Московский авиационный институт (национальный исследовательский университет)

Институт №8 «Компьютерные науки и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа по курсу "Дискретный анализ" №3

Студент: Клименко В. М.

Преподаватель: Макаров Н. К.

Группа: М8О-203Б-22

Дата: _____

Оценка: _____

Подпись: _____

Оглавление

Цель работы

Изучить утилиты для проверки корректной работы с памятью

Дневник выполнения работы

Для диагностики кода я использовал такую утилиту как `gprof`, а для проверки работы с памятью я использовал такие утилиты как `dmalloc` и `valgrind`.

Для тестирования я написал небольшую программу на питоне, которая генерирует тестовый файл. В нем создается список со словами, потом над ними выполняются случайные команды из лабораторной работы:

```
import random
import string

LINE_COUNT = 50000
WORD_COUNT = 500

TREE_FILE_PATH = './test.avl'

words_list = []

for _ in range(WORD_COUNT):
      word = ''.join(random.choices(string.ascii_lowercase,
k=random.randint(1, 256)))
      words_list.append(word)

output_file = open('test.txt', 'w')

for _ in range(LINE_COUNT):
      random_word = words_list[random.randint(0, WORD_COUNT - 1)]
      command = random.randint(0, 99)
      if 0 <= command < 40: # + word number
      output_file.write(f'+ {random_word} {random.randint(0, 2**64 -
1)}\n')
      if 40 <= command < 50: # - word
      output_file.write(f'- {random_word}\n')
      if 50 <= command < 70: # word
```

```
        output_file.write(f'{random_word}\n')
        if 80 <= command < 90: # save
        output_file.write(f'! Save {TREE_FILE_PATH}\n')
        if 90 <= command < 100: # load
        output_file.write(f'! Load {TREE_FILE_PATH}\n')

output_file.close()
```

В среднем на 100 команд должно происходить 40 вставок, 10 удалений, 20 поисков, 10 сохранений и 10 загрузок

Диагностика

gprof

В первом отчете `gprof` я получил, что количество правых поворотов деревьев было намного меньше чем левых, для некоторых тестов правых поворотов вообще не происходило, что неожиданно, так как если выполняются случайные команды для случайных слов, то ожидается примерно равное количество левых и правых поворотов:

```
Flat profile:


Each sample counts as 0.01 seconds.
  %   cumulative   self              self  total
 time   seconds   seconds   calls  us/call  us/call  name
 41.67     0.38  0.38  1920955    0.20  0.31  _avl_insert
 10.00     0.47  0.09 98071041    0.00  0.00  node_height
  8.89     0.55  0.08  1935837    0.04  0.04  str_to_lower
  7.78     0.61  0.07 18502782    0.00  0.01  node_balance
  6.67     0.68  0.06  3841910    0.02  0.02  key_value_new
  6.67     0.73  0.06  4926       12.18 12.18  _avl_save_to_path
  5.00     0.78  0.04 16650415    0.00  0.01  avl_rebalance
  3.33     0.81  0.03  1920955    0.02  0.38  avl_insert
  3.33     0.84  0.03  4935       6.08  6.08  node_free
```

```
  3.33     0.87  0.03  4934       6.08 158.31  avl_load_from_path
  2.22     0.89  0.02                          _init
  1.11     0.90  0.01  1905643    0.01  0.01   node_new
  0.00     0.90  0.00  1852365    0.00  0.01   avl_rotate_left
  0.00     0.90  0.00  9931       0.00  0.00   _avl_find
  0.00     0.90  0.00  9931       0.00  0.04   avl_find
  0.00     0.90  0.00  4952       0.00  0.14   avl_delete
  0.00     0.90  0.00  4951       0.00  0.10   _avl_delete
  0.00     0.90  0.00  4935       0.00  6.08   avl_free
  0.00     0.90  0.00  4935       0.00  0.00   avl_new
  0.00     0.90  0.00  4926       0.00  12.18   avl_save_to_path
  0.00     0.90  0.00        7    0.00  0.01   avl_rotate_right
```

Как оказалось в программе есть несколько ошибок, из-за которых это происходит:

1. Макросы (конечно же)

В программе был написан макрос для поиска максимума между двумя переменными:

```
#define max(a, b) (a) > (b) ? (a) : (b)
```

