УДК: 004.43 **Технические науки**

Авдеева Яна Александровна, *студентка ФДО и СПО ФГБОУ ВО «Национальный исследовательский Мордовский государственный университет им. Н.П. Огарёва» (г. Саранск)*

E-mail: Sergushinaes@yandex.ru

Прокин Александр Александрович, *преподаватель, «Национальный исследовательский Мордовский государственный университет им. Н.П. Огарёва» (г. Саранск)*

МОДЕЛЬ ПАМЯТИ В ЯЗЫКАХ ПРОГРАММИРОВАНИЯ

Аннотация: В данной статье подробно рассмотрены: модель памяти с высокоуровневой точки зрения — виды памяти, аллокаторы, сборщик мусора.

Ключевые слова: языки программирования, модели, память, виды памяти, аллокаторы, сборщик мусора.

Annotation: This article details: memory model from a high-level point of view-types of memory, allocators, garbage collector.

Keywords: programming languages, models, memory, types of memory, allocators, garbage collector.

Память — одна из не простых тем в информатике, но понимание устройства памяти компьютера даёт возможность разрабатывать более эффективные и сложные программы.

Существует 3 типа памяти: статический, автоматический и динамический.

Статический – выделение памяти до начала выполнения самой программы. Вид такой памяти доступен в течении всего времени выполнения программы. В большинстве случаев, в языках программирования, при

размещении объекта в статической памяти необходимо лишь задекларировать его в глобальной области видимости [1; 2; 3; 4].

Автоматический – вид памяти, который также известен как «размещение на стеке», – самый основной вид. Он автоматически выделяет различные аргументы и локальные переменные функции, а также другую различную метаинформацию при вызове необходимых функций, а далее он освобождает память при выходе из неё [2].

Стек, как структура данных, работает по такому принципу как LIFO («последним пришёл – первым ушёл»). Т.е. добавление и удаление значений в стеке возможно только с одной и той же стороны.

Автоматическая память работает, непосредственно, на основе стека, чтобы, при вызове из какой-то определённой части программы, функция не затёрла уже используемую автоматическую память, а добавила свои данные в конец стека, увеличивая его размер. При завершении такой функции её данные непременно будут удалены с конца стека, следовательно, уменьшая его размер. Длина стека останется прежней, что и до вызова этой функции, а у вызывающей функции указатель на конец стека будет указывать на тот же адрес [3].

В зависимости от конкретной платформы детали реализации автоматической памяти могут быть различными. Например, кому очищать из стека определённую метаинформацию функции и её аргументы: вызывающей функции или вызываемой? Как передавать результат: через стек или, что определённо быстрее, через регистры процессора (память, расположенную, непосредственно, на кристалле процессора). На все эти вопросы отвечает конкретная реализация calling convention – описание технических особенностей вызова подпрограмм, которое определяет способы передачи параметров, а также результата функции, способы вызова или возврата из функции.

Фиксированный размер автоматической памяти, определяется линковщиком (обычно – 1 мегабайт), от конкретной системы и настроек компилятора или линковщика зависит максимальный размер.

Если приложение вдруг выйдет за максимум автоматической памяти, его там может ждать Page Fault (сигнал SIGSEGV в POSIX-совместимых системах: Mac OS X, Linux, BSD и т. д.) – ошибка сегментации, которая приводит к аварийному завершению программы.

Динамическая – выделение памяти из ОС по требованию приложения.

Перед запуском программы автоматическая и статическая память выделяются единоразово. При их нехватке, либо если модель LIFO не совсем подходит, используется динамическая память [5].

При необходимости приложение может запросить у ОС дополнительную память напрямую через системный вызов или через аллокатор.

Когда выделяется память в распоряжение программы поступает указатель на начало выделенной памяти, который тоже должен где-то храниться, а именно: в статической, автоматической или также в динамической памяти. Чтобы возвратить память назад в аллокатор необходим именно сам указатель. Если попытаться использовать уже очищенную память, то это может привести к завершению программы с сигналом SIGSEGV.

Динамическую память, как основную, используют языки сверхвысокого уровня, т.е. создают все или почти все объекты в динамической памяти, а в статической памяти или держат указатели на эти объекты.

Максимальный размер динамической памяти зависит от многих факторов: среди них ОС, процессор, аппаратная архитектура в целом, максимальный размер ОЗУ у конкретного устройства.

У динамической памяти существует две явные проблемы. Во-первых, любое выделение или освобождение памяти в ОС — это системный вызов, который замедляет работу программы. А решением этой проблемы является аллокатор.

Аллокатор — это часть программы, запрашивающая память большими кусками через системные вызовы напрямую у ОС (в POSIX-совместимых ОС это mmap для выделения памяти и unmap — для освобождения), затем по частям приложению отдаёт эту память (в Си это могут быть

функции malloc()/free()). Такой подход увеличивает производительность, но также может вызвать фрагментацию памяти при длительной работе программы.

malloc()/free() и mmap/unmap — это не одно и то же. Первый является простейшим аллокатором в libc, а второй является системным вызовом. В большинстве языков аллокатор можно использовать только по умолчанию, но в языках с более низкоуровневой моделью памяти можно использовать и другие различные аллокаторы [2].

