уведомление
SIGPROF	29	Завершение	Истечение таймера профилирования	Отладка
SIGXCPU	30	Завершение с дампом памяти	Процесс превысил лимит процессорного времени	Исключение
SIGXFSZ	31	Завершение с дампом памяти	Процесс превысил допустимый размер файла	Исключение

Сигналы с 32 по 64 это сигналы реального времени. В отличие от обычных сигналов, сигналы реального времени имеют очередь, при использовании специальных функций для отправки сигнала (sigqueue — передача сигнала или данных процессу) могут передавать информацию (целое число или указатель), доставляются в том же порядке, в котором были отправлены.

В данной работе рассматриваются сигналы **SIGFPE(11)** и **SIGSEGV(8)** (деление на ноль и ошибочный доступ к памяти).

3.2 Генерация сигналов

3.2.1 Путем некорректных операций

Далее в листинге 3.2 приведен пример генерации соответствующих сигналов-исключений в результате некорректных операций, описанных в коде программы.

```
1 #include <unistd.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <signal.h>
5 #include <cstring>
6
7 void GenerateFPException() {
8     int a = 1;
```

```
9       int b = a / (a - 1);
10  }
11  void GenerateSEGException() {
12      int* a = NULL;
13      *a = 1;
14  }
15
16  int main(int argc, char *argv[]){
17      if (argc == 2) {
18          if (strcmp(argv[1], "-FP") == 0)
19              GenerateFPException();
20          else if (strcmp(argv[1], "-SEG") == 0) {
21              GenerateSEGException();
22          }
23      }
24   return 0;
25  }
```

Листинг 3.2: p1.cpp

Скомпилируем и запустим программу, с необходимыми ключами.

```
1  root@kali:~/Desktop/linuxEx/p1# ./p1.o -FP
2  Floating point exception
3  root@kali:~/Desktop/linuxEx/p1# ./p1.o -SEG
4  Segmentation fault
```

Листинг 3.3: выполнение p1.cpp

Как и ожидалось, система отреагировала на исключения, написав в консоль соответствующее сообщение. Проверим лог системы на наличие этих исключений.

```
1  root@kali:~/Desktop/linuxEx/p1# less /var/log/messages | grep p1
2  Dec  2 09:59:23 kali kernel: [   18.591646] RAPL PMU: hw unit of domain pp1-gpu
       ↪ 2^-0 Joules
3  Dec  3 11:48:24 kali kernel: [11954.248859] traps: p1.o[3365] trap divide error
       ↪ ip:55ea9c54565f sp:7ffdb8ad42a0 error:0 in p1.o[55ea9c545000+1000]
4  Dec  3 11:54:40 kali kernel: [12330.290723] p1.o[3381]: segfault at 0 ip
       ↪ 000055655cee0677 sp 00007ffe0fe26c70 error 6 in p1.o[55655cee0000+1000]
```

Листинг 3.4: лог системы

В системном логе также оказались записи о данных исключениях.

```
1  root@kali:~/Desktop/linuxEx/p1# gdb p1.o
2  (gdb) run -FP
3  Starting program: /root/Desktop/linuxEx/p1/p1.o -FP
4
5  Program received signal SIGFPE, Arithmetic exception.
6  0x000055555555465f in GenerateFPException() ()
7  (gdb) run -SEG
8  The program being debugged has been started already.
9  Start it from the beginning? (y or n) y
10 Starting program: /root/Desktop/linuxEx/p1/p1.o -SEG
11
12 Program received signal SIGSEGV, Segmentation fault.
13 0x0000555555554677 in GenerateSEGException() ()
```

Листинг 3.5: лог gdb

При выполнении в отладчике, выведено больше информации, в частности указан какой сигнал(исключение) был получен.

3.2.2 Генерация сигнала через raise()

Для “искусственной” генерации сигналов определенного типа для целей отладки и тестирования может быть применена функция raise, имеющая следующий прототип [3]:

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

Листинг 3.6: Прототип raise

Далее представлена программа использующая данную функцию.

```
1  #include <unistd.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <signal.h>
5  #include <cstring>
6
7  void GenerateFPException() {
8      raise(8);
9  }
10 void GenerateSEGException() {
11     raise(11);
12 }
13
14 int main(int argc, char *argv[]){
15     if (argc == 2) {
16         if (strcmp(argv[1], "-FP") == 0)
17             GenerateFPException();
18         else if (strcmp(argv[1], "-SEG") == 0) {
19             GenerateSEGException();
20         }
21     }
22  return 0;
23 }
```

Листинг 3.7: p1_raise.cpp

Функция успешно вызвала требуемые сигналы(исключения).

```
1 ali:~/Desktop/linuxEx/p1# ./p1_raise.o -FP
2 Floating point exception
3 root@kali:~/Desktop/linuxEx/p1# ./p1_raise.o -SEG
4 Segmentation fault
```

Листинг 3.8: Запуск p1_raise.cpp

Стоит отметить что в данном случае в системный логе записей об ошибке не возникло, это произошло по причине того что необходимый сигнал был сразу сгенерирован вручную, а не посредством системы после ошибки по ходу выполнения программы.

3.3 Обработка сигналов

Существуют различные способы для обработки сигналов, рассмотрим применение **signal** и **sigaction**.

3.3.1 Функция signal

Функция[4] предназначена для обработки определенных видов сигналов.

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

Листинг 3.9: Прототип signal

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.

В листинге 3.10, представлена программа в которой имеются необходимые обработчики.

