Исследование качества программ

Для реализации словаря из предыдущей лабораторной работы, необходимо провести исследование скорости выполнения и потребления оперативной памяти. В случае выявления ошибок или явных недочётов, требуется их исправить.

Результатом лабораторной работы является отчёт, состоящий из:

- Дневника выплонения работы, в котором отражено что и когда делалось, какие средства использовались и какие результаты были достигнуты на каждом шаге выполнения лабораторной работы.
- Выводов о найденных недочётах.
- Сравнение работы исправленной программы с предыдущей версией.
- Общих выводов о выполнении лабораторной работы, полученном опыте.

Минимальный набор используемых средств должен содержать утилиту gprof и библиотеку dmalloc, однако их можно заменять на любые другие аналогичные или более развитые утилиты (например, Valgrind или Shark) или добавлять к ним новые (например, gcov).

Дополнительная информация

gprof

Утилиту gprof следует изучить самостоятельно по оперативной документации операционной системы Unix.

Библиотека dmalloc

Документация

Ниже следует краткое описание ее использования для небольшой тестовой программы.

Допустим, написана небольшая программа, имеющая две ошибки использования динамической памяти, которые не могут быть проверены на этапе компиляции, а только на этапе времени выполнения:

```
1:  #include <stdio.h>
2:  #include <stdlib.h>
```

```
 3:
 4:  #ifdef DMALLOC
 5:  #include "dmalloc.h"
 6:  #endif
 7:
 8:  void *foo() {
 9:    char *p = malloc(1023);
10:    p[1023] = '\0';
11:    return p;
12:  }
13:
14:  int main() {
15:    foo();
16:    return 0;
17:  }
```

В программе есть две ошибки:

1. В 10-й строке происходит обращение к 1024-му элементу динамического массива p, при том что было выделено 1023 элемента. При выполнении программы эта ошибка, скорее всего, не будет обнаружена, потому что реально размер выделенной памяти будет кратен размеру страницы, то есть будет заведомо больше 1023 элементов.
2. Нигде в программе не уничтожается память, которая была выделена в 9-й строке.

Если скомпилировать и запустить программу, то скорее всего никаких проблем ни при компиляции, ни на этапе выполнения программы, не возникнет:

```
$ gcc -Wall dmalloc-test.c -o dtest
$ ./dtest
$
```

Для использования библиотеки dmalloc нужно включить в текст программы специальный файл dmalloc.h, подменяющий вызовы функций аллокации памяти на свои, и определить макрос DMALLOC для включения основной функциональности библиотеки. Кроме того, можно еще определить макрос DMALLOC_FUNC_CHECK для включения дополнительной функциональности библиотеки по проверке корректности параметров различных функций из стандартной библиотеки языка C. Наконец, при компоновке программы, нужно подключить к сборке библиотеку dmalloc.

Таким образом, компиляция программы может быть произведена следующим образом:

```
$ gcc -Wall -DDMALLOC -DDMALLOC_FUNC_CHECK \
     -I/usr/local/include -L/usr/local/lib dmalloc-test.c \
     -o dtest -ldmalloc
```

Перед запуском исследуемой программы нужно определить переменную

среды DMALLOC_OPTIONS в которую передать информацию о том, как должна работать библиотека dmalloc. Полный список опций и их расшифровку следует искать в документации к библиотеке. Для того, чтобы в командном интерпретаторе bash правильно установить окружение среды, можно воспользоваться следующей командой:

```
eval `command dmalloc -b -l logfile -i 100 low`
```

При этом отчет о проверках будет писаться в файл logfile.

Теперь, если запустить программу, то она завершиться аварийно:

```
$ ./dtest
debug-malloc library: dumping program, fatal error
   Error: failed OVER picket-fence magic-number check (err 27)
Abort trap: 6 (core dumped)
```

Тем самым, библиотека dmalloc «поймала» обращение за пределы выделенной памяти; аварийное же завершение программы позволило создать посмертный дамп памяти, который можно исследовать при помощи отладчика gdb.

Если модифицировать исходную программу, исправив обращение за пределы выделенной памяти (допустим, заменив 1023-й индекс на 1022-й), то программа отработает нормально. Отчет о работе будет выглядеть примерно следующим образом:

```
1192785155: 1: Dmalloc version '5.5.2' from 'http://dmalloc.com/'
1192785155: 1: flags = 0x4e48503, logfile 'logfile'
1192785155: 1: interval = 100, addr = 0, seen # = 0, limit = 0
1192785155: 1: starting time = 1192785155
1192785155: 1: process pid = 23403
1192785155: 1: Dumping Chunk Statistics:
1192785155: 1: basic-block 8192 bytes, alignment 8 bytes
1192785155: 1: heap address range: 0x16005a000 to 0x160060000, 24576 bytes
1192785155: 1:     user blocks: 1 blocks, 7167 bytes (29%)
1192785155: 1:    admin blocks: 2 blocks, 16384 bytes (66%)
1192785155: 1:    total blocks: 3 blocks, 24576 bytes
1192785155: 1: heap checked 1
1192785155: 1: alloc calls: malloc 1, calloc 0, realloc 0, free 0
1192785155: 1: alloc calls: recalloc 0, memalign 0, valloc 0
1192785155: 1: alloc calls: new 0, delete 0
1192785155: 1:   current memory in use: 1023 bytes (1 pnts)
1192785155: 1:  total memory allocated: 1023 bytes (1 pnts)
1192785155: 1:  max in use at one time: 1023 bytes (1 pnts)
1192785155: 1: max alloced with 1 call: 1023 bytes
1192785155: 1: max unused memory space: 1025 bytes (50%)
1192785155: 1: top 10 allocations:
1192785155: 1:  total-size  count in-use-size  count  source
1192785155: 1:        1023      1        1023      1  dmalloc-test.c:9
1192785155: 1:        1023      1        1023      1  Total of 1
1192785155: 1: Dumping Not-Freed Pointers Changed Since Start:
1192785155: 1:  not freed: '0x16005b808|s1' (1023 bytes) from 'dmalloc-test.c:9'
1192785155: 1:  total-size  count  source
1192785155: 1:        1023      1  dmalloc-test.c:9
```

```
1192785155: 1:        1023      1  Total of 1
1192785155: 1: ending time = 1192785155, elapsed since start = 0:00:00
```

В конце отчета имеется секция «Dumping Not-Freed Pointers», в которой видно, что один блок памяти, размером 1023 байта, выделенный в 9-й строке, не был освобожден до конца работы программы. Таким образом можно отследить утечки памяти в своих программах.

Перед выполнением лабораторной работы настоятельно рекомендуется самостоятельно изучить документацию к библиотеке dmalloc. Использование дополнительных опций, не описанных выше, улучшит впечатление от выполненной работы.