Семинар 15: Аллокация памяти

Igor Zhirkov

14 декабря 2021 г.

Содержание

В рамках этого семинара мы познакомимся с тем, как может быть устроена куча.

1 Что такое куча

С данными связана такая характеристика как *время жизни*. Это тот промежуток выполнения программы, когда обращаться к этим данным корректно и не ведёт к неопределённому поведению.

Данные в C могут быть выделены следующими способами:

- Автоматически: локальные переменные функций создаются в момент их запуска и живут до их завершения. Правильно сказать: гарантированное время жизни переменных от запуска функции до её завершения.

  Зная, что компиляторы оптимизируют локальные переменные, мы больше не будем говорить, что они выделяются в стеке.

- Глобальные переменные и локальные переменные функций, если локальные переменные помечены `static`.

  И те, и другие создаются в глобальной области данных.

Других гарантированно возможных способов выделить память в “голом” C нет.

Чтобы выделить память в стеке или в глобальной области данных необходимо знать, сколько именно памяти понадобится, уже в момент компиляции. Однако иногда мы не знаем

этого заранее; например, когда принимаем по сети размер строки и саму строку; её нужно где-то выделить, а сколько под это выделять байт — неясно.

В таком случае приходится использовать `аллокаторы` — специальные программы, которые умеют резервировать области памяти и управляют тем, где будут находиться блоки памяти, которые программист запрашивает.

Функции `malloc`, `calloc`, `realloc`, `free` — это интерфейс к аллокатору из стандартной библиотеки C. С помощью вызова `malloc` можно выделить память, время жизни которой будет неопределённо долгим. Она может существовать до завершения работы программы, как глобальные переменные, или быть возвращена для переиспользования с помощью функции `free`.

Некоторые советы по работе с памятью:

- Если вам нужно выделить память, подумайте, нельзя ли её выделить в стеке. Это всегда проще, чем в куче.
- Не стесняйтесь передавать не очень большие структуры по значению.

2 Аллокатор блоков

Самый простой аллокатор возвращает блоки фиксированного размера. Их можно заполнять не полностью, остаток будет теряться. Помимо блоков нужно хранить карту их занятости; здесь она хранится в виде массива `bool`'ов, но для компактности можно было бы упаковывать булевы значения по восемь в каждый байт.

```
#define HEAP_BLOCKS 80
#define BLOCK_CAPACITY 1024

struct heap {
  struct block {
    char contents[BLOCK_CAPACITY];
  } blocks[HEAP_BLOCKS];
  bool is_occupied[HEAP_BLOCKS];
} global_heap = {0};
```

Задание Изучите файл `heap-0.c` и реализуйте в нём недостающие функции для резервирования блока и для возвращения блока в пул доступных. Почему `block_id` содержит ссылку на `struct heap`?

3 Аллокатор множества блоков

В предыдущем аллокаторе есть несколько существенных недостатков:

- блоки и заголовки перемежаются, и размер блоков фиксирован, поэтому нельзя занять несколько блоков подряд.
- размер блока фиксирован, и потому мы можем нести накланые расходы по памяти.

Задание Модифицируйте аллокатор так, чтобы он умел выделять несколько блоков подряд. См. файл `heap-1.c`. У блоков теперь будет статус с большей гранулярностью, нежели “занят — не занят”.

```
enum block_status { BLK_FREE = 0, BLK_ONE, BLK_FIRST, BLK_CONT, BLK_LAST };
```

Блоки будут связываться в одну из следующих конфигураций:

```
BLK_FREE //  свободный блок
BLK_ONE //  единичный занятый блок.
// после него или свободный или занятый другими данными.

BLK_FIRST, BLK_CONT, BLK_CONT, BLK_LAST
// последовательность блоков с  началом, серединкой и концом.

BLK_FIRST, BLK_LAST  // то же, но без серединки.
```

Выделение памяти должно привести к выделению адекватного количества блоков; освобождение памяти должно привести к освобождению блоков начиная с указанного, если этот блок – начало.

Задание придумайте два теста для вашего кода.

4 Чего не хватает для эффективности?

Чтобы аллокатор стал более эффективным, можно отказаться от фиксированного размера блока, снабдив каждый из них заголовком и организовав в связный список. Затем из единой непрерывной кучи можно сделать несколько фрагментов и добавить довыделение памяти (а именно страниц от операционной системы) в случае, когда куча закончилась. Это и будет темой лабораторной работы.