Я надеялся, что знал что делаю… Даже скобки вокруг переменных поставил, потому что знал что с этим могут возникнуть проблемы. Вот одно из мест, в котором я вызываю этот макрос:

```
current->height = max(node_height(current->left),
node_height(current->right)) + 1;
```

То есть высота вершины приравнивается к сумме максимума высот детей и единицы… По крайней мере, так задумывалось…

Если раскрыть макрос и посмотреть, что происходит на самом деле то можно увидеть, что прибавленная единица попадает во вторую ветку условного оператора (для наглядности и оптимизации я вынес высоты детей в отдельные переменные):

```
int64_t left_height = node_height(current->left);
int64_t right_height = node_height(current->right);
current->height = (left_height) > (right_height) ?
(left_height) : (right_height) + 1
```

Чтобы этого избежать “имплементацию” макроса нужно обернуть в скобки:

```
#define max(a, b) ((a) > (b) ? (a) : (b))
```

Также, чтобы убрать повторяющийся код и немного его оптимизировать (по отчету `gprof` видно, что `node_height` занимает немаленькое время исполнения, а кол-во его вызовов множится из-за макроса), создам новую функцию, которая бы обновляла высоту переданной вершины:

```
int64_t node_update_height(node *current) {
    if (current == NULL) return 0;

    int64_t left_height = node_height(current->left);
    int64_t right_height = node_height(current->right);
    current->height = max(left_height, right_height) + 1;

    return current->height;
}
```

По окончании этих модификаций, вызовов `avl_rotate_right` больше не стало, однако вызовов `node_height` стало меньше на 20%:

```
Flat profile:


Each sample counts as 0.01 seconds.
%   cumulative   self              self  total
time   seconds   seconds    calls  us/call  us/call  name
29.37      0.37  0.37  1993586    0.19  0.40  _avl_insert
9.92       0.49  0.12 21196846    0.01  0.01  node_update_height
8.73       0.60  0.11  3987172    0.03  0.03  key_value_new
7.94       0.70  0.10 80938396    0.00  0.00  node_height
7.94       0.81  0.10  2008469    0.05  0.05  str_to_lower
```

```
5.95       0.88  0.07 19272352   0.00  0.01  node_balance
5.56       0.95  0.07  4926       14.21 14.21  _avl_save_to_path
4.37       1.00  0.06 17347858   0.00  0.01  avl_rebalance
3.97       1.05  0.05  1977660   0.03  0.03  node_new
3.97       1.10  0.05  4935       10.13 10.13  node_free
3.97       1.16  0.05  4934     10.13   222.55  avl_load_from_path
2.38       1.19  0.03  1993586   0.02  0.49  avl_insert
1.98       1.21  0.03  1924494   0.01  0.03  avl_rotate_left
1.59       1.23  0.02  4935       4.05  4.05  avl_new
0.79       1.24  0.01  4952       2.02  2.19  _avl_delete
0.79       1.25  0.01                         _init
0.79       1.26  0.01                         main
0.00       1.26  0.00  9931       0.00  0.00  _avl_find
0.00       1.26  0.00  9931       0.00  0.05  avl_find
0.00       1.26  0.00  4952       0.00  2.24  avl_delete
0.00       1.26  0.00  4935       0.00  10.13  avl_free
0.00       1.26  0.00  4926       0.00  14.21  avl_save_to_path
```

2. Неверная высота NULL’овой вершины

Здесь все понятно: для NULL’овой вершины высота должна быть на одну меньше, чем у листа, однако сейчас это не так: у новой высота 0, и у NULL’овой — 0, если же сделать высоту NULL’овой вершины -1, получим следующий результат:

```
Flat profile:


Each sample counts as 0.01 seconds.
%   cumulative   self            self  total
time   seconds   seconds   calls  us/call  us/call  name
35.92      0.37  0.37  1997054   0.19  0.35  _avl_insert
8.74       0.46  0.09 73419426   0.00  0.00  node_height
```

```
 8.74       0.55  0.09 17387981    0.01  0.01  node_balance
 7.77       0.63  0.08  3994108    0.02  0.02  key_value_new
 7.77       0.71  0.08  2011937    0.04  0.04  str_to_lower
 6.80       0.78  0.07 19321732    0.00  0.01  node_update_height
 4.85       0.83  0.05  4934  10.13   188.50  avl_load_from_path
 4.85       0.88  0.05  4926  10.15 10.15  _avl_save_to_path
 3.88       0.92  0.04 15454758    0.00  0.01  avl_rebalance
 2.91       0.95  0.03  1981118    0.02  0.02  node_new
 2.91       0.98  0.03  4935  6.08  6.08  node_free
 1.94       1.00  0.02  1997054    0.01  0.41  avl_insert
 1.94       1.02  0.02  4935  4.05  4.05  avl_new
 0.97       1.03  0.01                          main
 0.00       1.03  0.00  1933210    0.00  0.01  avl_rotate_left
 0.00       1.03  0.00  9931  0.00  0.00  _avl_find
 0.00       1.03  0.00  9931  0.00  0.04  avl_find
 0.00       1.03  0.00  4952  0.00  0.15  _avl_delete
 0.00       1.03  0.00  4952  0.00  0.19  avl_delete
 0.00       1.03  0.00  4935  0.00  6.08  avl_free
 0.00       1.03  0.00  4926  0.00  10.15  avl_save_to_path
 0.00       1.03  0.00       277   0.00  0.01  avl_rotate_right
```

