Лекция 26. Ошибки и отладка программ на языке Си. Часть 1.

Ревун Артем Леонидович
ст. преп. кафедры вычислительных систем

Курс «Программирование», весенний семестр, 2024
Сибирский государственный университет телекоммуникаций и информатики (г. Новосибирск)

Содержание

- Определение отладки и отладки программ
- Классификация ошибок
- Знакомство с инструменты отладки на Си: printf debugging, GNU Debugger (GDB), Valgrind (Memcheck), strace/ltrace.

Ошибки в коде (1)

Ошибки в коде – это непредвиденная ситуация, которая может возникнуть во время выполнения программы и требует специальной обработки для корректной работы программы.

In many ways, C is a permissive language; programmers are allowed to shoot themselves in the foot or other body parts if they choose to, and C will make no effort to stop them.

Jens Gustedt

Ошибки в коде (2)

В языке Си в отличии от языков с автоматической обработкой исключений, ***ответственность за обнаружение и обработку ошибок полностью лежит на программисте***. Не обработанные ошибки могут приводить к **непредсказуемому поведению программы**, включая **сбои**, **повреждение данных** и **скрытые уязвимости** в безопасности.

Ошибки в коде (3)

Понимание типов ошибок — позволяет находить ошибки, определять их местонахождение и принимать действия к их исправлениям:

- Синтаксические ошибки
- Семантические ошибки
- Логические ошибки
- Ошибки времени выполнения

Ошибки в коде (4)

Синтаксическая ошибка – опечатка, не знание синтаксиса, которая тормозит компиляцию.

Это наиболее простые ошибки для обнаружения. Они возникают из-за **нарушения правил языка программирования**.

```
printf("Hello, World!\n")
return 0;
```

Компилятор указывает строку и место ошибки. Иногда он говорит о проблеме чуть позже, чем она есть.

Ошибки в коде (5)

Семантическая ошибка – программа работает, но не так, как задумано.

Программа компилируется и запускается, но логика поведения отличается от ожидаемой.

```
if (num_purchases >= 10)
    printf("Вы получили скидку!");
    subtotal = subtotal * 0.9;
```

Ошибки в коде (6)

Логическая ошибка – всё «работает», но результат неверный. Ошибка в алгоритме, которая приводит к некорректному выводу.

```
int i, j;
j=(++i)*2+ --i;
```

Возникает неопределенное поведение или undefined behaviour, UB.

Ошибки в коде (7)

Ошибки времени выполнения – программа падает во время работы. Частые причины: деление на ноль,

выход за границы массива, доступ к освобожденной памяти, бесконечные циклы.

Эти ошибки труднее всего найти, особенно если они проявляются не всегда.

Ошибки в коде (8)

Как определить/исправить ошибки в коде:

- Читать документацию
- Читайте сообщения об ошибках внимательно
- Тестируйте функции по отдельности
- Используйте отладчик (debugger)
- Пишите юнит-тесты

Отладка (1)

Отладка (debugging) — это процесс выявления, анализа и устранения дефектов программы, которые приводят к её некорректному поведению или аварийному завершению.

Цель: обеспечить детерминированность поведения , предсказуемое использование ресурсов и надежность исполнения .

Отладка (2)

Основные инструменты отладки на Си:

- **printf debugging** — неформальный, но эффективный способ диагностики.
- GNU Debugger (GDB) — основной инструмент отладки на СИ
- Valgrind (Memcheck) — обнаруживает утечки памяти, чтение неинициализированной памяти
- strace/ltrace — трассировка системных вызовов и библиотечных функций

Отладка (3)

printf debugging

Это подход к диагностике программы, при котором вы вручную добавляете вызовы **printf()** (или аналогичные функции вывода) в ключевые точки кода, чтобы **наблюдать** за состоянием переменных, порядком выполнения, а также **для поиска источника ошибок**.

Отладка (4)

```
int a = 5; int b = 0; int result;
printf("[DEBUG] a = %d, b = %d\n", a, b);
if (b != 0) {
    result = a / b;
    printf("[DEBUG] result = %d\n", result);
} else {
    printf("[ERROR] Division by zero!\n");
}
```

Отладка (5)

- Добавьте тег [DEBUG] к сообщениям , чтобы легко фильтровать их в логе.
- Использование функциональных макросов

```
#define DEBUG_PRINT(fmt, ...) \
fprintf(stderr, "[DEBUG] " fmt "\n", ##__VA_ARGS__)
```

- Выводите адреса, указатели и регистры , если требуется глубокая диагностика

```
DEBUG_PRINT("ptr = %p", ptr);
```

Отладка (6)

- Использование `fflush`(), особенно при отладке аварийного завершения:

```
fflush(stdout);
```

- Перенаправляйте вывод в файл , чтобы сохранять результаты:

```
./main > debug.log 2>&1
```

Отладка (7)

Преимущества

- Видеть текущие значения переменных
- Отслеживать поток управления
- Обнаруживать места, куда программа "не доходит "

Недостатки

- Изменение поведения программы
- Загромождение кода(Много printf)
- Требуется перекомпиляция

Отладка (8)

GNU Debugger (GDB) — это мощный инструмент отладки с открытым исходным кодом , разработанный проектом GNU. Он позволяет анализировать поведение программы на низком уровне , что особенно важно при системном программировании.

```
GNU gdb (GDB) 14.1
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-w64-mingw32".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.
```

Отладка (9)

GNU Debugger (GDB) предназначен для:

- Пошагового выполнения программы
- Установки точек останова (breakpoints)
- Просмотра состояния памяти и регистров
- Анализа падений программы (например, segfault)
- Работы с многопоточными и low-level программами

Отладка (10)

Для полноценной работы GDB программа должна быть скомпилирована с флагом –g

```
gcc -g -o my_program my_program.c
```

Этот флаг добавляет в исполняемый файл информацию о: именах переменных, номерах строк, именах функций, типах данных.