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <cstring>

void GenerateFPException() {
    int a = 1;
    int b = a / (a - 1);
}
void GenerateSEGException() {
    int* a = NULL;
    *a = 1;
}

void FP_handler(int signum){
    printf("Received signal FP %d\n", signum);
    exit(EXIT_FAILURE);
}

void SEG_handler(int signum){
    printf("Received signal SEG %d\n", signum);
    exit(EXIT_FAILURE);
}


int main(int argc, char *argv[]){
    if (signal(SIGFPE, FP_handler) == SIG_ERR)
        printf("Cant set SIGFP handler\n");
    if (signal(SIGSEGV, SEG_handler) == SIG_ERR)
        printf("Cant set SIGSEGV handler\n");

    if (argc == 2) {
        if (strcmp(argv[1], "-FP") == 0)
            GenerateFPException();
        else if (strcmp(argv[1], "-SEG") == 0) {
            GenerateSEGException();
        }
    }
 return 0;
}
```

Листинг 3.10: p2.cpp

Выполнение программы продемонстрировало корректность работы обработчиков, ожидаемые сигналы были успешно обработаны.

```
root@kali:~/Desktop/linuxEx/p1# gcc p2.cpp -o p2.o
```

```
root@kali:~/Desktop/linuxEx/p1# ./p2.o –FP
Received signal FP 8
root@kali:~/Desktop/linuxEx/p1# ./p2.o –SEG
Received signal SEG 11
```

Листинг 3.11: Запуск p2.cpp

Запустим программу в отладчике.

```
root@kali:~/Desktop/linuxEx/p1# gdb p2.o

(gdb) run –FP
Starting program: /root/Desktop/linuxEx/p1/p2.o –FP

Program received signal SIGFPE, Arithmetic exception.
0x000055555555477f in GenerateFPException() ()
(gdb) bt
#0  0x000055555555477f in GenerateFPException() ()
#1  0x000055555555487e in main ()
(gdb) s
Single stepping until exit from function _Z19GenerateFPExceptionv,
which has no line number information.
0x00005555555547a0 in FP_handler(int) ()
(gdb) bt
#0  0x00005555555547a0 in FP_handler(int) ()
#1  <signal handler called>
#2  0x000055555555477f in GenerateFPException() ()
#3  0x000055555555487e in main ()
(gdb) c
Continuing.
Received signal FP 8
[Inferior 1 (process 3622) exited with code 01]
```

Листинг 3.12: Отладка p2.cpp

В логе 3.12 строчках 6-7 вывелось сообщение о получении сигнала **SIGFPE**, в стеке вызовов на данный момент имеется лишь две функции(main и GenerateFPException). Выполнив команду s(step) был совершен переход в функцию перехватчик, о чем свидетельствует стек вызовов в строчках 16-19. Далее была выполнена команда c(continue) в следствии чего программа завершила свою работу.

3.3.2 Функция sigaction

Системный вызов sigaction()[5] используется для изменения выполняемого процессом действия при получении определённого сигнала.

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

Листинг 3.13: Прототип sigaction

Прототип имеет следующие параметры:

1. **signum** - номер перехватываемого сигнала;
2. **sigaction *act** - указатель на структуру, описывающую правила установки обработчика;

3. **sigaction *oldact** - принимает уже установленные правила обработчика сигнала.

Структура **sigaction** имеет следующий вид:

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

Листинг 3.14: Прототип sigaction

Структура содержит следующие поля:

1. **sa_handler** - указывает действие, которое должно быть связано с signum, то есть указатель на функцию обработчик;
2. **sa_sigaction** - если в sa_flags указан SA_SIGINFO, то sa_sigaction (вместо sa_handler) задаёт функцию обработки:
3. **sa_mask** - задание маски сигналов, которые должны блокироваться;
4. **sa_flags** - указание набора флагов, которые изменяют поведение сигнала;
5. **sa_restorer** - код батута(трамплина). Не предназначено для использования в приложении.

Далее приведен листинг, использующий данную функцию.

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <cstring>

void GenerateFPException() {
    int a = 1;
    int b = a / (a - 1);
}
void GenerateSEGException() {
    int* a = NULL;
    *a = 1;
}

void FP_handler(int signum){
    printf("Received signal FP %d\n", signum);
    exit(EXIT_FAILURE);
}

void SEG_handler(int signum){
    printf("Received signal SEG %d\n", signum);
    exit(EXIT_FAILURE);
}


int main(int argc, char *argv[]){
    struct sigaction FP_action, SEG_action;

    /* Инициализация структуры обработчиком */
```

```
31      FP_action.sa_handler = FP_handler;
32      sigemptyset (&FP_action.sa_mask);
33      FP_action.sa_flags = 0;
34
35      SEG_action.sa_handler = SEG_handler;
36      sigemptyset (&SEG_action.sa_mask);
37      SEG_action.sa_flags = 0;
38
39      sigaction (SIGFPE, &FP_action, NULL);
40      sigaction (SIGSEGV, &SEG_action, NULL);
41
42      if (argc == 2) {
43          if (strcmp(argv[1], "-FP") == 0)
44              GenerateFPException();
45          else if (strcmp(argv[1], "-SEG") == 0) {
46              GenerateSEGException();
47          }
48      }
49   return 0;
50  }
```

Листинг 3.15: p3.cpp

Сигналы были также успешно обработаны.

```
1  root@kali:~/Desktop/linuxEx/p1# gcc p3.cpp -o p3.o
2  root@kali:~/Desktop/linuxEx/p1# ./p3.o -FP
3  Received signal FP 8
4  root@kali:~/Desktop/linuxEx/p1# ./p3.o -SEG
5  Received signal SEG 11
```