Вызовов функции стало действительно больше, но все равно этого мало. Почему же? На самом деле, теперь в коде все верно, а большое количество поворотов влево происходит из-за формата хранения AVL дерева в файле. Сейчас все значения хранятся по порядку, из-за чего и происходят постоянные повороты влево, так как каждое следующее значение будет идти в самый правый лист. Для проверки этой теории будем сохранять значения вершин в обратном порядке:

```c
void _avl_save_to_path(node *current, FILE *file) {
     if (current == NULL) return;
```

```
        _avl_save_to_path(current->right, file);
        fprintf(file, "k: %s v: %"PRIu64"\n", current->value.key,
 current->value.value);
        _avl_save_to_path(current->left, file);
 }
```

Отчет gprof:

```
Flat profile:


Each sample counts as 0.01 seconds.
  %   cumulative   self              self  total
 time   seconds   seconds   calls  us/call  us/call  name
 39.13     0.36  0.36  1993578    0.18  0.31  _avl_insert
 11.96     0.47  0.11  3987156    0.03  0.03  key_value_new
  8.70     0.55  0.08 73272216    0.00  0.00  node_height
  7.61     0.62  0.07 15423187    0.00  0.01  avl_rebalance
  6.52     0.68  0.06 19283171    0.00  0.01  node_update_height
  6.52     0.74  0.06  1993578    0.03  0.38  avl_insert
  5.43     0.79  0.05  4926  10.15 10.15  _avl_save_to_path
  4.35     0.83  0.04 17352937    0.00  0.00  node_balance
  3.26     0.86  0.03  2008461    0.01  0.01  str_to_lower
  2.17     0.88  0.02  4935  4.05  4.05  node_free
  2.17     0.90  0.02  4934  4.05   168.43  avl_load_from_path
  1.09     0.91  0.01  1977651    0.01  0.01  node_new
  1.09     0.92  0.01                       _init
  0.00     0.92  0.00  1928965    0.00  0.01  avl_rotate_right
  0.00     0.92  0.00  9931       0.00  0.00  _avl_find
  0.00     0.92  0.00  9931       0.00  0.01  avl_find
  0.00     0.92  0.00  4952       0.00  0.13  _avl_delete
  0.00     0.92  0.00  4952       0.00  0.14  avl_delete
```

```
  0.00     0.92  0.00  4935       0.00  4.05  avl_free
  0.00     0.92  0.00  4935       0.00  0.00  avl_new
  0.00     0.92  0.00  4926       0.00  10.15  avl_save_to_path
  0.00     0.92  0.00  1027       0.00  0.01  avl_rotate_left
```

Итак, теперь поворотов вправо стало действительно намного больше, тайна раскрыта!

