Санкт-Петербургский Государственный Политехнический Университет

Институт информационных технологий и управления

Кафедра Компьютерных Систем и Программных Технологий

Отчет по лабораторной работе

По дисциплине «Системное программирование»

По теме «Инструментальные средства профилирования, трассировки и диагностики в ОС»

Работу выполнила студентка группы № 13541/3 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Климова Д.А.

Работу принял преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Душутина Е.В.

.

Санкт-Петербург

2017

Программа работы

В Linux

Ознакомиться с возможностями утилит strace/ptrace, syslog и их аналогов,

провести эксперименты, наблюдая с их помощью за выполнением других стандартных утилит, а затем собственных программ таким образом, чтобы максимально продемонстрировать возможности средств трассировки и журналирования.

В качестве испытуемых программ можно использовать в том числе и программы для исследования IPC (4 курс), а затем «исключений» (5 курс).

Модифицировать код испытуемых программ вставками соответствующих системных функций для профилирования функционирования этих программ (трассировки и «логирования»), сохранить в файле и подробно проанализировать полученный событийно-временной срез функционирования в отчете.

В Windows

Ознакомиться с составом утилит– функциональных аналогов Linux-утилит, написанных под ОС Windows(WinTools). Провести эксперименты по их использованию, задокументировать и запротоколировать результаты аналогично этапу работы с Linux.

Ознакомиться и провести эксперименты по применению нативных средств профилирования и журналирования, встроенных в Windows ОС, включая средства с графическим интерфейсом.

Ход работы

**Linux**

strace – утилита, использующаяся в UNIX-системах для профилирования и диагностики программ. Данная утилита позволяет определить системные вызовы, используемые в анализируемой программе.

В общем случае запуск strace выглядит так: Strace program\_name

Утилита запустит программу program\_name и будет выводить в поток стандартного вывода сообщения о выполняемых системных вызовах. Зачастую засорение стандартного вывода сообщениями трассировки нежелательно, поскольку в нём будет трудно отыскать то, что выводит процесс.

|  |
| --- |
| ubuntu@ubuntu-K53SJ:~/5/1$ strace top  … read(8, "19248 1624 1344 97 0 265 0\n", 1024) = 27  close(8) = 0  stat("/proc/2289", {st\_mode=S\_IFDIR|0555, st\_size=0, ...}) = 0  open("/proc/2289/stat", O\_RDONLY) = 8  read(8, "2289 (evolution-calen) S 1873 19"..., 1024) = 321  close(8) = 0  open("/proc/2289/statm", O\_RDONLY) = 8  read(8, "268834 12463 3406 2 0 205298 0\n", 1024) = 31  close(8) = 0  stat("/proc/2313", {st\_mode=S\_IFDIR|0555, st\_size=0, ...}) = 0  open("/proc/2313/stat", O\_RDONLY) = 8  read(8, "2313 (gvfsd-burn) S 1873 1940 19"..., 1024) = 303  close(8) = 0  open("/proc/2313/statm", O\_RDONLY) = 8  read(8, "67901 1174 1069 7 0 55431 0\n", 1024) = 28  close(8) = 0  stat("/proc/2319", {st\_mode=S\_IFDIR|0555, st\_size=0, ...}) = 0  open("/proc/2319/stat", O\_RDONLY) = 8  read(8, "2319 (gvfsd-trash) S 1873 1940 1"..., 1024) = 304  close(8) = 0  open("/proc/2319/statm", O\_RDONLY) = 8  read(8, "90159 2129 1417 10 0 73898 0\n", 1024) = 29  close(8) = 0  stat("/proc/2330", {st\_mode=S\_IFDIR|0555, st\_size=0, ...}) = 0  open("/proc/2330/stat", O\_RDONLY) = 8  read(8, "2330 (gvfsd-metadata) S 1873 194"..., 1024) = 308  close(8) = 0  open("/proc/2330/statm", O\_RDONLY) = 8  read(8, "31484 1454 1266 17 0 18619 0\n", 1024) = 29  close(8) = 0 … |

Лучше перенаправить вывод strace в отдельный файл, который потом можно анализировать: strace -o trace\_output.txt program\_name

|  |
| --- |
| ubuntu@ubuntu-K53SJ:~/5/1$ strace -o str.txt -s 512 -v top |

Содержимое файла str.txt

|  |
| --- |
| … execve("/usr/bin/top", ["top"], ["LC\_PAPER=ru\_RU.UTF-8", "XDG\_VTNR=7", "XDG\_SESSION\_ID=c2", "LC\_ADDRESS=ru\_RU.UTF-8", "CLUTTER\_IM\_MODULE=xim", "LC\_MONETARY=ru\_RU.UTF-8", "SELINUX\_INIT=YES", "XDG\_GREETER\_DATA\_DIR=/var/lib/lightdm-data/ubuntu", "SESSION=ubuntu", "GPG\_AGENT\_INFO=/run/user/1000/keyring-hc2xyj/gpg:0:1", "SHELL=/bin/bash", "XDG\_MENU\_PREFIX=gnome-", "VTE\_VERSION=3409", "TERM=xterm", "LC\_NUMERIC=ru\_RU.UTF-8", "WINDOWID=69206027", "GNOME\_KEYRING\_CONTROL=/run/user/1000/keyring-hc2xyj", "UPSTART\_SESSION=unix:abstract=/com/ubuntu/upstart-session/1000/1873", "GTK\_MODULES=overlay-scrollbar:unity-gtk-module", "USER=ubuntu", "LC\_TELEPHONE=ru\_RU.UTF-8", "LS\_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31;01:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:\*.tar=01;31:\*.tgz=01;31:\*.arj=01;31:\*.taz=01;31:\*.lzh=01;31:\*.lzma=01;31:\*.tlz=01;31:\*.txz=01;31:\*.zip=01;31:\*.z=01;31:\*.Z=01;31:\*.dz=01;31:\*.gz=01;31:\*.lz=01;31:\*.xz=01;31:\*.bz2=01;31:\*.bz=01;31:\*.tbz=01;31:\*.tbz2=01;31:\*.tz=01;31:\*.deb=01;31:\*.rpm=01;31:\*.jar=01;31:\*.war=01;31:\*.ear=01;31:\*.sar=01;31:\*.rar=01;31:\*.ace=01;31:\*.zoo=01;31:\*.cpio="..., … |

Теперь мы хотим изучить трассируемый процесс более подробно. Например, могут быть интересны не только детали, но и полная таблица всех системных вызовов, используемых процессом, включая ошибки.