Листинг 3.16: Запуск p3.cpp

Вывод отладчика также был идентичен прошлому опыту.

```
1  root@kali:~/Desktop/linuxEx/p1# gdb p3.o
2
3  (gdb) run -FP
4  Starting program: /root/Desktop/linuxEx/p1/p3.o -FP
5
6  Program received signal SIGFPE, Arithmetic exception.
7  0x000055555555477f in GenerateFPException() ()
8  (gdb) bt
9  #0  0x000055555555477f in GenerateFPException() ()
10 #1  0x00005555555548c2 in main ()
11 (gdb) s
12 Single stepping until exit from function _Z19GenerateFPExceptionv,
13 which has no line number information.
14 0x00005555555547a0 in FP_handler(int) ()
15 (gdb) bt
16 #0  0x00005555555547a0 in FP_handler(int) ()
17 #1  <signal handler called>
18 #2  0x000055555555477f in GenerateFPException() ()
19 #3  0x00005555555548c2 in main ()
20 (gdb) finish
21 Run till exit from #0  0x00005555555547a0 in FP_handler(int) ()
22 Received signal FP 8
23 [Inferior 1 (process 3907) exited with code 01]
```

Листинг 3.17: Отладка p3.cpp

3.3.3 Расширенный вызов sigaction

У sigaction имеется возможность расширить возможности функции-обработчика, если использовать для этого параметр **sa_sigaction**. В частности это происходит за счет структуры **siginfo_t**[6] передаваемой в качестве параметра.

```
typedef struct {
    int si_signo;
    int si_code;
    union sigval si_value;
    int si_errno;
    pid_t si_pid;
    uid_t si_uid;
    void *si_addr;
    int si_status;
    int si_band;
} siginfo_t;
```

Листинг 3.18: Структура siginfo_t

Далее представлена программа, в которой использован расширенный обработчик.

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <cstring>
#include <iostream>

using namespace std;

void GenerateFPException() {
    int a = 1;
    int b = a / (a - 1);
}
void GenerateSEGException() {
    int* a = NULL;
    *a = 1;
}

void SIGNAL_handler(int signum, siginfo_t *info, void *context){
    switch(signum){
        case SIGFPE:
            printf("Received signal FP %d\n", signum);
            break;
        case SIGSEGV:
            printf("Received signal SEG %d\n", signum);
            break;
    }
    cout << "Signal number: " << info->si_signo << endl;
    cout << "Errno: " << info->si_errno << endl;
    cout << "Signal code: " << info->si_code << endl;
    cout << "PID: " << info->si_pid << endl;
    cout << "UID: " << info->si_uid << endl;
    cout << "Status: " << info->si_status << endl;
    cout << "Signal address: " << info->si_addr << endl;
    cout << "Signal event: " << info->si_band << endl;
    cout << "File descriptor: " << info->si_fd << endl;
    exit(EXIT_FAILURE);
}

```

```
40 int main(int argc, char *argv[]){
41     struct sigaction NEW_action;
42
43 /* Инициализация структуры обработчиком */
44     sigemptyset(&NEW_action.sa_mask);
45     NEW_action.sa_flags = SA_SIGINFO;
46     NEW_action.sa_sigaction = &SIGNAL_handler;
47
48     if (argc == 2) {
49         if (strcmp(argv[1], "-FP") == 0 && sigaction(SIGFPE, &NEW_action, NULL)
    ↪  != -1)
50             GenerateFPException();
51         else if (strcmp(argv[1], "-SEG") == 0 && sigaction(SIGSEGV, &NEW_action
    ↪ , NULL) != -1) {
52             GenerateSEGException();
53         }
54     }
55  return 0;
56 }
```

Листинг 3.19: p4.cpp

Как и ожидалось, вывод стал более информативным.

```
1  root@kali:~/Desktop/linuxEx/p1# ./p4.o -FP
2  Received signal FP 8
3  Signal number: 8
4  Errno: 0
5  Signal code: 1
6  PID: 661326735
7  UID: 21961
8  Status: 479434334
9  Signal address: 0x55c9276b0b8f
10 Signal event: 94322438114191
11 File descriptor: 479434334
12 root@kali:~/Desktop/linuxEx/p1# ./p4.o -SEG
13 Received signal SEG 11
14 Signal number: 11
15 Errno: 0
16 Signal code: 1
17 PID: 0
18 UID: 0
19 Status: 914833304
20 Signal address: 0
21 Signal event: 0
22 File descriptor: 914833304
```

Листинг 3.20: Запуск p4.cpp

Рассмотрим работу в отладчике.

