УДК 004.421

И. В. Тархова – магистрант кафедры вычислительных систем и сетей
А. В. Аксенов – научный руководитель

ПРИНЦИПЫ ОРГАНИЗАЦИИ УПРАВЛЕНИЯ ПАМЯТЬЮ В ЗАДАЧЕ КОНТЕКСТНОГО МОДЕЛИРОВАНИЯ

Часто для эффективного управления памятью в приложении возникает необходимость отказаться от стандартного менеджера памяти, предлагаемого компилятором, и написать свой собственный менеджер памяти (аллокатор). В данной работе будут рассмотрены особенности создания менеджера памяти для программы контекстного моделирования, выполняющей кодирование и декодирование входной последовательности символов с целью сжатия данных.

Контекстное моделирование подразумевает под собой построение модели, содержащей статистику об обрабатываемом файле, т.е. вероятностное распределение для символов алфавита для всех состояний модели. Модель является древовидной структурой, узлы которой связаны указателями. В ходе работы программы модель постоянно наращивается, т.е. постепенно увеличивается количество узлов, из которых она состоит. Узел представляет собой структуру данных, содержащую четыре указателя на другие узлы, два счетчика (для вычисления вероятности встречи определенного символа для текущего состояния модели) и, собственно, сами данные. Из-за того, что модель расширяется, необходимо постоянно выделять память под новые узлы. Для выделения динамической памяти компилятор языка C++ обращается к функциям API операционной системы, например, в ОС семейства Windows это функция HeapAlloc. Таким образом, при частом выделении памяти под структуры узлов модели, постоянно происходит обращение к стандартным функциям ОС, что отрицательно сказывается на времени выполнения программы.

Специфика задачи накладывает ограничения на реализацию аллокатора. В ходе работы программы, память все время выделяется, увеличивается количество накопленной статистики, но прирост эффективности от хранения старой статистики со временем становится все меньше, кроме того, ресурс памяти не безграничен, поэтому необходимо избавляться от старой статистики. Следует учитывать, что объем оставшейся статистики должен быть достаточен для эффективного кодирования и декодирования.

Основной стратегией разрабатываемого менеджера памяти является запрос больших фрагментов памяти, организации пулов памяти, и дальнейшее распределение выделенной памяти внутри приложения. Следовательно, обращение к стандартным функциям ОС будет происходить только один раз при запуске программы, а не каждый раз при добавлении нового узла модели.

В соответствии со стандартом C++, размер указателя зависит от конкретной реализации компилятора и не связан напрямую с разрядностью используемой платформы. Однако на большинстве современных ОС общего назначения (настольные UNIX совместимые системы, MS Windows) используются модели данных, в которых размер указателя соответствует разрядности адресной шины у архитектуры этих платформ. Ширина шины адреса определяет объем адресуемой памяти. Например, для Visual Studio 2008 под управлением Windows 7 32-bit размер указателя равен четыре байта. В разрабатываемом аллокаторе за счет того, что адресация происходит внутри пула, размер указателя может быть уменьшен. Вместо 2^32 байт оперативной памяти, мы адресуем только память пула. Возьмем пул размером 1 Мб, размер узла 16 байт, тогда в пуле могут храниться 65536 узлов, а значит для их адресации понадобится по два байта на указатель (рис. 1).

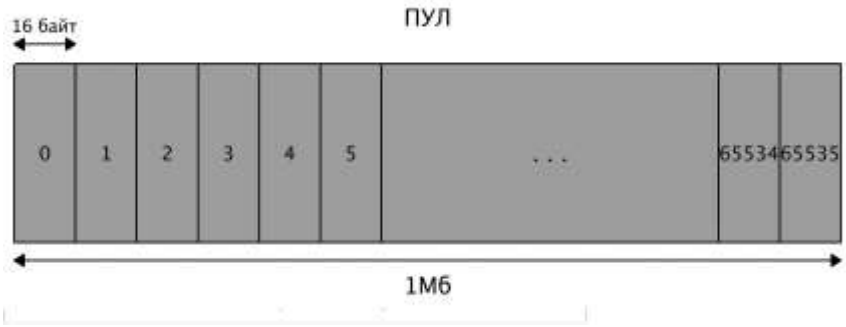

Рис. 1. Пул памяти

В аллокаторе для организации накопления и обновления статистики будут использоваться два пула памяти. Если заполненность первого пула достигает 80%, то одновременно с ним начинает заполняться второй пул. Таким образом, когда первый пул заполнится до конца, во втором пуле будет находиться достаточное количество свежей статистики, необходимой для эффективного кодирования. После чего первый пул может быть очищен. Далее процесс попеременного заполнения пулов будет повторяться на всем протяжении работы программы (рис. 2).