$ strace -c command

С помощью опции –c можно вывести таблицу всех используемых системных вызовов:

|  |
| --- |
| ubuntu@ubuntu-K53SJ:~/5/1$ strace -c top  % time seconds usecs/call calls errors syscall  ------ ----------- ----------- --------- --------- ----------------  50.44 0.000057 0 1554 close  22.12 0.000025 0 1566 read  16.81 0.000019 0 1570 22 open  10.62 0.000012 0 775 14 stat  0.00 0.000000 0 9 write  0.00 0.000000 0 34 fstat  0.00 0.000000 0 29 lseek  0.00 0.000000 0 51 mmap  0.00 0.000000 0 20 mprotect  0.00 0.000000 0 23 munmap  0.00 0.000000 0 5 brk  0.00 0.000000 0 105 2 rt\_sigaction  0.00 0.000000 0 15 ioctl  0.00 0.000000 0 16 15 access  0.00 0.000000 0 1 dup  0.00 0.000000 0 1 dup2  0.00 0.000000 0 1 nanosleep  0.00 0.000000 0 66 alarm  0.00 0.000000 0 2 socket  0.00 0.000000 0 2 2 connect  0.00 0.000000 0 1 execve  0.00 0.000000 0 1 uname  0.00 0.000000 0 44 fcntl  0.00 0.000000 0 10 getdents  0.00 0.000000 0 1 getuid  0.00 0.000000 0 1 arch\_prctl  0.00 0.000000 0 1 sched\_getaffinity  0.00 0.000000 0 5 openat  0.00 0.000000 0 3 pselect6  0.00 0.000000 0 1 dup3  ------ ----------- ----------- --------- --------- ----------------  100.00 0.000113 5913 55 total  ubuntu@ubuntu-K53SJ:~/5/1$ |

Кроме этого, у опции –e trace уже есть несколько предопределенных наборов. Например, можно указать значение опции process и получить все используемые системные вызовы, относящиеся к управлению процессами. Так же можно получить функции работающие с сигналами, ipc и т.д.

strace -e trace=\signal -o strace\_sig.txt top

|  |
| --- |
| ubuntu@ubuntu-K53SJ:~/5/1$ strace -e trace=\signal -o strace\_sig.txt top |

Содержимое файла strace\_sig.txt

|  |
| --- |
| rt\_sigaction(SIGRT\_32, {0x405250, [], SA\_RESTORER, 0x7f3af6873cb0}, NULL, 8) = 0  rt\_sigaction(SIGRT\_31, {0x405250, [], SA\_RESTORER, 0x7f3af6873cb0}, NULL, 8) = 0  rt\_sigaction(SIGRT\_30, {0x405250, [], SA\_RESTORER, 0x7f3af6873cb0}, NULL, 8) = 0  rt\_sigaction(SIGRT\_29, {0x405250, [], SA\_RESTORER, 0x7f3af6873cb0}, NULL, 8) = 0  rt\_sigaction(SIGRT\_28, {0x405250, [], SA\_RESTORER, 0x7f3af6873cb0}, NULL, 8) = 0  rt\_sigaction(SIGRT\_27, {0x405250, [], SA\_RESTORER, 0x7f3af6873cb0}, NULL, 8) = 0  rt\_sigaction(SIGRT\_26, {0x405250, [], SA\_RESTORER, 0x7f3af6873cb0}, NULL, 8) = 0  rt\_sigaction(SIGRT\_25, {0x405250, [], SA\_RESTORER, 0x7f3af6873cb0}, NULL, 8) = 0  rt\_sigaction(SIGRT\_24, {0x405250, [], SA\_RESTORER, 0x7f3af6873cb0}, NULL, 8) = 0  rt\_sigaction(SIGRT\_23, {0x405250, [], SA\_RESTORER, 0x7f3af6873cb0}, NULL, 8) = 0  rt\_sigaction(SIGRT\_22, {0x405250, [], SA\_RESTORER, 0x7f3af6873cb0}, NULL, 8) = 0  rt\_sigaction(SIGRT\_21, {0x405250, [], SA\_RESTORER, 0x7f3af6873cb0}, NULL, 8) = 0  rt\_sigaction(SIGRT\_20, {0x405250, [], SA\_RESTORER, 0x7f3af6873cb0}, NULL, 8) = 0  rt\_sigaction(SIGRT\_19, {0x405250, [], SA\_RESTORER, … |

strace -e trace=\file -o strace\_file.log top

|  |
| --- |
| ubuntu@ubuntu-K53SJ:~/5/1$ strace -e trace=\file -o strace\_file.log top  strace\_file.log  execve("/usr/bin/top", ["top"], [/\* 70 vars \*/]) = 0  access("/etc/ld.so.nohwcap", F\_OK) = -1 ENOENT (No such file or directory)  access("/etc/ld.so.preload", R\_OK) = -1 ENOENT (No such file or directory)  open("/etc/ld.so.cache", O\_RDONLY|O\_CLOEXEC) = 3  access("/etc/ld.so.nohwcap", F\_OK) = -1 ENOENT (No such file or directory)  open("/lib/x86\_64-linux-gnu/libprocps.so.3", O\_RDONLY|O\_CLOEXEC) = 3  access("/etc/ld.so.nohwcap", F\_OK) = -1 ENOENT (No such file or directory)  open("/lib/x86\_64-linux-gnu/libtinfo.so.5", O\_RDONLY|O\_CLOEXEC) = 3  access("/etc/ld.so.nohwcap", F\_OK) = -1 ENOENT (No such file or directory)  open("/lib/x86\_64-linux-gnu/libdl.so.2", O\_RDONLY|O\_CLOEXEC) = 3  access("/etc/ld.so.nohwcap", F\_OK) = -1 ENOENT (No such file or directory)  open("/lib/x86\_64-linux-gnu/libc.so.6", O\_RDONLY|O\_CLOEXEC) = 3  open("/sys/devices/system/cpu/online", O\_RDONLY|O\_CLOEXEC) = 3  open("/proc/self/stat", O\_RDONLY) = 3  open("/usr/lib/locale/locale-archive", O\_RDONLY|O\_CLOEXEC) = 3  open("/usr/share/locale/locale.alias", O\_RDONLY|O\_CLOEXEC) = 3  open("/usr/share/locale/ru/LC\_MESSAGES/procps-ng.mo", O\_RDONLY) = -1 ENOENT (No such file or directory)  open("/usr/share/locale-langpack/ru/LC\_MESSAGES/procps-ng.mo", O\_RDONLY) = -1 ENOENT (No such file or directory)  open("/usr/share/locale/en/LC\_MESSAGES/procps-ng.mo", O\_RDONLY) = -1 ENOENT (No such file or directory)  open("/usr/share/locale-langpack/en/LC\_MESSAGES/procps-ng.mo", O\_RDONLY) = -1 ENOENT (No such file or directory)  open("/etc/ld.so.cache", O\_RDONLY|O\_CLOEXEC) = 3  access("/etc/ld.so.nohwcap", F\_OK) = -1 ENOENT (No such file or directory)  open("/lib/x86\_64-linux-gnu/tls/x86\_64/libnuma.so", O\_RDONLY|O\_CLOEXEC) = -1 ENOENT (No such file or directory)  ubuntu@ubuntu-K53SJ:~/5/1$ strace -e trace=\network -o strace\_net\_top2.log top  strace\_net\_top2.log  socket(PF\_LOCAL, SOCK\_STREAM|SOCK\_CLOEXEC|SOCK\_NONBLOCK, 0) = 5  connect(5, {sa\_family=AF\_LOCAL, sun\_path="/var/run/nscd/socket"}, 110) = -1 ENOENT (No such file or directory)  socket(PF\_LOCAL, SOCK\_STREAM|SOCK\_CLOEXEC|SOCK\_NONBLOCK, 0) = 5  connect(5, {sa\_family=AF\_LOCAL, sun\_path="/var/run/nscd/socket"}, 110) = -1 ENOENT (No such file or directory)  +++ exited with 0 +++ |

Таким образом, утилита strace предоставляет широкий набор возможностей по диагностике используемых программой системных вызовов.