```
1  root@kali:~/Desktop/linuxEx/p1# gdb p4.o
2
3  (gdb) run -FP
4  Starting program: /root/Desktop/linuxEx/p1/p4.o -FP
5
6  Program received signal SIGFPE, Arithmetic exception.
7  0x0000555555554b8f in GenerateFPException() ()
8  (gdb) s
9  Single stepping until exit from function _Z19GenerateFPExceptionv,
10 which has no line number information.
11 0x0000555555554bb0 in SIGNAL_handler(int, siginfo_t*, void*) ()
```

```
(gdb) bt
#0  0x0000555555554bb0 in SIGNAL_handler(int, siginfo_t*, void*) ()
#1  <signal handler called>
#2  0x0000555555554b8f in GenerateFPException() ()
#3  0x0000555555554ecb in main ()
(gdb) finish
Run till exit from #0  0x0000555555554bb0 in SIGNAL_handler(int, siginfo_t*,
    ↪ void*) ()
Received signal FP 8
Signal number: 8
Errno: 0
Signal code: 1
PID: 1431653263
UID: 21845
Status: 479434334
Signal address: 0x555555554b8f
Signal event: 93824992234383
File descriptor: 479434334
[Inferior 1 (process 4104) exited with code 01]
```

Листинг 3.21: Отладка p4.cpp

Полученный адрес сигнала(исключения) из структуры **siginfo_t**, как и ожидалось совпадает с адресом из отладчика.

3.4 Вложенные исключения

Рассмотрим поведение программы, если в обработчике исключения вызвать еще одно исключение.

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <cstring>

void GenerateFPException() {
    int a = 1;
    int b = a / (a - 1);
}
void GenerateSEGException() {
    int* a = NULL;
    *a = 1;
}

void FP_handler(int signum){
    printf("Received signal FP %d\n", signum);
    GenerateSEGException();
    exit(EXIT_FAILURE);
}

void SEG_handler(int signum){
    printf("Received signal SEG %d\n", signum);
    exit(EXIT_FAILURE);
}


int main(int argc, char *argv[]){
    if (signal(SIGFPE, FP_handler) == SIG_ERR)
```

```
30          printf("Cant set SIGFP handler\n");
31      if (signal(SIGSEGV, SEG_handler) == SIG_ERR)
32          printf("Cant set SIGSEGV handler\n");
33
34      if (argc == 2) {
35          if (strcmp(argv[1], "-FP") == 0)
36              GenerateFPException();
37          else if (strcmp(argv[1], "-SEG") == 0) {
38              GenerateSEGException();
39          }
40      }
41   return 0;
42  }
```

Листинг 3.22: p5.cpp

```
1  root@kali:~/Desktop/linuxEx/p1# g++ p5.cpp -o p5.o
2  root@kali:~/Desktop/linuxEx/p1# ./p5.o -FP
3  Received signal FP 8
4  Received signal SEG 11
```

Листинг 3.23: Запуск p5.cpp

```
1  root@kali:~/Desktop/linuxEx/p1# gdb p5.o
2
3  (gdb) run -FP
4  Starting program: /root/Desktop/linuxEx/p1/p5.o -FP
5
6  Program received signal SIGFPE, Arithmetic exception.
7  0x000055555555479f in GenerateFPException() ()
8  (gdb) bt
9  #0  0x000055555555479f in GenerateFPException() ()
10 #1  0x00005555555548a3 in main ()
11 (gdb) s
12 Single stepping until exit from function _Z19GenerateFPExceptionv,
13 which has no line number information.
14 0x00005555555547c0 in FP_handler(int) ()
15 (gdb) bt
16 #0  0x00005555555547c0 in FP_handler(int) ()
17 #1  <signal handler called>
18 #2  0x000055555555479f in GenerateFPException() ()
19 #3  0x00005555555548a3 in main ()
20 (gdb) c
21 Continuing.
22 Received signal FP 8
23
24 Program received signal SIGSEGV, Segmentation fault.
25 0x00005555555547b7 in GenerateSEGException() ()
26 (gdb) bt
27 #0  0x00005555555547b7 in GenerateSEGException() ()
28 #1  0x00005555555547e6 in FP_handler(int) ()
29 #2  <signal handler called>
30 #3  0x000055555555479f in GenerateFPException() ()
31 #4  0x00005555555548a3 in main ()
32 (gdb) s
33 Single stepping until exit from function _Z20GenerateSEGExceptionv,
34 which has no line number information.
35 0x00005555555547f0 in SEG_handler(int) ()
36 (gdb) bt
37 #0  0x00005555555547f0 in SEG_handler(int) ()
38 #1  <signal handler called>
```

```
#2  0x00005555555547b7 in GenerateSEGException() ()
#3  0x00005555555547e6 in FP_handler(int) ()
#4  <signal handler called>
#5  0x000055555555479f in GenerateFPException() ()
#6  0x00005555555548a3 in main ()
(gdb) c
Continuing.
Received signal SEG 11
[Inferior 1 (process 4292) exited with code 01]
```

Листинг 3.24: Отладка p5.cpp

В ходе отладки было выявлено, что после генерации сигнала об ошибке в ходе арифметической операции был вызван соответствующий обработчик, который сгенерировал сигнал SIGSEGV. Далее, системой без какого-либо ожидания завершения предыдущего обработчика было передано управление обработчику сигнала SIGSEGV, который завершил работу программы, что отражает асинхронность работы сигналов.

Для решения этой проблемы можно использовать блокировку сигналов.

3.5 Блокировка сигналов

Для блокировки сигналов можно использовать функцию sigprocmask[7], которая позволяет поставить фильтр для блокировки сигналов.

```
int sigprocmask(int how, const sigset_t *set, sigset_t *oldset);
```

Листинг 3.25: прототип sigprocmask

Работа вызова зависит от значения how:

- **SIG_BLOCK** Набор блокируемых сигналов — объединение текущего набора и аргумента set.
- **SIG_UNBLOCK** Сигналы в set удаляются из списка блокируемых сигналов. Допускается разблокировать незаблокированные сигналы.
- **SIG_SETMASK** Набор блокируемых сигналов приравнивается к аргументу set.

Если значение oldset не равно NULL, то предыдущее значение маски сигналов записывается в oldset.

