Пермский филиал федерального государственного автономного
образовательного учреждения высшего образования
«Национальный исследовательский университет
«Высшая школа экономики»

Факультет экономики, менеджмента и бизнес-информатики

Чепоков Елизар Сергеевич

УПРАВЛЕНИЕ ПАМЯТЬЮ

Реферат

студента образовательной программы «Программная инженерия»
по направлению подготовки *09.03.04 Программная инженерия*

Доцент кафедры
информационных
технологий в бизнесе
_____
Л. Н. Лядова

Пермь, 2020 год

Стек

Стек – структура данных, организованная по принципу LIFO (Last In – First Out). Это память с линейно упорядоченными ячейками и специальным механизмом доступа, исключающим необходимость указания адреса при записи и чтении. Являясь неотъемлемой частью архитектуры процессора, стек поддерживается на аппаратном уровне с помощью команд для работы со стеком и специальных регистров (SS, SP, BP).

Различают аппаратный и программный стек. Первый используется для хранения адресов возврата из функций и их аргументов, а второй - пользовательская модель, или структура, данных.

Стек - часть памяти ОЗУ компьютера, которая предназначается для временного хранения байтов, используемых микропроцессором; при этом используется порядок запоминания байтов «последним вошел – первым вышел», поскольку такие ввод и вывод организовывать проще всего, также операции осуществляются очень быстро. Действия со стеком производится при помощи регистра указателя стека. Любое повреждение этой части памяти приводит к фатальному сбою.

Обычно стек реализуется в виде однонаправленного списка, где каждый элемент содержит помимо хранимой информации в стеке указатель на следующий элемент стека.

Также часто стек располагается в одномерном массиве с упорядоченными адресами. Такая организация стека удобна, если элемент информации занимает в памяти фиксированное количество слов, например, 1 слово. При этом отпадает необходимость хранения в элементе стека явного указателя на следующий элемент стека, что экономит память. При этом указатель стека обычно является регистром процессора и указывает на адрес головы стека.

Основные операции, реализованные для стека:

- Добавление элемента в вершину стека (PUSH);
- Извлечение с удалением элемента, находящегося в вершине стека (POP).
- Чтение элемента, находящегося в вершине стека (PEEK или TOP).

Рассмотрим пример программы, работающей со стеком:

```
#include <iostream>
#include <stack>
```

```
int main() {
  struct stack *stk;
  int i,n;
  float elem;
  stk = (struct stack*)malloc(sizeof(struct stack));
  init(stk);
  printf("Введите количество элементов в стеке: ");
  scanf("%d", &n);
  for(i=0; i<n; i++) {
    printf("Введите элемент %d:", i);
    scanf("%f", &elem);
    push(stk,elem);
  }
  printf("В стеке %d элементов\n\n", getcount(stk));
  stkPrint(stk);
  printf("Верхний элемент %f\n",stkTop(stk));
  do {
    printf("Извлекаем элемент %f, ", pop(stk));
    printf("в стеке осталось %d элементов\n", getcount(stk));
  } while(isempty(stk) == 0);
  getchar(); getchar();
  return 0;
}
```

Куча

Куча в программировании — название структуры данных, с помощью которой реализована динамически распределяемая память приложения, а также объём памяти, зарезервированный под эту структуру.

Для размещения и удаления динамических объектов используются примитивы «создать объект» и «удалить объект». Кроме того, перед началом работы программы выполняется инициализация кучи, в ходе которой вся изначально выделенная под кучу память отмечается как свободная.

Время жизни стековых переменных ограничено временем жизни функций, в которых эти переменные созданы, а переменные, выделенные в куче, существуют, пока выполняется программа или пока программист самостоятельно их не удалит.

Функции кучи:

1. GetProcessHeaps – Возвращает количество активных куч и извлекает дескрипторы для всех активных куч для вызывающего процесса. Функция получает дескриптор к куче по умолчанию вызывающего процесса, а также дескрипторы к любым дополнительным частным кучам, созданным вызовом функции HeapCreate в любом потоке процесса.

```
count = GetProcessHeaps(0, NULL); //количество куч процесса
```