Syslog

syslog - стандарт отправки сообщений о происходящих в системе событиях (логов), использующийся в компьютерных сетях, работающих по протоколу IP.

Протокол syslog прост: отправитель посылает короткое текстовое сообщение, размером меньше 1024 байт получателю сообщения. Получатель при этом носит имя <syslogd>, <syslog daemon>, либо же, <syslog server>. Сообщения могут отправляться как по UDP, так и по TCP. Как правило, такое сообщение отсылается в открытом виде.

Syslog используется для удобства администрирования и обеспечения информационной безопасности. Он реализован под множество платформ и используется в множестве устройств. Поэтому, использование syslog позволяет обеспечить сбор информации с разных мест и хранение её в едином репозитории.

В UNIX-системах для использования syslog необходимо подключить файл syslog.h. Пример использования:

openlog("irq\_exc", LOG\_PID, LOG\_LOCAL1);

syslog(LOG\_INFO, "My logging message: setting signal handlers");

closelog();

openlog() устанавливает связь с программой, ведущей системный журнал. Первый аргумент добавляется к каждому сообщению и обычно представляет собой название программы. Второй аргумент – option указывает флаг управляющий работой openlog() и соответствующих вызовов syslog(). В данном примере у option установлен LOG\_PID. Это значит, что к каждому сообщению будет добавляться pid нашей программы.

syslog() создает сообщение и передает его демону. Первым аргументом syslog() получает уровень критичности сообщения. Всего существует 8 уровней: самый нижний LOG\_DEBUG, самый верхний LOG\_EMERG. Остальные аргументы передаются так же, как в printf.

closelog() закрывает описатель, используемый для записи данных в журнал.

Программа  test.c, использующую syslog:

|  |
| --- |
| #include <stdlib.h>  #include <stdio.h>  #include <syslog.h>  #define DEBUG  int main(){  int i=0;  openlog("test",LOG\_PID,LOG\_USER);  #ifdef DEBUG  syslog(LOG\_DEBUG,"try to sending 10 messages");  #endif  for (i=0;i<10;i++){  syslog(LOG\_INFO,"info message [i = %d] ",i);  };  #ifdef DEBUG  syslog(LOG\_DEBUG,"try log to stderr");  #endif  closelog();  openlog("test\_stderr",LOG\_PERROR | LOG\_PID,LOG\_USER);  syslog(LOG\_INFO,"this is attempt to use stderr for syslog");  closelog();  return 0;  }; |

Компилируем программу:

|  |
| --- |
| ubuntu@ubuntu-K53SJ:~/5/1$ gcc -o test test.c  ubuntu@ubuntu-K53SJ:~/5/1$ ./test  test\_stderr[4584]: this is attempt to use stderr for syslog |

После запуска программы, был просмотрен файл журнала /var/log/messages, в котором можно было наблюдать сгенерированные программой записи :

|  |
| --- |
| Sep 22 01:34:28 ubuntu-K53SJ udisksd[2283]: Mounted /dev/sda4 at /media/ubuntu/58DC4029DC4003AA on behalf of uid 1000  Sep 22 01:43:19 ubuntu-K53SJ kernel: [ 2157.417854] usb 1-1.4: reset high-speed USB device number 5 using ehci-pci  Sep 22 01:45:28 ubuntu-K53SJ dbus[709]: [system] Activating service name='org.freedesktop.hostname1' (using servicehelper)  Sep 22 01:45:28 ubuntu-K53SJ kernel: [ 2286.845627] systemd-hostnamed[4321]: Warning: nss-myhostname is not installed. Changing the local hostname might make it unresolveable. Please install nss-myhostname!  Sep 22 01:45:28 ubuntu-K53SJ dbus[709]: [system] Successfully activated service 'org.freedesktop.hostname1'  Sep 22 02:01:36 ubuntu-K53SJ test[4584]: try to sending 10 messages  Sep 22 02:01:36 ubuntu-K53SJ test[4584]: info message [i = 0]  Sep 22 02:01:36 ubuntu-K53SJ test[4584]: info message [i = 1]  Sep 22 02:01:36 ubuntu-K53SJ test[4584]: info message [i = 2]  Sep 22 02:01:36 ubuntu-K53SJ test[4584]: info message [i = 3]  Sep 22 02:01:36 ubuntu-K53SJ test[4584]: info message [i = 4]  Sep 22 02:01:36 ubuntu-K53SJ test[4584]: info message [i = 5]  Sep 22 02:01:36 ubuntu-K53SJ test[4584]: info message [i = 6]  Sep 22 02:01:36 ubuntu-K53SJ test[4584]: info message [i = 7]  Sep 22 02:01:36 ubuntu-K53SJ test[4584]: info message [i = 8]  Sep 22 02:01:36 ubuntu-K53SJ test[4584]: info message [i = 9]  Sep 22 02:01:36 ubuntu-K53SJ test[4584]: try log to stderr  Sep 22 02:01:36 ubuntu-K53SJ test\_stderr[4584]: this is attempt to use stderr for syslog  Sep 22 02:03:36 ubuntu-K53SJ dbus[709]: [system] Activating service name='org.freedesktop.hostname1' (using servicehelper)  Sep 22 02:03:36 ubuntu-K53SJ kernel: [ 3374.969727] systemd-hostnamed[4622]: Warning: nss-myhostname is not installed. Changing the local hostname might make it unresolveable. Please install nss-myhostname!  Sep 22 02:03:36 ubuntu-K53SJ dbus[709]: [system] Successfully activated service 'org.freedesktop.hostname1'  Sep 22 02:04:49 ubuntu-K53SJ dhclient: DHCPREQUEST of 192.168.0.102 on eth0 to 192.168.0.1 port 67 (xid=0x60155aa)  Sep 22 02:04:49 ubuntu-K53SJ dhclient: DHCPACK of 192.168.0.102 from 192.168.0.1 |

Данный механизм позволяет в системных программах создавать одинаковые системы логирования для облегчения администрирования программы.