Без этого **GDB будет работать**, но без возможности просматривать исходный код и переменные.

Отладка (11)

Запуск отладчика:

```
gdb ./my_program
```

Запуск отладчика совместно с программой:

```
gdb -ex run --args ./my_program arg1 arg2
```

`run(r)`	Запуск программы
`break(b)`	Установка точки останова (по имени функции или номеру строки)
`step(s)`	Выполнить следующую строку (входить внутрь функций)
`next(n)`	Выполнить следующую строку (не входить в функции)
`continue(c)`	Продолжить выполнение до следующей точки останова

Отладка (12)

`display`	Добавить выражение из списка автоматического отображения
`undisplay`	Удалить выражение из списка автоматического отображения
`print(p)`	Вывести значение переменной или выражения
`info registers`	Показать содержимое регистров процессора
`backtrace(bt)`	Показать трассировку стека вызовов
`Disassemble`	Показать машинный код функции
`quit(q)`	Выйти из GDB

Отладка (13)

```
#include <stdio.h>
#define N 10
int main() {
int arr[N]; unsigned i;
for (i = N – 1; i >= 0; --i) {
 arr[i] = i; printf(“%d ”,arr[i]); }
return 0; }
```

Отладка (13)

```
$ gcc -Wall -o example main.c
//Компилируется без проблем
$ ./example
Bus error (core dumped)
```

Отладка (14)

```
$ gcc –Wall –g –O0 –o example main.c
$ gdb ./example
(gdb) run
Program received signal SIGBUS, Bus error.
0x0000000000400509 in main () at main.c:11
6                   arr[i] = i;
```

Отладка (15)

```
(gdb) print i                    p i
$1 = 4294967295
(gdb) break main                 b main
(gdb) run                        r
(gdb) Breakpoint 1 at 0x7ff7e114145d:
file path, line 5.
5 for (i = N - 1; i >= 0; --i) {
(gdb) display i                  d i
```

Отладка (16)

```
(gdb) next                              n
. . .
(gdb) next                              n
6               for (i = N - 1; i >= 0; --i)
{
1: i = 0
...
```

Отладка (17)

```
(gdb) next                          n
1: i = 4294967295
Thread 1 received signal SIGSEGV,
Segmentation fault.
0x00007ff7e114146a in main () at path:6
6           arr[i] = i;
printf("%d",arr[i]); }
```

Отладка (18)

```
(gdb) next                                  n
[New Thread 17832.0x3404]
Thread 1 received signal SIGSEGV,
Segmentation fault.
0x00007ff7e114146a in main () at path:6
6           arr[i] = i;
printf("%d",arr[i]); }
```

Отладка (17)

```
(gdb) next                                  n
[Thread 17832.0x4aa0 exited with code
3221225477]
[Thread 17832.0x6434 exited with code
3221225477]

Program terminated with signal SIGSEGV,
Segmentation fault.
The program no longer exists.
```

Трассировка (1)

Трассировка – процесс *пошагового* исполнения программы.

В режиме трассировки программист видит последовательность выполнения операторов, а также может запросить текущие значения переменных на данном шаге выполнения программы. Это упрощает процесс обнаружения ошибок.

Трассировка (2)

Трассировка может быть начата и окончена в любом месте программы.

Выполнение программы может останавливаться:

- на каждой команде;
- на точках останова;

Трассировка может выполняться *с заходом* в процедуры и без него.

Трассировка (3)

Без оптимизации

```
int a = 10;

int b = 20;

int c = a + b;




printf("c = %d\n", c);
. . .
```

gcc -O0 -S main.c

```
movl    $10, -12(%rbp)

movl    $20, -8(%rbp)

movl    -8(%rbp), %eax
movl    -12(%rbp), %edx
addl    %edx, %eax
movl    %eax, -4(%rbp)

movl    $.LC0, %eax
movl    -4(%rbp), %edx
movl    %edx, %esi
movq    %rax, %rdi
movl    $0, %eax
call    printf
```

Трассировка (4)

Первый ур. оптимизации

```
int a = 10;

int b = 20;

int c = a + b;




printf("c = %d\n", c);
. . .
```

gcc -O1 -S main.c

```
movl    $30, %edx




movl    $.LC0, %esi
movl    $1, %edi
movl    $0, %eax
call    __printf_chk
```

Трассировка (5)

Третий ур. оптимизации

```
int a = 10;

int b = 20;

printf("c = %d\n", c);


int c = a + b;


printf("c = %d\n", c);

. . .
```

`gcc -O3 -S main.c`

```
movl    $1, %edi
movl    $0, %eax

movl    $30, %edx


movl    $.LC0, %esi
call    __printf_chk
```

Спасибо за внимание

Ревун Артем Леонидович
ст. преп. кафедры вычислительных систем

Курс «Программирование», весенний семестр, 2024
Сибирский государственный университет телекоммуникаций и информатики (г. Новосибирск)