Очень трудно определить из-за сложности программ, когда необходимо освободить память в ОС, и это вторая явная проблема динамической памяти. Если забыть вызвать munmap() или free(), то произойдет следующая ситуация: память приложению уже будет не нужна, но ОС ещё будет считать, что эта память используется программой. Такую проблему называют «утечкой памяти». Есть несколько способов автоматического или полуавтоматического решения такой проблемы [6]:

RAII (Получение ресурса есть инициализация) – в ООП – организация получения доступа к ресурсу в конструкторе, а освобождения — в деструкторе соответствующего класса. Управление памятью достаточно реализовать в конструкторах и деструкторах, а компилятор вызовет их автоматически [7].

std::unique_ptr – класс уникального указателя. Он является единственным владельцем памяти и очищает её в своём деструкторе. Поэтому объекты класса std::unique_ptr не могут иметь никаких копий, но они имеют возможностьперемещения.

std::shared_ptr – класс общего указателя, который использует атомарный счётчик ссылок для того, чтобы подсчитать количество владельцев памяти. В конструкторе счётчик инкрементируется, в деструкторе – декрементируется. Как только счётчик становится равным нулю, то память сразу освобождается.

Но у std::shared_ptr существует такая проблема, например, когда объект А ссылается на объект В, а объект В ссылается на объект А. В этом случае у обоих объектов счётчик ссылок никогда не будет меньше одного и произойдёт утечка памяти. У такой проблемы существует два решения.

Использование std::weak_ptr, ссылающийся на объект, но не имеющий счётчика ссылок, и не может быть разыменован без предварительной конвертации в std::shared_ptr. Вторым решением этой проблемы является сборщик мусора [8].

Сборка мусора – одна из форм автоматического управления динамической памятью, помечающая все доступные из стека или статической памяти динамически выделенные объекты. Все объекты очищаются, если до них нельзя добраться через цепочку указателей, начиная с автоматической либо статической памяти, т. е. которые не были помечены сборщиком [7].

У каждого способа управления динамической памятью есть свои определённые плюсы и минусы. Чаще всего необходимо жертвовать производительностью программы для обеспечения скорости и простоты разработки, или наоборот: высокая производительность, а также ещё высокая требовательность к программистам, из-за чего вероятность ошибки при разработке программы выше и медленней чем сам процесс.

Библиографический список:

1. Новиков, Е. А. Языки программирования: конспект лекций / Е. А. Новиков, Ю. А. Шитов. – Красноярск : ИПК СФУ, 2008. – 407 с. – (Языки программирования : УМКД № 147-2007 / рук. творч. коллектива Ю. А. Шитов).

2. Титовский, С. Н. Языки программирования. Ассемблер : конспект лекций / С. Н. Титовский, Н. В. Титовская. – Красноярск : ИПК СФУ, 2008. – 125 с. – (Языки программирования : УМКД № 147-2007 / рук. творч. коллектива Ю. А. Шитов).

3. Сергушина Е. С. Теоретические аспекты анализа численности, состава и структуры персонала предприятия / Сергушина Е. С., Вечканова Е. А., Тумайкина А. Н., Сергушин С. Е. // Международный студенческий научный вестник. – 2018. – № 1. – С. 60.

4. Новиков, Е. А. Языки программирования. Язык С : конспект лекций / Е. А. Новиков, Ю. А. Шитов. – Красноярск : ИПК СФУ, 2008. – 407 с. – (Языки программирования : УМКД № 147-2007 / рук. творч. коллектива Ю. А. Шитов).

5. Языки программирования : метод. указания по лаб. работам / сост. : С. Н. Титовский, Н. В. Титовская, А. В. Патуринский, Ю. А. Шитов, Е. А. Новиков, Н. А. Богульская. – Красноярск : ИПК СФУ, 2008. – 97 с. – (Языки программирования : УМКД № 147-2007 / рук. творч. коллектива Ю. А. Шитов).

6. Панфилов, С. А. Алгоритм энергосбережения для автономных систем теплоснабжения / С. А. Панфилов, О. В. Кабанов // Вестник ЮУрГУ. Серия «Строительство и архитектура». – 2017. – Т. 17, № 1. – С. 67–74.

7. Прокин А. А., Богатырская В. А., Сергушина Е. С., Листратов И. С. Современное состояние и основные проблемы интернет-торговли в Российской Федерации [Электронный ресурс] // E-Scio: Электронное периодическое издание «E-Scio.ru». – Режим доступа: http://escio.ru/wp-content/uploads/2018/03/Прокин-А.-А.-Богатырская-В.-А.-Сергушина-Е.-С.-Листратов-И.-С..pdf.

8. Прокин А. А., Богатырская В. А., Сергушина Е. С., Лукин М. А. Создание и продвижение интерактивного веб-сайта для коммерческой организации Электронный ресурс] // E-Scio: Электронное периодическое издание «E-Scio.ru». – Режим доступа: http://escio.ru/wp-content/uploads/2018/05/Прокин-А.-А.-Богатырская-В.-А.-Сергушина-Е.-С.-Лукин-М.-А.-.pdf.