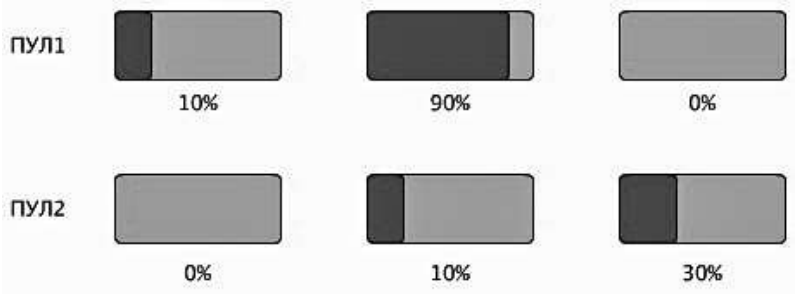

Рис. 2. Накопление статистики в пулах памяти

Проведем измерение времени, необходимого для выделения памяти под объекты. Сделаем по трем измерениям времени для каждого количества объектов, сначала без использования аллокатора, а затем с его использованием. В результате получим усредненные значения, представленные на следующем графике (рис. 3).

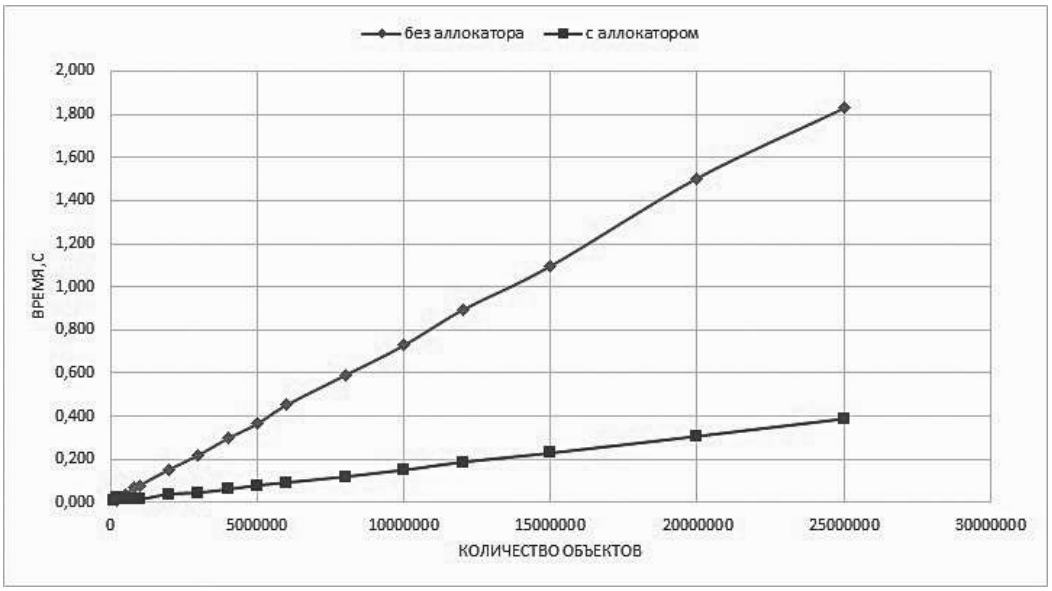

Рис. 3. Зависимость времени выделения памяти от количества выделяемых объектов

с использованием аллокатора и без его использования

Из графика видно, что при использовании аллокатора время выполнения программы меньше, чем при выделении памяти стандартными функциями C++, для любого количества объектов.

Библиографический список

1. Аксенов А. В. Разработка адаптивного алгоритма энтропийного кодирования // Научная сессия ГУАП.: СПб., 2010.
2. Шилдт Г. Самоучитель C++. СПб.: БХВ-Петербург, 2007.
3. Страуструп Б. Язык программирования C++: СПб.: Невский диалект, 2011.