Лекция 14

Область видимости vs время жизни

- Область видимости переменной 'x' – точки в исходном тексте программы, в которых идентификатор 'x' обозначает эту переменную
  - Вложенная область видимости может перекрыть переменную из объемлющей области видимости
- Время жизни переменной – промежутки времени во время выполнения программы, в которые под эту переменную выделена память

Классы памяти

- Глобальные переменные
  - Существуют от момента запуска программы на выполнение до завершения работы
  - Память под глобальные переменные резервируется в исполняемом файле
- Локальные переменные
  - Существуют от момента входа в блок до момента выхода из блока
  - Память резервируется на стеке или в регистрах: стековый фрейм создается при входе в функцию и уничтожается при выходе
  - Обращения к локальным переменным транслируются в обращения к текущему стековому фрейму относительно регистра фрейма

Управление динамической памятью (кучей)

- Выполняется с помощью функций malloc, calloc, free, realloc, new, new[], delete, delete[]
- Память может запрашиваться фрагментами произвольного размера
- Память может освобождаться в произвольный момент времени
- Стандартные стратегии обслуживания (стек, очередь) неприменимы

Управление кучей

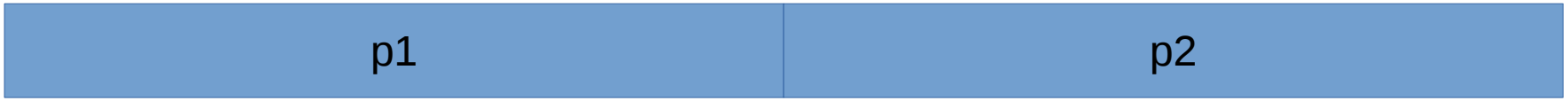
- Память может запрашиваться и освобождаться в нескольких нитях одновременно
- К структурам данных предъявляются разные требования по выравниванию, поэтому данные выравниваются по максимально жесткому требованию (4 байта – x86, 16 байт - x64)
-
- **Выделение памяти в куче намного медленнее, чем в стеке!**

Проблемы динамической памяти

- Необходимость блокировки в многонитевых программах
- Постепенное дробление больших непрерывных фрагментов памяти на маленькие
- Постепенная фрагментация динамической памяти
- Постепенный рост размера виртуального адресного пространства и невозможность возврата памяти ядру ОС

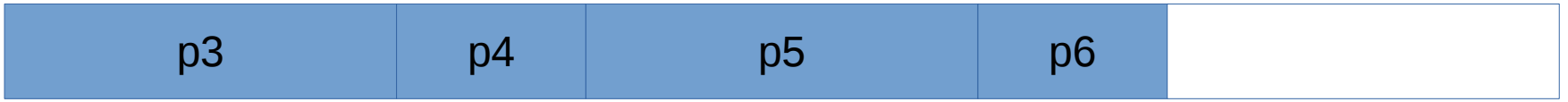
Фрагментация

p1 = malloc(4); p2 = malloc(4);

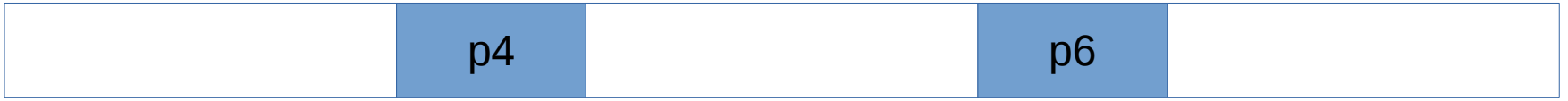


free(p1); p3 = malloc(2); p4 = malloc(1);

free(p2); p5 = malloc(2); p6 = malloc(1);



free(p3); free(p5);



Управление адресным пространством процесса

- Системный вызов sbrk() - изменить адрес конца сегмента данных

```
void  *sbrk(intptr_t increment);
```

- Сразу после загрузки исполняемого образа break addess — это конец сегмента данных
- sbrk возвращает предыдущее значение

Запрос памяти у ядра

- Когда при очередном вызове функции выделения памяти запрос не может быть удовлетворен, с помощью sbrk запрашивается порция памяти у ядра
- Как правило, память не возвращается ядру, даже если это возможно

Стратегии распределения

- Битовый массив блоков
- Списки свободных/занятых блоков

Битовый массив блоков

- Память разбивается на блоки выделения фиксированного размера (например, 16 байт)
- Каждому блоку выделения ставится в соответствие 1 бит в битовом массиве: 0 — блок свободен, 1 - занят

1	1	1	1	1	0	0	1	1	1	1	0	0	0

80 байт занято	32 св.	64 байта	48 байт

Списки блоков

- Вариант с одиним списком:
  - В памяти поддерживается двусвязный список выделенных/свободных блоков
  - Структура дескриптора блока:

```
struct memdesc {
    struct memdesc *prev;
    size_t size;
    unsigned char data[0];
};
```

  - Флаг свободный/занятый - младший бит size

Выделение памяти malloc

- Размер выделения округляется вверх до размера выравнивания (8 байт), 0 байт → 8
- В памяти ищется подходящий свободный блок
- Если свободного блока нет, запрашивается блок памяти с помощью sbrk() и помечается как свободный
- Найденный блок при необходимости дробится, формируется дескриптор блока памяти и возвращается указатель на поле data

Освобождение памяти free

- Из переданного указателя ptr вычитается 8, таким образом получаем указатель на дескриптор блока памяти
- Блок памяти помечается как свободный, при необходимости сливается с непосредственно предшествующим и/или следующим свободным блоком

Алгоритмы выделения блоков

- Первый подходящий
- Самый подходящий
- Быстрый подходящий: поддерживаются список свободных блоков наиболее часто запрашиваемых размеров
-
- Существует много различных алгоритмов управления динамической памятью для разных ситуаций, нет однозначно наилучшего

Реализации malloc/free

- В составе libc (например, glibc malloc)
- Tcmalloc (google performance tools)
  - (почти) Линейная масштабируемость при росте количества нитей, за счет большей фрагментации

Ошибки работы с динамической памятью

- Типичные ошибки:
  - Memory overrun (выход за положительную границу)
  - Memory underun (выход за 0)
  - Use after free
  - Double free
  - Free of non-allocated pointer
- Приводят к порче списков свободных блоков и падению программы в какой-то момент позже
- Программа valgrind — отладчик работы с динамической памятью

Автоматическое управление памятью

- Автоматическое управление памятью – программист не должен явно освобождать ранее выделенную память (free/delete/delete[])
- Освобождение памяти выполняется когда runtime (среда выполнения) “знает”, что блок более не используется

Автоматическое управление памятью

- Отслеживание времени жизни:
  - Подсчет ссылок (reference counting) – std::shared_ptr
  - Владение – std::unique_ptr
- Консервативное неперемещаемое освобождение
- Перемещающий сборщик мусора (mark & sweep garbage collector)

Перемещающий сборщик мусора

- Вся область динамической памяти делится на два региона: активный и резервный
- Когда память в активном регионе заканчивается, запускается сборка мусора
  - Начиная от “входных” указателей (из регистров, стека, глобальных переменных) активный регион обходится, и используемые блоки помечаются
  - Используемые блоки переносятся в резервный регион, устраняя фрагментацию,
  - Все указатели модифицируются, чтобы правильно указывать на новое местоположение
  - Резервный регион объявляется активным