Осталась еще одна проблема: слишком много вызовов `str_to_lower`. Большинство вызовов этой функции происходит во время считывания дерева из файла, однако мы же и так сохраняем ключи в нижнем регистре, их не нужно переводить. Чтобы этого избежать я создам функцию для вставки в которой переданные строки гарантируются быть в нижнем регистре:

```
avl_result avl_insert_lowercase(avl* tree, const key_value value) {
     avl_result result = avl_result_exists;
     tree->root = _avl_insert(tree->root, value, &result);
     return result;
}

avl_result avl_insert(avl *tree, const key_value value) {
     key_value key_value_lowercase = {
          .value = value.value
     };

     strncpy(key_value_lowercase.key, value.key, KEY_LEN);
     str_to_lower(key_value_lowercase.key, KEY_LEN);

     return avl_insert_lowercase(tree, key_value_lowercase);
}
```

Тогда программа работает намного быстрее:

```
Flat profile:


Each sample counts as 0.01 seconds.
  %   cumulative   self              self  total
 time   seconds   seconds   calls  us/call  us/call  name
 54.92     0.34  0.34  2015065    0.17  0.23  _avl_insert
  6.56     0.38  0.04 19434402    0.00  0.00  node_update_height
```

```
  6.56     0.41  0.04  2015065   0.02  0.02  key_value_new
  4.92     0.45  0.03 17483765   0.00  0.00  node_balance
  4.92     0.47  0.03  5132  5.85 108.06  avl_load_from_path
  4.10     0.50  0.03  2015065   0.01  0.24  avl_insert_lowercase
  3.28     0.52  0.02 73836334   0.00  0.00  node_height
  3.28     0.54  0.02 15534012   0.00  0.00  avl_rebalance
  3.28     0.56  0.02  5133      3.90  3.90  node_free
  1.64     0.57  0.01  1999571   0.01  0.01  node_new
  1.64     0.58  0.01  34993     0.29  0.29  str_to_lower
  1.64     0.59  0.01  9952      1.00  1.00  _avl_find
  1.64     0.60  0.01  5133      1.95  1.95  avl_new
  1.64     0.61  0.01  4949      2.02  2.02  _avl_save_to_path
  0.00     0.61  0.00  1949723   0.00  0.01  avl_rotate_left
  0.00     0.61  0.00  20036     0.00  0.52  avl_insert
  0.00     0.61  0.00  9952      0.00  1.29  avl_find
  0.00     0.61  0.00  5133      0.00  3.90  avl_free
  0.00     0.61  0.00  5005      0.00  0.06  _avl_delete
  0.00     0.61  0.00  5005      0.00  0.34  avl_delete
  0.00     0.61  0.00  5005      0.00  0.06  avl_delete_lowercase
  0.00     0.61  0.00  4949      0.00  2.02  avl_save_to_path
  0.00     0.61  0.00       472  0.00  0.01  avl_rotate_right
```

По-хорошему можно сделать подобные функции для удаления и поиска, чтобы программист, использующий эту библиотеку мог использовать их в случаях когда строка гарантированно в нижнем регистре, но это уже выходит за рамки лабораторной работы.

Память

Для более простой и удобной работы с памятью, я пользовался выделением памятью только в самых необходимых случаях — для выделения памяти под вершины и для выделения памяти под структуру данных `avl`. Все строки

хранились как статические массивы определенной длины, заданной по условию.

dmalloc

dmalloc запускался с флагами `-i 1 high` для максимально подробного отчета, но на тесте поменьше (1000 команд, 50 слов), так как иначе, программа работает очень долго.

Отчет dmalloc'a:

```
1713700928: 19862: Dmalloc version '5.6.5' from 'http://dmalloc.com/'
1713700928: 19862: flags = 0x4f4ed03, logfile './dmalloc-log'
1713700928: 19862: interval = 1, addr = 0x0, seen # = 0, limit = 0
1713700928: 19862: starting time = 1713700927
1713700928: 19862: process pid = 67754
1713700928: 19862: Dumping Chunk Statistics:
1713700928: 19862: basic-block 4096 bytes, alignment 8 bytes
1713700928: 19862: heap address range: 0x73bef99e7000 to
0x73bef9d33000, 3457024 bytes
1713700928: 19862:    user blocks: 215 blocks, 487168 bytes (51%)
1713700928: 19862:    admin blocks: 16 blocks, 65536 bytes (6%)
1713700928: 19862:    total blocks: 231 blocks, 946176 bytes
1713700928: 19862: heap checked 2153
1713700928: 19862: alloc calls: malloc 396, calloc 955, realloc 0,
free 801
1713700928: 19862: alloc calls: recalloc 0, memalign 0, valloc 0
1713700928: 19862: alloc calls: new 0, delete 0
1713700928: 19862:   current memory in use: 450688 bytes (550 pnts)
1713700928: 19862:  total memory allocated: 1161680 bytes (1351 pnts)
1713700928: 19862:  max in use at one time: 468288 bytes (597 pnts)
1713700928: 19862: max alloced with 1 call: 4096 bytes
1713700928: 19862: max unused memory space: 407136 bytes (46%)
```

```
1713700928: 19862: top 10 allocations:
1713700928: 19862:  total-size  count in-use-size  count  source
1713700928: 19862:    252784     854        94720 320  main.c:66
1713700928: 19862:          808   101        608   76  main.c:119
1713700928: 19862:    253592     955        95328 396  Total of 2
1713700928: 19862: Dumping Not-Freed Pointers Changed Since Start:
// их очень много
1713700928: 19862:  not freed: '0x73bef9ccd008|s1' (4096 bytes) from
'unknown'
// ...
1713700928: 19862:  not freed: '0x73bef9cd5008|s1' (296 bytes) from
'main.c:66'
// ...
1713700928: 19862:  not freed: '0x73bef9cd5208|s1' (472 bytes) from
'unknown'
// ...
1713700928: 19862:  not freed: '0x73bef9d32668|s1' (8 bytes) from
'main.c:119'
// ...
1713700928: 19862:  total-size  count  source
1713700928: 19862:    94720 320  main.c:66
1713700928: 19862:          608   76  main.c:119
1713700928: 19862:    95328 396  Total of 2
1713700928: 19862: ending time = 1713700928, elapsed since start =
0:00:01
```