Если значение set равно NULL, то маска сигналов не изменяется (т.е., значение how игнорируется), но текущее значение маски сигналов всё же возвращается в oldset (если его значение не равно NULL).

Далле представлена программа, которая блокирует сигнал SEG(11).

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <cstring>

void GenerateFPException() {
    raise(8);
}
void GenerateSEGException() {
    raise(11);
}
```

```
13
14
15 void FP_handler(int signum){
16     sigset_t sigset;
17     sigemptyset(&sigset);
18     sigaddset(&sigset, SIGSEGV);
19     sigprocmask(SIG_SETMASK, &sigset, NULL);
20
21     printf("Received signal FP %d\n", signum);
22     GenerateSEGException();
23     printf("Finishing FP handler\n");
24     exit(EXIT_FAILURE);
25 }
26
27 void SEG_handler(int signum){
28     printf("Received signal SEG %d\n", signum);
29     printf("Finishing SEG handler\n");
30     exit(EXIT_FAILURE);
31 }
32
33
34 int main(int argc, char *argv[]){
35     if (signal(SIGFPE, FP_handler) == SIG_ERR)
36         printf("Cant set SIGFP handler\n");
37     if (signal(SIGSEGV, SEG_handler) == SIG_ERR)
38         printf("Cant set SIGSEGV handler\n");
39
40     if (argc == 2) {
41         if (strcmp(argv[1], "-FP") == 0)
42             GenerateFPException();
43         else if (strcmp(argv[1], "-SEG") == 0) {
44             GenerateSEGException();
45         }
46     }
47  return 0;
48 }
```

Листинг 3.26: p6.cpp

При компиляции был добавлен ключ g, для формирования таблицы символов, для установки точки остановки.

```
1  root@kali:~/Desktop/linuxEx/p1# g++ p6.cpp -o p6.o -g
2
3  root@kali:~/Desktop/linuxEx/p1# gdb p6.o
4  (gdb) break 22
5  Breakpoint 1 at 0x939: file p6.cpp, line 22.
6  (gdb) run -FP
7  Starting program: /root/Desktop/linuxEx/p1/p6.o -FP
8
9  Program received signal SIGFPE, Arithmetic exception.
10 __GI_raise (sig=8) at ../sysdeps/unix/sysv/linux/raise.c:51
11 51   ../sysdeps/unix/sysv/linux/raise.c: No such file or directory.
12 (gdb) c
13 Continuing.
14 Received signal FP 8
15
16 Breakpoint 1, FP_handler (signum=8) at p6.cpp:22
17 22       GenerateSEGException();
18 (gdb) s
19 GenerateSEGException () at p6.cpp:11
```

```
11      raise(11);
(gdb) c
Continuing.
Finishing FP handler
[Inferior 1 (process 4785) exited with code 01]
```

Листинг 3.27: Компиляция и отладка p6.cpp

Точка остановки были установлена в 22 строчке, на функции генерации сигнала SEG, далее в отладчике был выполнен шаг(step), по которому был заход в функцию-генератор сигнала SEG, однако он был успешно заблокирован, и обработчик сигнала FP успешно закончил свою работу.

3.6 Обработка программных исключений

В unix-подобных системах, для обработки программных исключений используются стандартные средства обработки исключений.

Далее приведена программа, которая программно вызывает и обрабатывает исключения.

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <cstring>

class FPException {
    public:
        int code;
        FPException(int _code) {
            code = _code;
        }
    };

class SEGException {
    public:
        int code;
        SEGException(int _code) {
            code = _code;
        }
    };

void GenerateFPException() {
    throw FPException(1);
}

void GenerateSEGException() {
    throw SEGException(2);
}

int main(int argc, char *argv[]){
    try {
        if (argc == 2) {
            if (strcmp(argv[1], "-FP") == 0)
                GenerateFPException();
            else if (strcmp(argv[1], "-SEG") == 0) {
                GenerateSEGException();
            }
```

```
39          }
40      }
41      catch (FPException err){
42          printf("FPException has been handled\n");
43      }
44      catch (SEGException err) {
45          printf("SEGException has been handled\n");
46      }
47      return 0;
48  }
```

Листинг 3.28: p7.cpp

```
1  root@kali:~/Desktop/linuxEx/p1# g++ p7.cpp -o p7.o
2  root@kali:~/Desktop/linuxEx/p1# ./p7.o -FP
3  FPException has been handled
4  root@kali:~/Desktop/linuxEx/p1# ./p7.o -SEG
5  SEGException has been handled
```

Листинг 3.29: Запуск p7.cpp

```
1  root@kali:~/Desktop/linuxEx/p1# g++ p7.cpp -o p7.o -g
2  root@kali:~/Desktop/linuxEx/p1# gdb p7.o
3  (gdb) break 35
4  Breakpoint 1 at 0xadc: file p7.cpp, line 35.
5  (gdb) run -FP
6  Starting program: /root/Desktop/linuxEx/p1/p7.o -FP
7
8  Breakpoint 1, main (argc=2, argv=0x7fffffffe208) at p7.cpp:35
9  35                  GenerateFPException();
10 (gdb) S
11 GenerateFPException () at p7.cpp:24
12 24      throw FPException(1);
13 (gdb) S
14 FPException::FPException (this=0x555555768ca0, _code=1) at p7.cpp:11
15 11              code = _code;
16 (gdb) s
17 12          }
18 (gdb) s
19 FPException has been handled
20 [Inferior 1 (process 5752) exited normally]
```

Листинг 3.30: Отладка p7.cpp

Отладка показала после перехода в функцию-генератор исключения, сперва был создан объект класса, а затем само исключение что завершило работу программы. Рассмотрим этот процесс подробнее.

