**Технологии программирования.**

**Лекция 3**

Динамическая память. Работа с указателями в С++

Любой объект программы занимает место в оперативной памяти. Местоположение определяется при помощи адреса. При объявлении переменной, под нее выделяется место, размер зависит от типа переменной. Для доступа к переменной используется идентификатор. Предполагается, что все указатели одинаковы, т.е. внутреннее представление адресов всегда одно и тоже.

При работе с ОЗУ выделяется 3 типа памяти:

* **Статическая память**, т.е. участок памяти, выделяемый программе при ее запуске, для размещения глобальных или статических объектов, а также объектов определенных в пространстве имен. Данная память выделяется до вызова функции ***main().***
* **Автоматическая память**. Резервируется при запуске программы, до вызова функции ***main()***  из свободной ОЗУ и используется для размещения локальных объектов. А именно для объектов в теле функции, получаемых функции в качестве параметра в момент вызова. Автоматическую память называются стеком.
* **Динамическая память.** Представляет совокупность блоков, выделяемых из доступной ОЗУ во время работы программы под размещение конкретных объектов.

Образ ЕХЕ-файла программы. В адресном пространстве каждого процесса содержится образ запускаемого файла программы. У процесса имеется собственная область памяти, область описателей доступных файлов и область системных ресурсов. (Запустив 2 копии одной программы получится совершенно 2 разных процесса).

Процессу выделяется 4 Гбайтной адресное пространство. Программа может обращаться к этому пространству используя 32-х разрядный адрес. Виртуальное адресное пространство распределено следующим образом.

* Старший Гигабайт занят под системные приложения. Используется для совместимости процессов, системных очередей и т.д.
* 3-й гигабайт. Область совместного использования
* Младшие гигабайты — собственное адресное пространство процесса. В нем располагается ЕХЕ, динамические библиотеки, глобальные данные программы, стек программы, динамически выделяемая область.

Виртуальная память любого процесса может находится в одном из 3-х состояний:

* **Свободная**. Ссылки на блок памяти отсутствуют.
* **Зарезервированная.** Блок доступен для разработчиков, но не может быть выделена для последующего выделения
* **Выделенная.** Блок данных назначен некоторому хранилищу.

При работе с памятью неизбежно возникает...

Средства защиты памяти

В windows используется набор средств для организации защиты памяти. К ним относятся:

* Объектно-ориентированная защита памяти, которая предполагает, что как только любой процесс получает указатель на блок выделенных адресов. Специальный монитор ссылок безопасности проверяет разрешении ли данному процессу доступ к указанным областям памяти, при этом каждый блок областей воспринимается как отдельный объект со свои ми атрибутами защиты. Если область процессу доступна, то он получает на нее ссылку, в противном случае выполняется попытка поиска другого объекта. Если такой объект не найден, то процесс получает код ошибки связанный с невозможностью выделения памяти.
* **Отдельное адресное пространство для каждого процесса**. Данное средство защиты предполагает, что процессам запрещается получать доступ к физическим адресам, выделенным для других процессов, работающих в системе.
* **Организация 2-х режимов работы**. Работа процессов может выполнятся в режиме ядра и пользовательском режиме. В режиме ядра процессу разрешен доступ к системным данным, в пользовательском они не доступны.
* **Страничный механизм защиты.** Каждому процессу для работы выделяется набор виртуальных страниц, каждая страница снабжена набором признаков, которые определяют разрешенные типы пользования в указанном режиме работы (ядра или пользовательском)
* **Принудительная очистка страниц.** При выгрузке процесса, диспетчер обязан освободить все блоки памяти, занятые процессом, при возможности осуществляется дефрагментация параллельно со сборкой мусора.

Благодаря блочному распределению памяти

*Основной особенностью организации памяти в windows является возможность использования страничного преобразования. Каждому процессу для его работы выделяется набор виртуальных страниц. Внутри организуются страничные преобразования и в общем случае страницы могут пересекаться, однако каждый процесс будет считать, что адресное пространство принадлежит только ему, будет считать, что использует последовательный адреса.*

Каждый процесс для организации выделенной памяти получает 32-х разрядный адрес, который состоит из следующих элементов:

* **31-22 бит.** Адрес каталога, определявший строку в каталоге страниц, обеспечивающий ссылку на таблицу используемых страниц.
* **21-12 бит.** Являются смещением таблицей страниц, определяющее место хранения указателя на соответствующую страницу.
* **11-0 бит.** Определяют смещение 4-х килобайтной таблицы, определяющие строку, хранящий физический адрес выделенного участка доступной оперативной памяти.