Отчет на ~600 строк. Куча не освобожденных указателей. В чем дело?

Оказывается, тут две ошибки:

1. Я не освобождаю новое AVL дерево и файл дескриптор, если загружаемый файл не в том формате. После исправления кода, часть освобожденных указателей ушла.

```
avl* avl_load_from_path(const char *path) {
     // ...
     while (fgets(line, COMMAND_BUFFER_LENGTH, file) != NULL) {
          if (line[0] != 'k') {
               avl_free(tree);
               fclose(file);
               return NULL;
          }
          // ...
     }
     // ...
}
```

2. Я забыл поменять размер буфера для считывания из файла после изменения формата хранения AVL дерева.

```
#define KEY_LEN 257
#define VALUE_LEN 20
#define COMMAND_BUFFER_LENGTH 1 + 1 + KEY_LEN + 1 +
VALUE_LEN + 1 + 1
#define FILE_LINE_LEN 2 + 1 + KEY_LEN + 1 + 2 + 1 +
VALUE_LEN + 1 + 1
// ...
avl* avl_load_from_path(const char *path) {
     // ...
     char line[FILE_LINE_LEN];
     // ...
     while (fgets(line, FILE_LINE_LEN, file) != NULL) {
          // ...
     }
     // ...
}
```

Ура, теперь в отчете всего 31 строка!

```
1713717733: 2944: Dmalloc version '5.6.5' from 'http://dmalloc.com/'
1713717733: 2944: flags = 0x4f4ed03, logfile 'dmalloc-log'
1713717733: 2944: interval = 1, addr = 0x0, seen # = 0, limit = 0
```

```
1713717733: 2944: starting time = 1713717733
1713717733: 2944: process pid = 82366
1713717733: 2944: Dumping Chunk Statistics:
1713717733: 2944: basic-block 4096 bytes, alignment 8 bytes
1713717733: 2944: heap address range: 0x7560b99d7000 to
0x7560b9a2a000, 339968 bytes
1713717733: 2944:     user blocks: 17 blocks, 61440 bytes (57%)
1713717733: 2944:     admin blocks: 9 blocks, 36864 bytes (34%)
1713717733: 2944:     total blocks: 26 blocks, 106496 bytes
1713717733: 2944: heap checked 301
1713717733: 2944: alloc calls: malloc 20, calloc 131, realloc 0, free
149
1713717733: 2944: alloc calls: recalloc 0, memalign 0, valloc 0
1713717733: 2944: alloc calls: new 0, delete 0
1713717733: 2944:   current memory in use: 8192 bytes (2 pnts)
1713717733: 2944:  total memory allocated: 86640 bytes (151 pnts)
1713717733: 2944:  max in use at one time: 29352 bytes (62 pnts)
1713717733: 2944: max alloced with 1 call: 4096 bytes
1713717733: 2944: max unused memory space: 24472 bytes (45%)
1713717733: 2944: top 10 allocations:
1713717733: 2944:  total-size  count in-use-size  count  source
1713717733: 2944:     37296 126        0     0  main.c:68
1713717733: 2944:           40    5          0     0  main.c:120
1713717733: 2944:     37336 131        0     0  Total of 2
1713717733: 2944: Dumping Not-Freed Pointers Changed Since Start:
1713717733: 2944:  not freed: '0x7560b99e3008|s1' (4096 bytes) from
'unknown'
1713717733: 2944:  not freed: '0x7560b99e6008|s1' (4096 bytes) from
'unknown'
1713717733: 2944:  total-size  count  source
```

```
1713717733: 2944:           0     0  Total of 0

1713717733: 2944: ending time = 1713717733, elapsed since start =
0:00:00
```