Backtrace

backtrace – это системный вызов, который позволяет получать трассировку программы. Под трассировкой понимается массив активных в данный момент вызовов функций, т.е. текущий стек вызовов функций. . Для демонстрации возможностей рассмотрим следующую программу:

all\_signals.c

|  |
| --- |
| #include <execinfo.h>  #include <stdio.h>  #include <stdlib.h>  /\* Получаем информацию о стеке вызовов ф-ций и печатаем ее \*/  void print\_trace (void)  {  void \*array[10];  size\_t size;  char \*\*strings;  size\_t i;  size = backtrace (array, 10);  strings = backtrace\_symbols (array, size);  printf ("Obtained %zd stack frames.\n", size);  for (i = 0; i < size; i++)  printf ("%s\n", strings[i]);  free (strings);  }  /\* Заглушка просто для того, что бы бы можно было посмотреть стек \*/  void dummy\_function (void)  {  print\_trace ();  }  int main (void)  {  dummy\_function ();  return 0;  } |

Запустим ее :

|  |
| --- |
| ubuntu@ubuntu-K53SJ:~/5/1$ gcc -std=c99 -D\_POSIX\_C\_SOURCE=200112L -g -D\_XOPEN\_SOURCE=700 -rdynamic -o signals all\_signals.c  ubuntu@ubuntu-K53SJ:~/5/1$ ./signals  Obtained 5 stack frames.  ./signals(print\_trace+0x19) [0x400936]  ./signals(dummy\_function+0x9) [0x4009b8]  ./signals(main+0x9) [0x4009c3]  /lib/x86\_64-linux-gnu/libc.so.6(\_\_libc\_start\_main+0xf5) [0x7fdec13e2f45]  ./signals() [0x400859] |

Мы видим, что программа напечатала в консоль текущий стек вызовов функций.

Данный системный вызов дает большие возможности для отладки своих программ, так как от позволяет отслеживать каким образом программа дошла в ту или иную точку кода.

**Gdb**

gdb – отладчик проекта GNU, который работает на большинстве UNIX-систем и позволяет отлаживать программы, написанные на C/C++, Pascal, Ada, Fortran и т.д. Этот отладчик предоставляет огромные возможности для анализа и диагностирования программ. Простейший вариант использования данного отладчика – запускать свою программу в отладчике и в случае возникновения проблем мы сможем сразу приступить к отладке:

|  |
| --- |
| #include <unistd.h>  #include <sys/types.h>  #include <stdio.h>  int main()  {  /\* int a[] = {1, 2, 3};\*/  int i=0;  /\* printf("pid=%d\n", getpid());  while (1) {  sleep(1);  printf("i=%d\n",i);    }\*/  return 0;  } |

Запустим:

|  |
| --- |
| ubuntu@ubuntu-K53SJ:~/5/1$ gcc -o remote -g remote.c  ubuntu@ubuntu-K53SJ:~/5/1$ gdb ./remote  GNU gdb (Ubuntu 7.7.1-0ubuntu5~14.04.2) 7.7.1  Copyright (C) 2014 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-linux-gnu".  Type "show configuration" for configuration details.  For bug reporting instructions, please see:  <http://www.gnu.org/software/gdb/bugs/>.  Find the GDB manual and other documentation resources online at:  <http://www.gnu.org/software/gdb/documentation/>.  For help, type "help".  Type "apropos word" to search for commands related to "word"...  Reading symbols from ./remote...done.  (gdb) |

Код программы можно посмотреть при помощи команды list

|  |
| --- |
| (gdb) list  1 #include <unistd.h>  2 #include <sys/types.h>  3 #include <stdio.h>  4 int main()  5 {  6 /\* int a[] = {1, 2, 3};\*/  7 int i=0;  8 /\* printf("pid=%d\n", getpid());  9 while (1) {  10 sleep(1);  (gdb)  11 printf("i=%d\n",i);  12  13 }\*/  14 return 0;  15 } |

Команда print позволяет просматривать переменные и осуществлять арифметические операции, убедимся в этом :

|  |
| --- |
| (gdb) print 1+2  $1 = 3 |

Поставим break на main

|  |
| --- |
| (gdb) b 4  Breakpoint 1 at 0x4004f1: file remote.c, line 4. |

Запустим программу :

|  |
| --- |
| (gdb) r  Starting program: /home/ubuntu/5/1/remote  Breakpoint 1, main () at remote.c:7  7 int i=0; |

Просмотрим значение переменной i

|  |
| --- |
| (gdb) print i  $2 = 0 |

Установка нового значения осуществляется следующим образом :

|  |
| --- |
| (gdb) set var i=10  (gdb) print i  $3 = 10 |

Просмотрим адрес и размер переменной i

|  |
| --- |
| (gdb) print &i  $10 = (int \*) 0x7fffffffddfc  (gdb) print sizeof(i)  $11 = 4 |

Модифицируем программу

|  |
| --- |
| #include <unistd.h>  #include <sys/types.h>  #include <stdio.h>  int main()  {  int a[] = {1, 2, 3};  /\* int i=0;  /\* printf("pid=%d\n", getpid());  while (1) {  sleep(1);  printf("i=%d\n",i);    }\*/  return 0;  } |

|  |
| --- |
| ubuntu@ubuntu-K53SJ:~/5/1$ gcc -o remote -g remote.c  ubuntu@ubuntu-K53SJ:~/5/1$ gdb ./remote  Убедимся, что измениения прошли  (gdb) list  1 #include <unistd.h>  2 #include <sys/types.h>  3 #include <stdio.h>  4 int main()  5 {  6 int a[] = {1, 2, 3};  7 /\* int i=0;  8 /\* printf("pid=%d\n", getpid());  9 while (1) {  10 sleep(1);  (gdb)  11 printf("i=%d\n",i);  12  13 }\*/  14 return 0;  15 } |

Поставим точку останова

|  |
| --- |
| (gdb) break main  Breakpoint 1 at 0x4004f1: file remote.c, line 6. |

Запустим программу :

|  |
| --- |
| (gdb) run  Starting program: /home/ubuntu/5/1/remote  Breakpoint 1, main () at remote.c:6  6 int a[] = {1, 2, 3};  (gdb) list  1 #include <unistd.h>  2 #include <sys/types.h>  3 #include <stdio.h>  4 int main()  5 {  6 int a[] = {1, 2, 3};  7 /\* int i=0;  8 /\* printf("pid=%d\n", getpid());  9 while (1) {  10 sleep(1);  (gdb) next  14 return 0; |

Просмотрим содержимое массива

|  |
| --- |
| (gdb) print a  $1 = {1, 2, 3} |

Посмотрим какой тип а

|  |
| --- |
| (gdb) ptype a  type = int [3] |

Выведем 12 элементов по адресу а

|  |
| --- |
| (gdb) x/12xb &a  0x7fffffffddf0: 0x01 0x00 0x00 0x00 0x02 0x00 0x00 0x00  0x7fffffffddf8: 0x03 0x00 0x00 0x00 |

Модифицируем программу

|  |
| --- |
| #include <unistd.h>  #include <sys/types.h>  #include <stdio.h>  int main()  {  //int a[] = {1, 2, 3};  int i=0;  printf("pid=%d\n", getpid());  while (1) {  sleep(1);  printf("i=%d\n",i);    }  return 0;  } |

|  |
| --- |
| ubuntu@ubuntu-K53SJ:~/5/1$ ./remote  pid=7012  i=0  i=0  i=0  i=0  i=0  i=0  i=0  i=0  i=0  i=0  i=0 |

Откроем второй терминал и запустим в нем отладчик.