2. GetProcessHeap – Функция получает дескриптор к куче по умолчанию для вызывающего процесса. Процесс может использовать этот дескриптор для выделения памяти из кучи процессов, при этом не создавая частную кучу с помощью функции HeapCreate.
3. HeapCreate – Создает кучу, к которой может обратиться вызывающий процесс. Функция резервирует место в виртуальном адресном пространстве процесса и выделяет физическое хранилище для заданной начальной части этого блока. Если параметр HEAP_NO_SERIALIZE не указан (простое значение по умолчанию), куча сериализует доступ в вызывающем процессе. Сериализация вызывает взаимное исключение, когда два или более потоков одновременно пытаются выделить или освободить блоки из одной и той же кучи. Функции HeapLock и HeapUnlock можно использовать для блокирования и разрешения доступа к сериализованной куче.

```
hHeap = HeapCreate(HEAP_NO_SERIALIZE | HEAP_GENERATE_EXCEPTIONS,
		h_size, 0);
	if (!hHeap)
	{
		cout << "Heap create failed." << endl;
		return GetLastError();
	}
```

4. HeapAlloc – Выделяет блок памяти из кучи. Выделенная память не является подвижной.

   Распределение памяти

```
memory = (int*)HeapAlloc(hHeap, NULL, a_size * sizeof(int));
```

   Перераспределение памяти

```
memory = (int*)HeapReAlloc(hHeap, HEAP_ZERO_MEMORY, a, 2 * a_size
* sizeof(int));
```

5. HeapReAlloc – Перераспределяет блок памяти из кучи. Эта функция позволяет изменять размер блока памяти и изменять другие свойства блока памяти. Выделенная память не перемещается.
6. HeapFree – Освобождает блок памяти, выделенный из кучи функцией HeapAlloc или HeapReAlloc.

```
if (HeapFree(hDefaultProcessHeap, 0, aHeaps) == FALSE) {
		_tprintf(TEXT("Failed to free allocation from default
process heap.\n"));
	}
```

7. HeapLock - Если куча не является сериализуемой, то параллельный доступ нескольких потоков к этой куче может нарушить ее непротиворечивое

состояние и вызвать ошибку в работе приложения. Для того чтобы получить монопольный доступ к куче, поток должен вызвать функцию HeapLock. В случае успешного завершения эта функция блокирует доступ остальных потоков к куче и возвращает ненулевое значение, а в случае неудачи возвращает значение FALSE. Единственным параметром этой функции является дескриптор кучи, к которой поток хочет получить монопольный доступ.

8. HeapUnlock - Если куча заблокирована потоком при помощи функции HeapLock и другой поток вызывает какую-нибудь функцию для доступа к этой куче, то система переведет его в состояние ожидания до тех пор, пока поток, вызвавший функцию HeapLock, не вызовет функцию HeapUnlock. В случае успешного завершения эта функция разблокирует кучу и возвращает ненулевое значение, а в случае неудачи куча остается заблокированной и функция возвращает значение FALSE.

Виртуальная память

Виртуальная память — метод управления памятью компьютера, позволяющий выполнять программы, требующие больше оперативной памяти, чем имеется в компьютере, путём автоматического перемещения частей программы между основной памятью и вторичным хранилищем

В современных операционных системах, таких как Windows, приложения и многие системные процессы ссылаются на память с помощью адресов виртуальной памяти и после автоматически преобразуются оборудованием в реальные адреса оперативной памяти.

Всем процессам, работающим в 32-разрядных версиях Windows, назначаются виртуальные адреса в диапазоне от 0 до 4 294 967 295 (2*32–1 = 4 ГБ) независимо от того, сколько фактической оперативной памяти доступно на компьютере.

В конфигурации Windows по умолчанию 2 гигабайта (ГБ) этого виртуального адресного пространства выделены каждому процессу для частного использования, а другие 2 ГБ совместно применяются всеми процессами и операционной системой.

Расширение физических адресов (PAE) — это возможность 32-разрядной архитектуры Intel, которая расширяет адреса физической памяти до 36 разрядов.

PAE не меняет размер виртуального адресного, а только объем фактической оперативной памяти, которая доступна процессору.

Преобразование 32-разрядных адресов виртуальной памяти, используемое в коде, выполняемом в процессе, в 36-разрядный адрес ОЗУ осуществляется автоматически и прозрачно оборудованием компьютера в соответствии с таблицами преобразования, которые создаются операционной системой. Любая страница виртуальной памяти может быть связана с любой страницей физической памяти.

Оперативная память — это ограниченный ресурс, а виртуальная память для большинства практических целей не ограничена. Если объем памяти, используемый всеми текущими процессами, превышает объем ОЗУ, операционная система перемещает страницы (частями по 4 КБ) одного или нескольких виртуальных адресных пространств на жесткий диск компьютера. Это освобождает ОЗУ для других целей. В системах Windows такие “выгруженные” страницы хранятся в одном или нескольких файлах Pagefile.sys в корневом каталоге раздела. В каждом разделе может быть один такой файл.