С помощью команды

rbreak .*wind*.

были установлены точки остановки, на всех функциях соответствующих этому выражению.

```
1  root@kali:~/Desktop/linuxEx/p1# gdb p7.o
2
3  (gdb) rbreak .*wind*.
4  (gdb) run -FP
5  Starting program: /root/Desktop/linuxEx/p1/p7.o -FP
6
7  Breakpoint 15, 0x00007fc494101200 in _Unwind_RaiseException@plt () from /usr/lib/x86_64-
       ↪ linux-gnu/libstdc++.so.6
```

```
(gdb) c
Continuing.

Breakpoint 41, 0x00007fc493b6c144 in _Unwind_RaiseException () from /lib/x86_64-linux-
    ↪ gnu/libgcc_s.so.1
(gdb) c
Continuing.

Breakpoint 27, 0x00007fc493b5ea00 in _Unwind_Find_FDE@plt () from /lib/x86_64-linux-gnu/
    ↪ libgcc_s.so.1
(gdb) c
Continuing.

Breakpoint 47, 0x00007fc493b6e4d0 in _Unwind_Find_FDE () from /lib/x86_64-linux-gnu/
    ↪ libgcc_s.so.1
(gdb) c
Continuing.

Breakpoint 27, 0x00007fc493b5ea00 in _Unwind_Find_FDE@plt () from /lib/x86_64-linux-gnu/
    ↪ libgcc_s.so.1
(gdb) c
Continuing.

Breakpoint 47, 0x00007fc493b6e4d0 in _Unwind_Find_FDE () from /lib/x86_64-linux-gnu/
    ↪ libgcc_s.so.1
(gdb) c
Continuing.

Breakpoint 27, 0x00007fc493b5ea00 in _Unwind_Find_FDE@plt () from /lib/x86_64-linux-gnu/
    ↪ libgcc_s.so.1
(gdb) c
Continuing.

Breakpoint 47, 0x00007fc493b6e4d0 in _Unwind_Find_FDE () from /lib/x86_64-linux-gnu/
    ↪ libgcc_s.so.1
(gdb) c
Continuing.

Breakpoint 27, 0x00007fc493b5ea00 in _Unwind_Find_FDE@plt () from /lib/x86_64-linux-gnu/
    ↪ libgcc_s.so.1
(gdb) c
Continuing.

Breakpoint 47, 0x00007fc493b6e4d0 in _Unwind_Find_FDE () from /lib/x86_64-linux-gnu/
    ↪ libgcc_s.so.1
(gdb) c
Continuing.

Breakpoint 13, 0x00007fc494100f00 in _Unwind_GetLanguageSpecificData@plt () from /usr/
    ↪ lib/x86_64-linux-gnu/libstdc++.so.6
(gdb) c
Continuing.

Breakpoint 36, 0x00007fc493b69fc0 in _Unwind_GetLanguageSpecificData () from /lib/x86_64
    ↪ -linux-gnu/libgcc_s.so.1
(gdb) c
Continuing.

Breakpoint 9, 0x00007fc4940fff30 in _Unwind_GetRegionStart@plt () from /usr/lib/x86_64-
    ↪ linux-gnu/libstdc++.so.6
(gdb) c
Continuing.

Breakpoint 37, 0x00007fc493b69fd0 in _Unwind_GetRegionStart () from /lib/x86_64-linux-
    ↪ gnu/libgcc_s.so.1
(gdb) c
Continuing.

Breakpoint 16, 0x00007fc494101d30 in _Unwind_GetIPInfo@plt () from /usr/lib/x86_64-linux
    ↪ -gnu/libstdc++.so.6
(gdb) c
Continuing.

Breakpoint 34, 0x00007fc493b69f90 in _Unwind_GetIPInfo () from /lib/x86_64-linux-gnu/
    ↪ libgcc_s.so.1
(gdb) c
```

```
69 Continuing.
70
71 Breakpoint 25, 0x00007fc493b5e920 in _Unwind_GetCFA@plt () from /lib/x86_64-linux-gnu/
       ↪ libgcc_s.so.1
72 (gdb) c
73 Continuing.
74
75 Breakpoint 31, 0x00007fc493b69f30 in _Unwind_GetCFA () from /lib/x86_64-linux-gnu/
       ↪ libgcc_s.so.1
76 (gdb) c
77 Continuing.
78
79 Breakpoint 27, 0x00007fc493b5ea00 in _Unwind_Find_FDE@plt () from /lib/x86_64-linux-gnu/
       ↪ libgcc_s.so.1
80 (gdb) c
81 Continuing.
82
83 Breakpoint 47, 0x00007fc493b6e4d0 in _Unwind_Find_FDE () from /lib/x86_64-linux-gnu/
       ↪ libgcc_s.so.1
84 (gdb) c
85 Continuing.
86
87 Breakpoint 25, 0x00007fc493b5e920 in _Unwind_GetCFA@plt () from /lib/x86_64-linux-gnu/
       ↪ libgcc_s.so.1
88 (gdb) c
89 Continuing.
90
91 Breakpoint 31, 0x00007fc493b69f30 in _Unwind_GetCFA () from /lib/x86_64-linux-gnu/
       ↪ libgcc_s.so.1
92 (gdb) c
93 Continuing.
94
95 Breakpoint 27, 0x00007fc493b5ea00 in _Unwind_Find_FDE@plt () from /lib/x86_64-linux-gnu/
       ↪ libgcc_s.so.1
96 (gdb) c
97 Continuing.
98
99 Breakpoint 47, 0x00007fc493b6e4d0 in _Unwind_Find_FDE () from /lib/x86_64-linux-gnu/
       ↪ libgcc_s.so.1
100 (gdb) c
101 Continuing.
102
103 Breakpoint 25, 0x00007fc493b5e920 in _Unwind_GetCFA@plt () from /lib/x86_64-linux-gnu/
       ↪ libgcc_s.so.1
104 (gdb) c
105 Continuing.
106
107 Breakpoint 31, 0x00007fc493b69f30 in _Unwind_GetCFA () from /lib/x86_64-linux-gnu/
       ↪ libgcc_s.so.1
108 (gdb) c
109 Continuing.
110
111 Breakpoint 27, 0x00007fc493b5ea00 in _Unwind_Find_FDE@plt () from /lib/x86_64-linux-gnu/
       ↪ libgcc_s.so.1
112 (gdb) c
113 Continuing.
114
115 Breakpoint 47, 0x00007fc493b6e4d0 in _Unwind_Find_FDE () from /lib/x86_64-linux-gnu/
       ↪ libgcc_s.so.1
116 (gdb) c
117 Continuing.
118
119 Breakpoint 25, 0x00007fc493b5e920 in _Unwind_GetCFA@plt () from /lib/x86_64-linux-gnu/
       ↪ libgcc_s.so.1
120 (gdb) c
121 Continuing.
122
123 Breakpoint 31, 0x00007fc493b69f30 in _Unwind_GetCFA () from /lib/x86_64-linux-gnu/
       ↪ libgcc_s.so.1
124 (gdb) c
125 Continuing.
126
127 Breakpoint 10, 0x00007fc494100020 in _Unwind_SetGR@plt () from /usr/lib/x86_64-linux-gnu
       ↪ /libstdc++.so.6
128 (gdb) c
129 Continuing.
```

```
130 |
131 | Breakpoint 32, 0x00007fc493b69f40 in _Unwind_SetGR () from /lib/x86_64-linux-gnu/
    |     ↪ libgcc_s.so.1
132 | (gdb) c
133 | Continuing.
134 |
135 | Breakpoint 10, 0x00007fc494100020 in _Unwind_SetGR@plt () from /usr/lib/x86_64-linux-gnu
    |     ↪ /libstdc++.so.6
136 | (gdb) c
137 | Continuing.
138 |
139 | Breakpoint 32, 0x00007fc493b69f40 in _Unwind_SetGR () from /lib/x86_64-linux-gnu/
    |     ↪ libgcc_s.so.1
140 | (gdb) c
141 | Continuing.
142 |
143 | Breakpoint 19, 0x00007fc494102dd0 in _Unwind_SetIP@plt () from /usr/lib/x86_64-linux-gnu
    |     ↪ /libstdc++.so.6
144 | (gdb) c
145 | Continuing.
146 |
147 | Breakpoint 35, 0x00007fc493b69fb0 in _Unwind_SetIP () from /lib/x86_64-linux-gnu/
    |     ↪ libgcc_s.so.1
148 | (gdb) c
149 | Continuing.
150 | FPException has been handled
151 |
152 | Breakpoint 12, 0x00007fc494100a30 in _Unwind_DeleteException@plt () from /usr/lib/x86_64
    |     ↪ -linux-gnu/libstdc++.so.6
153 | (gdb) c
154 | Continuing.
155 |
156 | Breakpoint 45, 0x00007fc493b6c930 in _Unwind_DeleteException () from /lib/x86_64-linux-
    |     ↪ gnu/libgcc_s.so.1
157 | (gdb) c
158 | Continuing.
159 | [Inferior 1 (process 5985) exited normally]
```