Подключемся к процессу

|  |
| --- |
| ubuntu@ubuntu-K53SJ:~$ sudo gdb -p 7012  [sudo] password for ubuntu:  … |

Можно заметить, что выполнение программы приостановилось.

|  |
| --- |
| (gdb) list  1 #include <unistd.h>  2 #include <sys/types.h>  3 #include <stdio.h>  4 int main()  5 {  6 //int a[] = {1, 2, 3};  7 int i=0;  8 printf("pid=%d\n", getpid());  9 while (1) {  10 sleep(1);  (gdb)  11 printf("i=%d\n",i);  12  13 }  14 return 0;  15 } |

Создадим точку останова:

|  |
| --- |
| (gdb) b 11  Breakpoint 1 at 0x8048af: file remote.c, line 11 |

Продолжим выполнение :

|  |
| --- |
| (gdb) c  Continuing. |

Программа снова пришла в точку останова

|  |
| --- |
| Breakpoint 1, main() at remote.c:11  11 printf("i=%d\n",i); |

Установим для переменной i новое значение

|  |
| --- |
| (gdb) set var i=27600  (gdb) c  Continuing.  Breakpoint 1, main() at remote.c:11  11 printf("i=%d\n",i); |

В другом терминале видно, новое значение переменной i

|  |
| --- |
| i=0  i=0  i=0  i=27600 |

Удалим котрольную точку, и отключимся от рограммы :

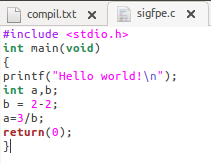
|  |
| --- |
| (gdb) d 1  (gdb) detach |

Программа продолжает функционировать :

|  |
| --- |
| i=27600  i=27600  i=27600  i=27600  i=27600 … |

Рассмотрим сигнал, посылаемый процессу, при попытке выполнения ошибочной арифметической операции.

Код программы



|  |
| --- |
| ubuntu@ubuntu-K53SJ:~/5/21$ gdb ./sigfpe  (gdb) r  Starting program: /home/ubuntu/5/21/sigfpe  Hello world!  Program received signal SIGFPE, Arithmetic exception.  0x000000000040054c in main ()  (gdb) bt  #0 0x000000000040054c in main ()  (gdb) catch syscall rt\_sigaction  Catchpoint 1 (syscall 'rt\_sigaction' [13]) |

В данном примере мы указали, что хотим отслеживать все вызовы функции rt\_sigaction(). Таким же образом можно отслеживать все остальные вызовы. Подобным образом можно так же отслеживать любые системные события.

Отладчик gdb предоставляет широкие возможности для отладки своих программ и диагностирования незнакомых программ.

Отладчик gdb имеет так же версию для Microsoft Windows, значит его так же можно использовать для отладки программ в ОС Windows.

valgrind

valgrind – инструментальное программное обеспечение, предназначенное для отладки использования памяти, обнаружения утечек памяти, а также профилирования. Данное ПО содержит несколько встроенных утилит для отладки и профилирования программ. Один из них – инструмент memcheck, который позволяет контролировать использование памяти в программе и искать ее утечки.

|  |
| --- |
| ubuntu@ubuntu-K53SJ:~/5/1$ gcc -std=c99 -D\_POSIX\_C\_SOURCE=200112L -g -D\_XOPEN\_SOURCE=700 -rdynamic -o signals all\_signals.c  ubuntu@ubuntu-K53SJ:~/5/1$ valgrind --tool=memcheck ./signals  ==5834== Memcheck, a memory error detector  ==5834== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.  ==5834== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info  ==5834== Command: ./signals  ==5834==  Obtained 5 stack frames.  ./signals(print\_trace+0x19) [0x400936]  ./signals(dummy\_function+0x9) [0x4009b8]  ./signals(main+0x9) [0x4009c3]  /lib/x86\_64-linux-gnu/libc.so.6(\_\_libc\_start\_main+0xf5) [0x4e58f45]  ./signals() [0x400859]  ==5834==  ==5834== HEAP SUMMARY:  ==5834== in use at exit: 0 bytes in 0 blocks  ==5834== total heap usage: 6 allocs, 6 frees, 2,038 bytes allocated  ==5834==  ==5834== All heap blocks were freed -- no leaks are possible  ==5834==  ==5834== For counts of detected and suppressed errors, rerun with: -v  ==5834== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0) |

Используя эту утилиту можно контролировать использование программой памяти.

Еще одним инструментом valgrind является callgrind, который работать с графом вызовов функций. В основном он используется для профилирования программ, т.к. позволяет понять в какой точке программа тратит больше всего времени. Callgrind формирует отчет в специальном формате и сохраняет его в специальном файле. Этот файл затем можно открыть какой-либо другой программой для анализа результатов. Одной из таких программ является KCachegrind.

|  |
| --- |
| ubuntu@ubuntu-K53SJ:~/5/1$ valgrind --tool=callgrind ./signals  ==10694== Callgrind, a call-graph generating cache profiler  ==10694== Copyright (C) 2002-2013, and GNU GPL'd, by Josef Weidendorfer et al.  ==10694== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info  ==10694== Command: ./signals  ==10694==  ==10694== For interactive control, run 'callgrind\_control -h'.  Obtained 5 stack frames.  ./signals(print\_trace+0x19) [0x400936]  ./signals(dummy\_function+0x9) [0x4009b8]  ./signals(main+0x9) [0x4009c3]  /lib/x86\_64-linux-gnu/libc.so.6(\_\_libc\_start\_main+0xf5) [0x4c48f45]  ./signals() [0x400859]  ==10694==  ==10694== Events : Ir  ==10694== Collected : 269253  ==10694==  ==10694== I refs: 269,253 |