Однако осталось еще два не освобожденных указателя на 4 килобайта, что странно, так как памяти такого размера я нигде не выделял.

Как оказалось, ошибок с памятью у меня нет, а за не освобожденные указатели отвечает libc. Это обсуждалось в issues на гитхабе dmalloc'а. Также я проверил это на небольшой программе:

```c
#include <stdio.h>
#ifdef DMALLOC
#include "dmalloc.h"
#endif

#define LEN 256

int main() {
      char *other = malloc(256);
      free(other);
      char string[LEN];
      while (fgets(string, LEN, stdin) != 0) {
          printf("%s", string);
      }
      return 0;
}
```

Отчет dmalloc:

```
1713717798: 4: Dmalloc version '5.6.5' from 'http://dmalloc.com/'

1713717798: 4: flags = 0x4f4ed03, logfile 'dmalloc-log'

1713717798: 4: interval = 1, addr = 0x0, seen # = 0, limit = 0

1713717798: 4: starting time = 1713717798

1713717798: 4: process pid = 82649

1713717798: 4: Dumping Chunk Statistics:

1713717798: 4: basic-block 4096 bytes, alignment 8 bytes

1713717798: 4: heap address range: 0x73fabf250000 to 0x73fabf290000,
262144 bytes
```

```
1713717798: 4:  user blocks: 5 blocks, 12288 bytes (42%)
1713717798: 4:  admin blocks: 2 blocks, 8192 bytes (28%)
1713717798: 4:  total blocks: 7 blocks, 28672 bytes
1713717798: 4: heap checked 5
1713717798: 4: alloc calls: malloc 3, calloc 0, realloc 0, free 1
1713717798: 4: alloc calls: recalloc 0, memalign 0, valloc 0
1713717798: 4: alloc calls: new 0, delete 0
1713717798: 4:   current memory in use: 8192 bytes (2 pnts)
1713717798: 4:  total memory allocated: 8448 bytes (3 pnts)
1713717798: 4:  max in use at one time: 8192 bytes (2 pnts)
1713717798: 4: max alloced with 1 call: 4096 bytes
1713717798: 4: max unused memory space: 8192 bytes (50%)
1713717798: 4: top 10 allocations:
1713717798: 4:  total-size  count in-use-size  count  source
1713717798: 4:        256   1          0    0  test.c:11
1713717798: 4:        256   1          0    0  Total of 1
1713717798: 4: Dumping Not-Freed Pointers Changed Since Start:
1713717798: 4:  not freed: '0x73fabf250008|s1' (4096 bytes) from
'unknown'
1713717798: 4:  not freed: '0x73fabf252008|s1' (4096 bytes) from
'unknown'
1713717798: 4:  total-size  count  source
1713717798: 4:        0    0  Total of 0
1713717798: 4: ending time = 1713717798, elapsed since start = 0:00:00
```