Листинг 3.31: Подробная отладка p7.cpp

По результатам отладки, был зафиксирован процесс раскрутки, вызов обработки исключения, завершение программы, что является стандартным порядком обработки исключений.

3.7 Вложенные исключения обработчики

Рассмотрим как происходит процесс обработки вложенных исключений.

```
1  #include <unistd.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <signal.h>
5  #include <cstring>
6
7  class FPException {
8      public:
9          int code;
10         FPException(int _code) {
11             code = _code;
12         }
13     };
14
15 class SEGException {
16     public:
17         int code;
18         SEGException(int _code) {
```

```
19              code = _code;
20          }
21      };
22
23  void GenerateFPException() {
24      throw FPException(1);
25  }
26
27  void GenerateSEGException() {
28      throw SEGException(2);
29  }
30
31  int main(int argc, char *argv[]){
32      try {
33          try {
34              try {
35                  GenerateFPException();
36              }
37              catch (int){
38                  printf("Wrong handler\n");
39              }
40          }
41          catch (FPException err){
42              printf("FPException has been handled\n");
43              GenerateSEGException();
44          }
45      }
46      catch (SEGException err) {
47          printf("SEGException has been handled\n");
48      }
49      return 0;
50  }
```

Листинг 3.32: Отладка p8.cpp

Как и полагалось оба исключения были успешно обработаны, не использовав лишний обработчик(int).

```
1  root@kali:~/Desktop/linuxEx/p1# g++ p8.cpp -o p8.o -g
2  root@kali:~/Desktop/linuxEx/p1# ./p8.o
3  FPException has been handled
4  SEGException has been handled
```