Интерфейс программы KCachegrind

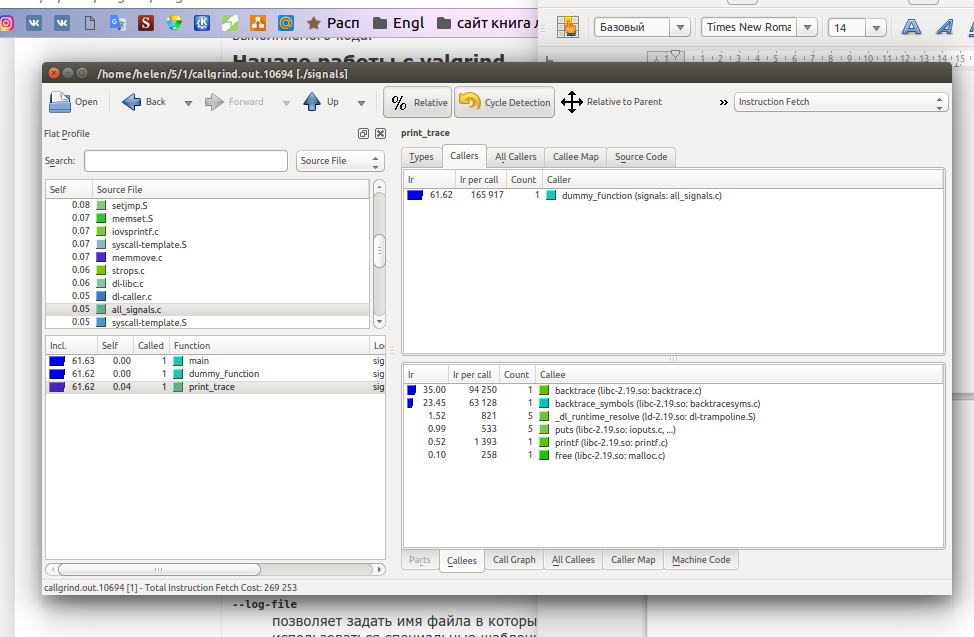


Рис 1 KCachegrind

Valgrind обладает большим количеством инструментов, поэтому с помощью него достаточно просто производить динамическую отладку и диагностирование программы.

dmesg

Во время загрузки системы, пока загружается ядро, система инициализации, драйвера, модули ядра, инициализируется оборудование на экран выводится большое количество сообщений, которые отображают информацию о состоянии ядра, процессе загрузки, а также состоянии устройств.

Именно на этапе загрузки может происходить много ошибок в сервисах или оборудовании. Просмотр этих сообщений может быть очень полезным.Но они пробегают очень быстро и мы не всегда можем успеть их прочитать. Но все они могут быть просмотрены с помощью команды dmesg.

Выполнив команду dmesg с правами суперпользователя видны все сообщения, которые выводило ядро во время загрузки. Здесь видно очень много полезной информации.

|  |
| --- |
| ubuntu@ubuntu-K53SJ:~/5/1$ dmesg | more  [ 0.000000] Initializing cgroup subsys cpuset  [ 0.000000] Initializing cgroup subsys cpu  [ 0.000000] Initializing cgroup subsys cpuacct  [ 0.000000] Linux version 4.4.0-53-generic (buildd@lgw01-18) (gcc version 4.8.4 (Ubuntu 4.8.4-2ubuntu1~14.04.3) ) #74~14.04.1-Ubuntu SMP Fri Dec 2 03:43:  31 UTC 2016 (Ubuntu 4.4.0-53.74~14.04.1-generic 4.4.30)  [ 0.000000] Command line: BOOT\_IMAGE=/boot/vmlinuz-4.4.0-53-generic root=UUID=8fe96e92-74fc-48e6-81ac-cb330331b3d8 ro quiet splash vt.handoff=7  [ 0.000000] KERNEL supported cpus:  [ 0.000000] Intel GenuineIntel  [ 0.000000] AMD AuthenticAMD  [ 0.000000] Centaur CentaurHauls  [ 0.000000] x86/fpu: xstate\_offset[2]: 576, xstate\_sizes[2]: 256  [ 0.000000] x86/fpu: Supporting XSAVE feature 0x01: 'x87 floating point registers'  [ 0.000000] x86/fpu: Supporting XSAVE feature 0x02: 'SSE registers'  [ 0.000000] x86/fpu: Supporting XSAVE feature 0x04: 'AVX registers'  [ 0.000000] x86/fpu: Enabled xstate features 0x7, context size is 832 bytes, using 'standard' format.  [ 0.000000] x86/fpu: Using 'eager' FPU context switches.  [ 0.000000] e820: BIOS-provided physical RAM map:  [ 0.000000] BIOS-e820: [mem 0x0000000000000000-0x000000000009e7ff] usable  [ 0.000000] BIOS-e820: [mem 0x000000000009e800-0x000000000009ffff] reserved  [ 0.000000] BIOS-e820: [mem 0x00000000000e0000-0x00000000000fffff] reserved  [ 0.000000] BIOS-e820: [mem 0x0000000000100000-0x000000001fffffff] usable  [ 0.000000] BIOS-e820: [mem 0x0000000020000000-0x00000000201fffff] reserved  [ 0.000000] BIOS-e820: [mem 0x0000000020200000-0x000000003fffffff] usable  [ 0.000000] BIOS-e820: [mem 0x0000000040000000-0x00000000401fffff] reserved  [ 0.000000] BIOS-e820: [mem 0x0000000040200000-0x00000000aace4fff] usable  [ 0.000000] BIOS-e820: [mem 0x00000000aace5000-0x00000000aacf7fff] reserved  [ 0.000000] BIOS-e820: [mem 0x00000000aacf8000-0x00000000aacf8fff] usable  [ 0.000000] BIOS-e820: [mem 0x00000000aacf9000-0x00000000aad8dfff] reserved  [ 0.000000] BIOS-e820: [mem 0x00000000aad8e000-0x00000000aad8efff] usable  [ 0.000000] BIOS-e820: [mem 0x00000000aad8f000-0x00000000aadb7fff] reserved  [ 0.000000] BIOS-e820: [mem 0x00000000aadb8000-0x00000000aadc7fff] usable  [ 0.000000] BIOS-e820: [mem 0x00000000aadc8000-0x00000000aade7fff] reserved  … |

Просмотр памяти

С помощью dmesg можно посмотреть количество доступной в системе памяти:

|  |
| --- |
| ubuntu@ubuntu-K53SJ:~/5/1$ dmesg | grep Memory  [ 0.000000] Memory: 3929904K/4104044K available (8205K kernel code, 1294K rwdata, 3976K rodata, 1488K init, 1292K bss, 174140K reserved, 0K cma-reserved)  [ 25.135460] [drm] Memory usable by graphics device = 2048M |