При загрузке любого процесса в память. В регистр CR3 загружается 20 битный адрес — указатель на каталог страниц, который определяет точку начала преобразования страниц.

При переключении между процессами ОС обновляет регистр CR3. Сам элемент таблицы страниц содержит следующую информацию:

* 31-27 бит. Биты защиты, которые поддерживают набор значений обрабатываемые при помощи WINAPI (доступность страниц, доступность для чтения\записи и т.д.)
* 26-7 бит. Физический адрес страницы.
* 6-2 бит. Индекс файла подкачки. Для ее организации в системе определено 16 файлов (ну вы поняли, да?)
* 2-0 бит. Состояние страницы в системе. TDP. T — переходная страниц. D — запись в страницу.

Для организации работы с динамической памятью в С++ предусмотрен широкий спектр операции, часть из них взято из ....

* Malloc. Позволяет определить ссылку на выделенный участок виртуальной памяти заданного размера. Возвращает не типизированный указатель. Для ее использования необходима явная типизация. Единственным параметром функции является размер памяти в байтах. Void\* malloc(size\_t size) {};
* Память выделенная malloc'ом должна быть освобождена при помощи **Free**. Компилятор настроен на автоматическое выравнивание адреса по границе параграфа. Каждый параграф представляет собой 16 байтовый блок, поэтому будет предпринята выделение 48 байт памяти.
* **Calloc.** Позволяет выделить n последовательных блоков памяти заданной длины. Возвращает не типизованный указатель, который требует явной типизации. Void\* calloc(size\_t nitem, size\_t size)
* **Farmalloc.** Позволяет получить указатель на блок памяти заданного размера за границей сегмента. Void\* farmalloc(size\_t size)
* **Realloc.** Функция перераспределения адресного пространства. Позволяет расширить, либо сузить ранее выжеденную область памяти. При выполнении функции, система просматривает область памяти начиная с указанного адреса. Если расширение памяти по заданному адресу возможно, то происходит смещение граничного маркера конца на заданное количество байт и в качестве результата функция возвращает точно такую же ссылку. Если объём запрашиваемой памяти по указанному адресу не достаточен, система просматривает доступные процессу адресное пространство с целью поиска требуемого участка. Если участок заданного размера обнаружен, то информация хранящаяся в информационном поле ранее выделенной памяти копируется по новому адресу и в качестве результата работы возвращается ссылку на вновь выделенный участок памяти. Если требуемый участок не обнаруживается, то возвращается 0 ссылка. realloc(void\* block, size\_t size)

**\* при копировании происходит очистка старой памяти?**

New and delete

* **new** позволяет выделить память под единичный объект. Всегда возвращает типизованный указатель.
* **delete** очищает память.

В windows имеется набор аргоритмов для управления областями памяти. Они различаются в зависимости от типов объектов, которыми они оперируют.

Для работы с объектами одного типа используют менеджеры объектов или stab-аллокаторы. Специфика алгоритмов заключается в том, что объём памяти, используемое для обработки объектов постоянен и выделение, и освобождение памяти происходит со строго фиксированной единицей

В менеджере объектов обычно используется для выделения и освобождения большого количества одинаковых объектов. Ограничение на количество объектов задается на момент загрузки процесса в систему. Под объекты выделяется массив памяти из максимального количества объектов заданного типа и содержащиеся в этой памяти элементы связываются со специальной структурой данных определяющих список свободных объектов. Тогда выделение памяти под объект — это возвращение ссылки на голову списка, а освобождение объекта — помещение ссылки на него конец списка очереди свободных объектов. Данный метод обычно используется в системе реального времени так как поиск свободного участка минимально трудоемок.

Stab-локатор является аналогом менеджера объектов, но в отличие от него имеет общий интерфейс для выделения и освобождения разнотипных объектов. Для каждого такого типа stab-локатор выделяет собственный pull, система создает кэш pull'ов, в котором определяется доступ ко всем зарегистрированные объектам.