Если убрать `printf` из программы, один из не освобожденных указателей пропадет:

```
1713717821: 3: Dmalloc version '5.6.5' from 'http://dmalloc.com/'
1713717821: 3: flags = 0x4f4ed03, logfile 'dmalloc-log'
1713717821: 3: interval = 1, addr = 0x0, seen # = 0, limit = 0
```

```
1713717821: 3: starting time = 1713717821
1713717821: 3: process pid = 82742
1713717821: 3: Dumping Chunk Statistics:
1713717821: 3: basic-block 4096 bytes, alignment 8 bytes
1713717821: 3: heap address range: 0x70b31f89c000 to 0x70b31f8da000,
253952 bytes
1713717821: 3:  user blocks: 3 blocks, 8192 bytes (40%)
1713717821: 3:  admin blocks: 2 blocks, 8192 bytes (40%)
1713717821: 3:  total blocks: 5 blocks, 20480 bytes
1713717821: 3: heap checked 4
1713717821: 3: alloc calls: malloc 2, calloc 0, realloc 0, free 1
1713717821: 3: alloc calls: recalloc 0, memalign 0, valloc 0
1713717821: 3: alloc calls: new 0, delete 0
1713717821: 3:   current memory in use: 4096 bytes (1 pnts)
1713717821: 3:  total memory allocated: 4352 bytes (2 pnts)
1713717821: 3:  max in use at one time: 4096 bytes (1 pnts)
1713717821: 3: max alloced with 1 call: 4096 bytes
1713717821: 3: max unused memory space: 4096 bytes (50%)
1713717821: 3: top 10 allocations:
1713717821: 3:  total-size  count in-use-size  count  source
1713717821: 3:        256   1          0    0  test.c:11
1713717821: 3:        256   1          0    0  Total of 1
1713717821: 3: Dumping Not-Freed Pointers Changed Since Start:
1713717821: 3:  not freed: '0x70b31f89c008|s1' (4096 bytes) from
'unknown'
1713717821: 3:  total-size  count  source
1713717821: 3:        0    0  Total of 0
1713717821: 3: ending time = 1713717821, elapsed since start = 0:00:00
```

Если еще же закомментировать строчку с вводом, второй не освобожденный указатель тоже пропадает:

```
1713717863: 2: Dmalloc version '5.6.5' from 'http://dmalloc.com/'
1713717863: 2: flags = 0x4f4ed03, logfile 'dmalloc-log'
1713717863: 2: interval = 1, addr = 0x0, seen # = 0, limit = 0
1713717863: 2: starting time = 1713717863
1713717863: 2: process pid = 82828
1713717863: 2: Dumping Chunk Statistics:
1713717863: 2: basic-block 4096 bytes, alignment 8 bytes
1713717863: 2: heap address range: 0x70afd3f5e000 to 0x70afd3f9a000,
245760 bytes
1713717863: 2:  user blocks: 1 blocks, 4096 bytes (33%)
1713717863: 2:  admin blocks: 2 blocks, 8192 bytes (67%)
1713717863: 2:  total blocks: 3 blocks, 12288 bytes
1713717863: 2: heap checked 3
1713717863: 2: alloc calls: malloc 1, calloc 0, realloc 0, free 1
1713717863: 2: alloc calls: recalloc 0, memalign 0, valloc 0
1713717863: 2: alloc calls: new 0, delete 0
1713717863: 2:   current memory in use: 0 bytes (0 pnts)
1713717863: 2:  total memory allocated: 256 bytes (1 pnts)
1713717863: 2:  max in use at one time: 256 bytes (1 pnts)
1713717863: 2: max alloced with 1 call: 256 bytes
1713717863: 2: max unused memory space: 256 bytes (50%)
1713717863: 2: top 10 allocations:
1713717863: 2:  total-size  count in-use-size  count  source
1713717863: 2:        256   1          0     0  test.c:11
1713717863: 2:        256   1          0     0  Total of 1
1713717863: 2: Dumping Not-Freed Pointers Changed Since Start:
1713717863: 2:  memory table is empty
```

```
1713717863: 2: ending time = 1713717863, elapsed since start = 0:00:00
```

По итогу, все указатели возможные указатели в программе освобождаются и утечек памяти в программе нет.

valgrind

Программа была скомпилирована с флагом `-ggdb` для сохранения дебаг символов. `valgrind` был запущен с флагом `--leak-check=full`:

```
==70608== Memcheck, a memory error detector
==70608== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==70608== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==70608== Command: ./main.out
==70608==
==70608==
==70608== HEAP SUMMARY:
==70608==  in use at exit: 0 bytes in 0 blocks
==70608==   total heap usage: 3,624 allocs, 3,624 frees, 1,834,488 bytes allocated
==70608==
==70608== All heap blocks were freed -- no leaks are possible
==70608==
==70608== For lists of detected and suppressed errors, rerun with: -s
==70608== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Как видно, утечек памяти в исправленной программе нет.

Выводы о полученных ошибках

Проверять работу программы нужно на множестве тестов, желательно если есть известная рабочая имплементация, сравнить получаемые результаты с ней.

Иногда тестов из тестовой системы недостаточно, нужно писать свои и проверять абсолютно весь функционал. По-хорошему нужно было бы написать программу на питоне, которая бы еще сверяла вывод программы с ожидаемым (например можно было бы завести словарь и провести с ним все те же операции).

Надо проверять всю ли память освободил, перед выходом из функции, а не паниковать и сразу выходить.

Сравнение с изначальной программой

Отчет `gprof` изначальной программы:

```
Flat profile:


Each sample counts as 0.01 seconds.
  %   cumulative   self              self  total
 time   seconds   seconds   calls  us/call  us/call  name
 33.33      0.18  0.18   876333    0.21  0.21  str_to_lower
 25.00      0.32  0.14   861349    0.16  0.24  _avl_insert
 12.96      0.39  0.07 39168095    0.00  0.00  node_height
  9.26      0.43  0.05  1722698    0.03  0.03  key_value_new
  5.56      0.47  0.03  4968  6.04  6.04  _avl_save_to_path
  5.56      0.49  0.03                             _init
  3.70      0.52  0.02   861349    0.02  0.50  avl_insert
  3.70      0.54  0.02       886   22.57 22.57  node_free
  0.93      0.54  0.01  6554121    0.00  0.01  avl_rebalance
  0.00      0.54  0.00  7353568    0.00  0.00  node_balance
  0.00      0.54  0.00   845894    0.00  0.00  node_new
```

```
  0.00     0.54  0.00   799316    0.00  0.01  avl_rotate_left
  0.00     0.54  0.00  9991  0.00  0.00  _avl_find
  0.00     0.54  0.00  9991  0.00  0.21  avl_find
  0.00     0.54  0.00  5075  0.00  0.00  avl_new
  0.00     0.54  0.00  5074  0.00  87.88  avl_load_from_path
  0.00     0.54  0.00  4993  0.00  0.09  _avl_delete
  0.00     0.54  0.00  4993  0.00  0.30  avl_delete
  0.00     0.54  0.00  4968  0.00  6.04  avl_save_to_path
  0.00     0.54  0.00       886   0.00  22.57  avl_free
  0.00     0.54  0.00       450   0.00  0.01  avl_rotate_right
```

Отчет `gprof` финальной программы:

```
Flat profile:


Each sample counts as 0.01 seconds.
  %   cumulative   self              self  total
 time   seconds   seconds   calls  us/call  us/call  name
 45.16     0.28  0.28  1978604    0.14  0.22  _avl_insert
 11.29     0.35  0.07  1978604    0.04  0.04  key_value_new
  8.06     0.40  0.05  4968       10.06 10.06  _avl_save_to_path
  7.26     0.45  0.04 72444130    0.00  0.00  node_height
  6.45     0.48  0.04 19068463    0.00  0.00  node_update_height
  4.84     0.52  0.03 15239679    0.00  0.01  avl_rebalance
  3.23     0.54  0.02  1978604    0.01  0.23  avl_insert_lowercase
  3.23     0.56  0.02  5075       3.94  3.94  avl_new
  2.42     0.57  0.01 17153602    0.00  0.00  node_balance
```

```
  1.61     0.58  0.01  1963191   0.01  0.01  node_new
  1.61     0.59  0.01  1913891   0.01  0.01  avl_rotate_left
  1.61     0.60  0.01  34958     0.29  0.29  str_to_lower
  1.61     0.61  0.01  5075      1.97  1.97  node_free
  1.61     0.62  0.01  5074      1.97   107.29  avl_load_from_path
  0.00     0.62  0.00  19974     0.00  0.51  avl_insert
  0.00     0.62  0.00  9991      0.00  0.00  _avl_find
  0.00     0.62  0.00  9991      0.00  0.29  avl_find
  0.00     0.62  0.00  5075      0.00  1.97  avl_free
  0.00     0.62  0.00  4993      0.00  0.07  _avl_delete
  0.00     0.62  0.00  4993      0.00  0.36  avl_delete
  0.00     0.62  0.00  4968      0.00  10.06  avl_save_to_path
  0.00     0.62  0.00       501  0.00  0.01  avl_rotate_right
```

Время исполнения немного увеличилось, но это не удивительно, ведь до этого программа работала некорректно: считывание полей с самыми большими ключами не работало, балансировка происходила неверно, происходили огромные утечки памяти.

Вывод

В ходе лабораторной работы я научился пользоваться различными утилитами для изучения работы программы, исправил множество различных ошибок, выявленных при помощи них.