Увеличение нагрузки или спроса после определенной точки приводит к значительному снижению производительности. Это значит, что определенный ресурс становится дефицитным, т. е. "узким местом". В определенный момент объем такого ресурса нельзя увеличить. Это значит, что достигнуто ограничение архитектуры.

Основное средство для отслеживания производительности системы и определения узких мест — системный монитор.

Ниже приведен пример программы для работы с виртуальной памятью:

```
#include <iostream>
#include "windows.h"

using namespace std;

void main(){
        setlocale(LC_ALL, "Russian");
        int *memory;
        memory = (int*)VirtualAlloc(NULL, size*sizeof(int), MEM_COMMIT, PAGE_READWRI
TE);
        if (!memory){\
        cout << "Ошибка в распределении" << endl;
        return GetLastError();
        }
        cout << " Адрес виртуальной памяти: " << a << endl;
        if (!VirtualFree(memory, 0, MEM_RELEASE)){
        cout << "Освобождение памяти неуспешно." << endl;
        return GetLastError();
```

```
        }
        return 0;
}
```

VirtualAlloc() – резервирует, фиксирует или изменяет состояние области страниц в виртуальном адресном пространстве вызывающего процессора. Память автоматически инициализируется до нуля. При успешном выполнении функция возвращает указатель на выделенную память, иначе NULL.

VirtualFree() освобождает выделенную память. При успешном выполнении функция возвращает нулевое значение.

Файлы

Проецируемые файлы позволяют резервировать регион адресного пространства и передавать ему физическую память, которая не выделяется из страничного файла, а берется из файла, уже находящегося на диске.

Как только файл спроецирован в память, к нему можно обращаться так, будто он целиком в нее загружен.

Для использования проецируемых в память файлов требуется выполнить три операции:

1. Создать или открыть объект ядра "файл", идентифицирующий дисковый файл, который будет использоваться как проецируемый в память. Функция CreateFile - вызвав CreateFile, операционной системе указывается, где находится физическая память для проекции файла на жестком диске в сети, на CD-ROM или в другом месте.

```
HANDLE CreateFile( PCSTR pszFileName, DWORD dwDesiredAccess, DWORD dwShareMode, PSECURITY_AIIRIBUTES psa, DWORD dwCreationDisposition, DWORD dwFlagsAndAttribules, HANDLE hTemplateFile);
```

- pszFileName - идентифицирует имя создаваемого или открываемого файла
- dwDesiredAccess - указывает способ доступа к содержимому файла:
- dwSbareMode – указывает тип совместного доступа к данному файлу:

1. Создать объект ядра "проекция файла", чтобы сообщить системе размер файла и способ доступа к нему. Функция CreateFileMapping – вызвав CreateFileMapping, системе сообщается, какой объем физической памяти нужен проекции файла.

```
HANDLE CreateFileMapping( HANDLE hFile, PSECURITY_ATTRIBUTES psa, DWORD fdwProtect, DWOPD dwMaximumSizeHigh, DWORD dwMaximumSizcLow, PCSTR pszName);
```

- hFile – идентифицирует описатель файла, проецируемый на адресное пространство процесса этот описатель.
- psa — указатель на структуру SECURITY_ATTRIBUTES, которая относится к обьекту ядра "проекция файла", для установки защиты по умолчанию ему присваивается NULL.
- dwMaximumSizeHigh и dwMaximum SizeLow сообщают системе максимальный размер файла в байтах.
- pszName – строка с нулевым байтом в конце; в ней указывается имя объекта "проекция файла", которое используется для доступа к данному объекту из другого процесса

2. Указать системе, как спроецировать в адресное пространство процесса объект "проекция файла" — целиком или частично. Функция MapViewOfFile - вызвав MapViewOfFile, система резервирует регион адресного пространства под данные файла и передаёт их как физическую память, отображенную на регион

```
PVOID MapViewOfFile( HANDLE hFileMappingObject, DWORD dwDesiredAccess, DWORD dwFileOffsetHigh, DWORD dwFileOffsetLow, SIZE_T dwNumberOfBytesToMap);
```

- hFileMappingObject - идентифицирует описатель объекта "проекция файла", возвращаемый предшествующим вызовом либо CreateFtleMapping,
- либо OpenFileMapping
- dwDesiredAccess - идентифицирует вид доступа к данным:
- dwFileOffsetHigh и dwFileOffsetLow – сообщение системе, какой байт файла данных считать в представлении первым.
- dwNumberOfBytesToMap - указывается, сколько байтов файла данных должно быть спроецировано на адресное пространство.