Просмотр состояния сетевых адаптеров

Поскольку в dmesg сохраняются все сообщения о состоянии устройств, возможно посмотреть состояние сетевого адаптера, или, возможно, ошибки которые возникли при его инициализации:

|  |
| --- |
| ubuntu@ubuntu-K53SJ:~/5/1$ dmesg | grep eth  [ 1.152232] r8169 0000:04:00.0 eth0: RTL8168e/8111e at 0xffffc900007f8000, bc:ae:c5:65:22:c0, XID 0c200000 IRQ 25  [ 1.152236] r8169 0000:04:00.0 eth0: jumbo features [frames: 9200 bytes, tx checksumming: ko]  [ 25.369118] VGA switcheroo: detected Optimus DSM method \\_SB\_.PCI0.PEGR.GFX0 handle  [ 31.931370] r8169 0000:04:00.0 eth0: link down  [ 31.931383] r8169 0000:04:00.0 eth0: link down  [ 31.931428] IPv6: ADDRCONF(NETDEV\_UP): eth0: link is not ready  [ 33.486320] r8169 0000:04:00.0 eth0: link up  [ 33.486347] IPv6: ADDRCONF(NETDEV\_CHANGE): eth0: link becomes ready |

Просмотр ошибок dmesg

Просмотрим ошибки, произошедшие во время загрузки или работы системы:

|  |
| --- |
| ubuntu@ubuntu-K53SJ:~/5/1$ dmesg | grep error  [ 24.160922] EXT4-fs (sda3): re-mounted. Opts: errors=remount-ro |

Сообщения, выводимые при загрузке ядра ОС, могут быть изучены и после загрузки ОС благодаря наличию утилиты dmesg. Ввиду того, что дисковые устройства определяются ядром ОС в процессе загрузки системы, вы также можете использовать утилиту dmesg для поиска информации о дисковых устройствах.

|  |
| --- |
| ubuntu@ubuntu-K53SJ:~/5/1$ dmesg | grep 'sd[a-z]' | head  [ 1.523981] sd 0:0:0:0: [sda] 976773168 512-byte logical blocks: (500 GB/466 GiB)  [ 1.523984] sd 0:0:0:0: [sda] 4096-byte physical blocks  [ 1.524022] sd 0:0:0:0: [sda] Write Protect is off  [ 1.524024] sd 0:0:0:0: [sda] Mode Sense: 00 3a 00 00  [ 1.524037] sd 0:0:0:0: [sda] Write cache: enabled, read cache: enabled, doesn't support DPO or FUA  [ 1.559761] sda: sda1 sda2 sda3 sda4  [ 1.560402] sd 0:0:0:0: [sda] Attached SCSI disk  [ 2.250301] EXT4-fs (sda3): mounting ext3 file system using the ext4 subsystem  [ 2.352484] EXT4-fs (sda3): mounted filesystem with ordered data mode. Opts: (null)  [ 24.160922] EXT4-fs (sda3): re-mounted. Opts: errors=remount-ro |

**Windows**

Так же одним из наиболее популярных отладчиков для Windows является встроенный отладчик Visual Studio. Он позволяет получить огромное количество информации: производительность программы, стек вызовов, значения переменных и т.д.

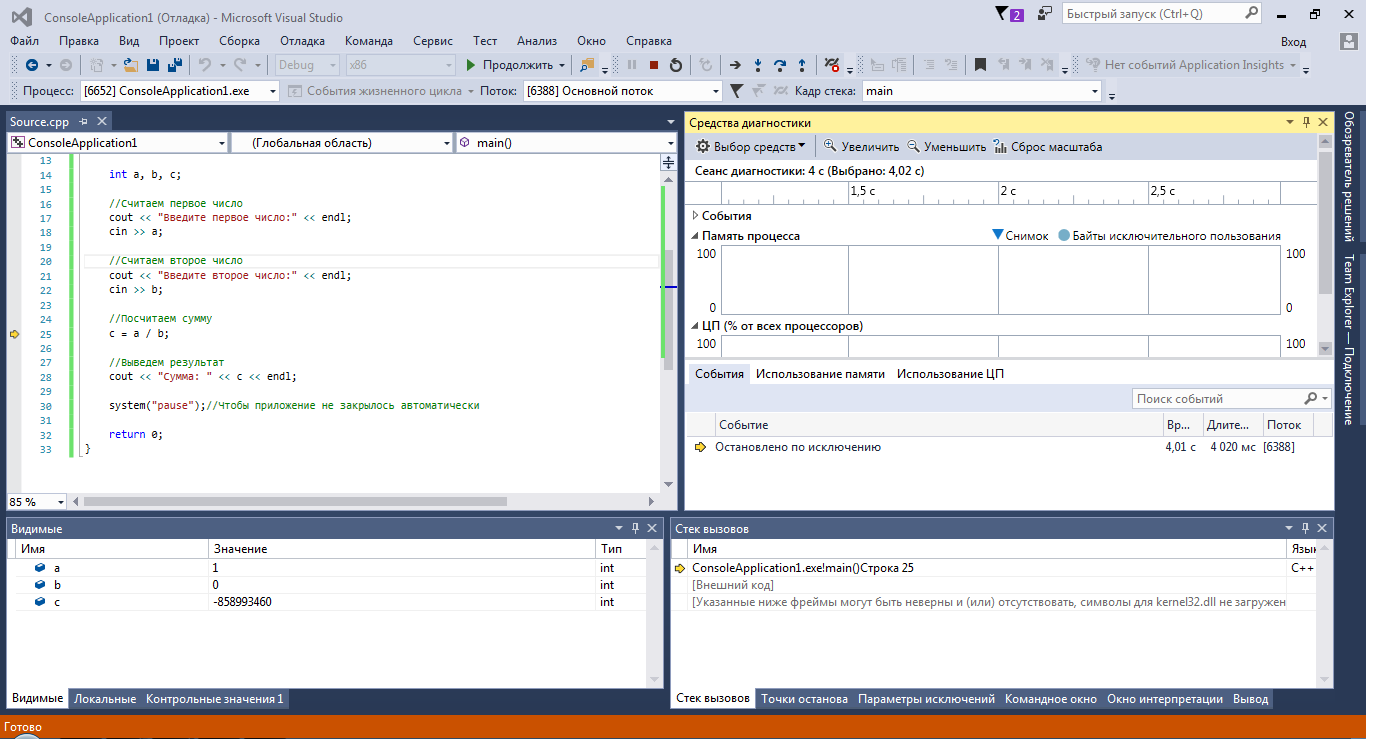


Рис 2 Отладчик VisualStudio

Еще одним отладчиком для Windows является достаточно новый инструмент Dr. Memory. Это отладчик, в котором так же есть возможность просмотра системных вызовов и возможности, схожие с Memcheck. Если просто запустить какую-то программу в отладчике, он просто сформирует отчет с возникшими ошибками.