Листинг 3.33: Запуск p8.cpp

Рассмотрим что происходит при отладке. Дополнительно были установлены точки остановки с помощью функции **catch event**, где event - throw, catch, exec и т.д.

```
1  root@kali:~/Desktop/linuxEx/p1# gdb p8.o
2  (gdb) catch throw
3  Catchpoint 1 (throw)
4  (gdb) catch catch
5  Catchpoint 2 (catch)
6  (gdb) run
7  Starting program: /root/Desktop/linuxEx/p1/p8.o
8
9  Catchpoint 1 (exception thrown), 0x00007faa430a1bad in __cxa_throw () from /usr
       ↪ /lib/x86_64-linux-gnu/libstdc++.so.6
10 (gdb) c
11 Continuing.
12
```

```
Catchpoint 2 (exception caught), 0x00007faa430a098f in __cxa_begin_catch ()
    ↪ from /usr/lib/x86_64-linux-gnu/libstdc++.so.6
(gdb) c
Continuing.
FPException has been handled

Catchpoint 1 (exception thrown), 0x00007faa430a1bad in __cxa_throw () from /usr
    ↪ /lib/x86_64-linux-gnu/libstdc++.so.6
(gdb) c
Continuing.

Catchpoint 2 (exception caught), 0x00007faa430a098f in __cxa_begin_catch ()
    ↪ from /usr/lib/x86_64-linux-gnu/libstdc++.so.6
(gdb) c
Continuing.
SEGException has been handled
[Inferior 1 (process 6120) exited normally]
```

Листинг 3.34: Отладка p8.cpp

Отладка показала, что при возникновении исключения в самом глубоко вложенном блоке try системой был проигнорирован ближайший(неподходящий) обработчик, в следствии чего поиск был продолжен, до нахождения требуемого обработчика во внешнем блоке.

3.8 Использование goto

Оператор goto будет использован для проверки того, что в таком случае не раскручивается.

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <cstring>

class FPException {
    public:
        int code;
        FPException(int _code) {
            code = _code;
        }
    };

class SEGException {
    public:
        int code;
        SEGException(int _code) {
            code = _code;
        }
    };

void GenerateFPException() {
    throw FPException(1);
}

void GenerateSEGException() {
    throw SEGException(2);
```

```
29 }
30
31 int main(int argc, char *argv[]){
32     try {
33         try {
34             try {
35                 printf("Enter deepest try block\n");
36                 goto myPoint;
37                 GenerateFPException();
38             }
39             catch (int){
40                 printf("Wrong handler\n");
41             }
42         }
43         catch (FPException err){
44             printf("FPException has been handled\n");
45             GenerateSEGException();
46         }
47     }
48     catch (SEGException err) {
49         printf("SEGException has been handled\n");
50     }
51     myPoint:
52     printf("Exit all blocks with goto operator\n");
53     return 0;
54 }
```

Листинг 3.35: p9.cpp

```
1  root@kali:~/Desktop/linuxEx/p1# g++ p9.cpp -o p9.o -g
2  root@kali:~/Desktop/linuxEx/p1# ./p9.o
3  Enter deepest try block
4  Exit all blocks with goto operator
5  root@kali:~/Desktop/linuxEx/p1# gdb p9.o
6
7  (gdb) catch throw
8  Catchpoint 1 (throw)
9  (gdb) catch catch
10 Catchpoint 2 (catch)
11 (gdb) rbreak .*wind*.
12 (gdb) run
13 Starting program: /root/Desktop/linuxEx/p1/p9.o
14 Enter deepest try block
15 Exit all blocks with goto operator
16 [Inferior 1 (process 6201) exited normally]
```

Листинг 3.36: Запуск и отладка p9.cpp

Как и ожидалось никакие точки остановки не сработали, так как раскрутки стека не происходило.

Вывод

В данной работе был проведен анализ обработки исключений в Linux для аппаратных и программных исключений, в виде обработки сигналов. Были проведены опыты:

- по обработке сигналов;
- вложенные обработчики и исключения;
- блокировка сигналов;
- использование goto.

Механизм сигналов является уникальным, позволяет организовывать как синхронную, так и ассинхронную обработку исключительных ситуаций.

На мой взгляд структурированная обработка исключений(SEH) в windows более гибкой, и более рационально использовать именно её.

Также в работе был использован отладчик gdb, что повысило мои способности по отладке программ вне графической среды какой-либо IDE.

Литература

[1] signal - overview of signals [Электронный ресурс]. — URL: http://manpages.ubuntu.com/manpages/xenial/en/man7/signal.7.html (дата обращения: 2017-11-28).

[2] Signal (IPC) [Электронный ресурс]. — URL: https://en.wikipedia.org/wiki/Signal_(IPC) (дата обращения: 2017-11-28).

[3] raise - send a signal to the caller [Электронный ресурс]. — URL: https://linux.die.net/man/3/raise (дата обращения: 2017-12-03).

[4] signal() - Unix, Linux System Call [Электронный ресурс]. — URL: http://www.tutorialspoint.com/unix_system_calls/signal.htm (дата обращения: 2017-12-03).

[5] sigaction [Электронный ресурс]. — URL: http://ru.manpages.org/sigaction/2 (дата обращения: 2017-12-03).

[6] data structure containing signal information [Электронный ресурс]. — URL: https://www.mkssoftware.com/docs/man5/siginfo_t.5.asp (дата обращения: 2017-12-03).

[7] SIGPROCMASK [Электронный ресурс]. — URL: http://man7.org/linux/man-pages/man2/sigprocmask.2.html (дата обращения: 2017-12-03).