**Для работы с объектами одного размера:**

* **Алгоритм битовой маски.** При выделении процессу области памяти набор блоков заданного размера маркируется специальным массивом, определяющий занятость или возможность доступа к заданному блоку. При необходимости выделения диспетчер просматривает битовый массив, определяет блок доступ, к которому возможен, возвращает ссылку на этот блок в качестве результата выделения памяти и изменяет значение битовой маски (1 — занято, 0 - свободно). При освобождении достаточно изменить значение соответствующего бита маски, что приведет к возможности повторного использования данного участка, а, следовательно, к потере хранимых в нем данных
* **Алгоритм близнецов.** Используется для организации работы с блоками памяти кратными степеням двойки. Диспетчер разбивает доступную процессу области памяти на pull'ы блоков разного размера. При необходимости выделения участка памяти диспетчер пытается найти требуемый блок в минимальном pull'е в ближайшем минимальном поле. Если в данном поле свободные блоки имеются адрес блока возвращается в качестве результата работы. Если не обнаружен, то производится попытка поиска в блоке 2-х кратного размера. Если найдет, то он извлекается из pull'a, разбивается пополам, часть блока перемещается в pull меньшего размера, а адрес на 2-ю часть возвращается в качестве адреса на 2-ю часть. При освобождении
* **Метод граничных маркеров.** Заключается в том, что выделенный блок памяти представляет собой двусвязный список всех занятых и свободных блоков памяти. Элементами списка является шапки блоков называемые маркеры. Модификация списка происходит при выделении памяти процессу под объект и освобождение ранее выделенной памяти. При поиске требуемого участка памяти могут использовать различные алгоритмы. В частности, алгоритм поиска первого соответствия, наиболее подходящего и наименее подходящего. Основным достоинством является отсутствие привязки к типу и размеру объектов. В соответствии с заданным элементом, в область памяти добавляется 2 дескриптора: начала и конца блоков. Для организации связанного списка перед дескриптором начала блока находится служебная информация, определяющая ссылки на следующий и предыдущий блок памяти, в зависимости от состояния текущего блока. В структуре элемента памяти определен размер текущего выделенного информационного блока. Флаг занятости определяет возможность доступа к заданному участку памяти. Если в результате последовательного освобождения возникают последовательно освобожденные участки, то они объединяются в единый блок, который затем может быть использован как целиком, так и по частям.

Утечка памяти

В Visual Studio встроена библиотека CRT. В нем находится CRTDEBUG, позволяющий определять состояние динамической памяти. Эти операторы используются в процессе отладки приложения, позволяют получить снимки памяти и по снимкам определять правильность ее использования.

Процесс сборки мусора

Особенности работы динамической памяти CLR. Как известно в обычном режиме работы для данных приложения выделяется 2 области памяти, в которых располагаются объекты программы:

* Стек времени выполнения
* Куча

В стеке хранятся локальные переменные программы, а в куче размещаются объекты, создаваемые путем вызова оператора работы с динамической памятью. В общем случае программист обязан заботится об освобождения динамической памяти после ее использования, однако на практике … оказывается довольно сложно, особенно в случае использования распределенных объектов. Платформа Net включает в свой состав среду времени выполнения CLR, которая позволяет решить эту задачу, включив механизм сборки мусора. Сборка мусора – процесс при котором автоматически отслеживаются ссылки на объекты приложения и объект существует только до тех пор, пока приложение его использует. Как только приложение перестает ссылаться на объект, он должен быть уничтожен. CLR реализует сборку мусора, используя дополнительную область памяти, которая называется управляемая куча. В отличие от обычной кучи, управляемая куча включает механизм сборки мусора. В управляемой среде CLR, управляющая куча всегда выделяется под управляющие данные. Эти данные называются «контролируемыми сборщиком мусора». Сборка мусора запускается автоматически в случае если среда выполнения обнаруживает, что в выполняемой куче недостаточно места для стабильной работы приложения. Каждая ссылка со стороны приложения на экземпляр типа в управляемой кучей, считается корнем. Среда находит копии каждого типа и при запуске сборки мусора строит граф всех размещенных в куче объектов, которые достижимы через корень приложения. В случае, если сборщик мусора обнаруживает достаточное количество недостижимых объектов, то происходит их удаление и уплотнение достижимых объектов. Таким образом процесс сборки мусора состоит из 3 этапов:

1. Маркировка. На данном этапе выполняется поиск всех используемых объектов и составляется граф.
2. Перемещение. Предполагает обновление ссылок на существующие объекты.
3. Сжатие. Предполагает освобождение пространства объектами и сжатие выживших объектов

Для организации сборки мусора используется метод поколений. Данный метод основан на распределении всех объектов по отдельным областям кучи. Распределение происходит исходя из того какое количество операций сборки мусора пережил тот или иной объект. Каждый раз, когда объект пережил сборку мусора, он переходит в следующее поколение. Сборщик мусора использует 3 поколения объектов:

1. Содержит кратко живущие объекты.
2. Содержит кратко живущие объекты, пережившие 1 сборку мусора. Является буфером между кратко и долго живущими объектами
3. Содержит долго живущие объекты, на которые существует ссылка на протяжений всей работы приложения.

Сборка мусора начинается с 0 поколения и подобное распределение объектов и такой подход позволяет существенно увеличить эффективность сборщика.