Рассмотрим его на примере следующей программы :

|  |
| --- |
| #include <iostream>  using namespace std;//Подключение стандартной библиотеки функций  int main()  {  setlocale(LC\_ALL, "RUSSIAN");//Для корректного отображения русского языка  //Расскоментировать строчки ниже, если с русским будут проблемы  //setlocale(LC\_ALL, "ru\_RU.UTF-8");  //setlocale(LC\_ALL, "");  int a, b, c;  //Считаем первое число  cout << "Введите первое число:" << endl;  cin >> a;  //Считаем второе число  cout << "Введите второе число:" << endl;  cin >> b;  //Посчитаем сумму  c = a / b;  //Выведем результат  cout << "Сумма: " << c << endl;  system("pause");//Чтобы приложение не закрылось автоматически  return 0;  } |

Скомпилируем код и запустим Dr. Memory

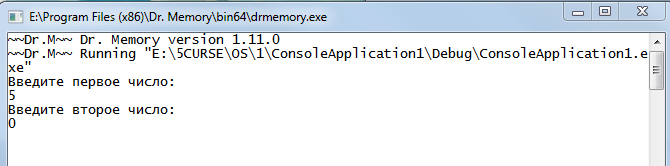
****

Рис 3 Dr. Memory

**Содержимое файла results.txt**

|  |
| --- |
| Dr. Memory version 1.11.0 build 2 built on Aug 29 2016 02:42:07  Dr. Memory results for pid 6556: "ConsoleApplication1.exe"  Application cmdline: "E:\5CURSE\OS\1\ConsoleApplication1\Debug\ConsoleApplication1.exe"  Recorded 115 suppression(s) from default E:\Program Files (x86)\Dr. Memory\bin\suppress-default.txt  ===========================================================================  FINAL SUMMARY:  DUPLICATE ERROR COUNTS:  SUPPRESSIONS USED:  NO ERRORS FOUND:  0 unique, 0 total unaddressable access(es)  0 unique, 0 total uninitialized access(es)  0 unique, 0 total invalid heap argument(s)  0 unique, 0 total GDI usage error(s)  0 unique, 0 total handle leak(s)  0 unique, 0 total warning(s)  0 unique, 0 total, 0 byte(s) of leak(s)  0 unique, 0 total, 0 byte(s) of possible leak(s)  ERRORS IGNORED:  2 potential error(s) (suspected false positives)  (details: C:\Users\А\AppData\Roaming\Dr. Memory\DrMemory-ConsoleApplication1.exe.6556.000\potential\_errors.txt)  4 potential leak(s) (suspected false positives)  (details: C:\Users\А\AppData\Roaming\Dr. Memory\DrMemory-ConsoleApplication1.exe.6556.000\potential\_errors.txt)  60 unique, 183 total, 9442 byte(s) of still-reachable allocation(s)  (re-run with "-show\_reachable" for details)  Details: C:\Users\А\AppData\Roaming\Dr. Memory\DrMemory-ConsoleApplication1.exe.6556.000\results.txt |

Еще одним из инструментов отладки в Windows является отладчик из пакета Windows Kits называемый windbg. Он обладает схожими возможностями с остальными отладчиками: позволяет получать stacktrace, устанавливать точки останова, смотреть таблицу символов и т.д.

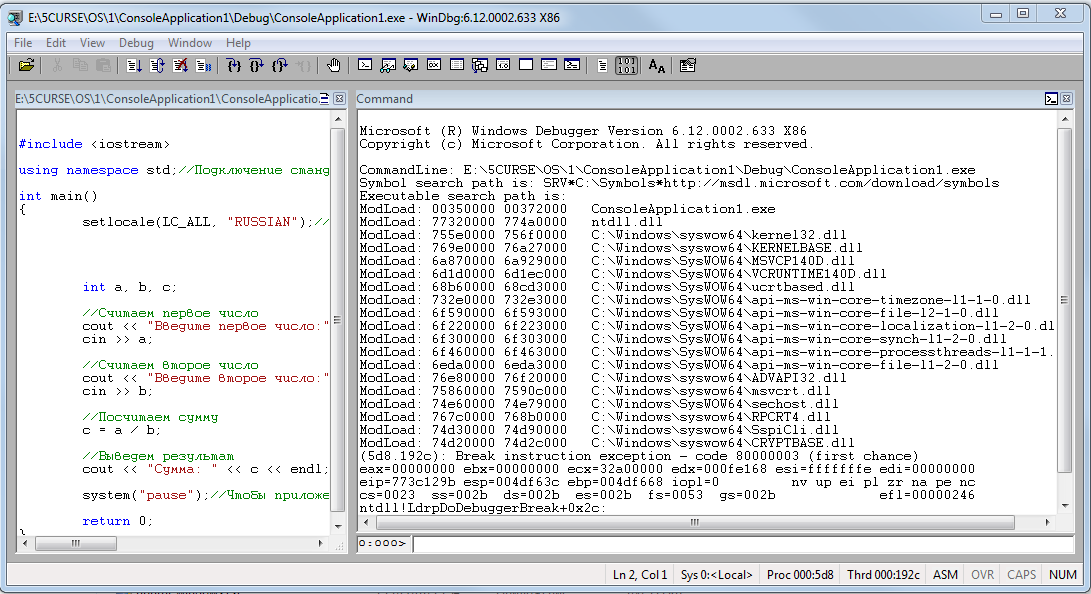


Рис. 4. WinDbg

**Вывод**

Отладчик Visual Studio позволяет вести наблюдение за поведением программы во время выполнения и выявлять проблемы. Отладчик работает со всеми языками программирования Visual Studio и библиотеками. С помощью отладчика можно прерывать или приостанавливать выполнение программы с целью проверки кода, вычислять и редактировать значения переменных программы, отслеживать состояние регистров, просматривать инструкции, созданные из исходного кода, а также просматривать область памяти, используемую приложением. Dr. Memory. обладает достаточно большим набором возможностей и удобен в использовании. WinDbg — позволяет отлаживать 32/64 битные приложения пользовательского уровня, драйвера, может быть использован для анализа аварийных дампов памяти, WinDbg поддерживает автоматическую загрузку отладочных символов, имеется встроенный скриптовый язык для автоматизации процесса отладки

**Список использованных источников**

1. Man strace:

<https://linux.die.net/man/1/strace>

1. Man backtrace:

<http://man7.org/linux/man-pages/man3/backtrace_symbols.3.html>

1. Man syslog:

<https://linux.die.net/man/3/syslog>

1. Сайт gdb:

<https://www.gnu.org/software/gdb/>

1. Сайт valgrind:

http://valgrind.org/

1. Сайт техподдержки Ubuntu: <http://help.ubuntu.ru/wiki/%D1%81%D0%B1%D0%BE%D1%80%D0%BA%D0%B0_%D1%8F%D0%B4%D1%80%D0%B0>
2. Сайт ядра linux:

<https://www.kernel.org/>

1. Сайт dr.memory:

<http://www.drmemory.org/>