Белорусский государственный университет
Факультет прикладной математики и информатики
Кафедра технологии программирования
Доц. Побегайло А.П.

7. Работа с виртуальной памятью в Windows.

7.1. Состояния виртуальной памяти процесса.

В операционных Windows виртуальный адрес процесса отличается от линейного адреса этого же процесса только интерпретацией бит линейного адреса. Поэтому можно сказать, что каждому процессу в Windows также доступно два гигабайта виртуальной памяти. Это не значит, что процесс может использовать всю эту память одновременно. Количество виртуальной памяти, доступной процессу, зависит от емкости физической памяти и дисков. Чтобы ограничить процесс в использовании виртуальной памяти, некоторые страницы в таблице страниц могут быть помечены просто как недоступные.

После этих замечаний перейдем к описанию состояния виртуальной памяти процесса. С точки зрения процесса страницы его виртуальной памяти могут находиться в одном из трех состояний:

свободны для использования (free);
зарезервированы, но не используются процессом (reserved);
распределены процессу для использования (committed).

Поясним эти состояния более подробно.

Первоначально при запуске процесса все страницы виртуальной памяти считаются свободными, естественно кроме тех, в которые загружена сама программа.

Чтобы распределить для использования свободные или зарезервированные страницы виртуальной памяти, процесс должен вызвать функцию *VirtualAlloc*. Только после успешного завершения этой функции процесс может использовать распределенную ему виртуальную память.

Третье состояния характеризует виртуальные страницы как зарезервированные. Это значит, что эти виртуальные страницы зарезервированы процессом для дальнейшего использования и не будут выделяться системой для использования процессу без точного указания процессом их адреса. Следует отметить, что при резервировании виртуальных страниц реальная память под эти страницы не выделяется.

7.2. Резервирование, распределение и освобождение виртуальной памяти.

Для резервирования или распределения области виртуальной памяти процесс должен вызвать функцию:

```
LPVOID  VirtualAlloc (
        LPVOID      lpAddress,          // область для распределения или резервирования
        SIZE_T      dwSize,             // размер области
        DWORD       flAllocationType,   // тип распределения
        DWORD       flProtect           // тип защиты доступа
);
```

которая в случае успешного завершения возвращает адрес виртуальной памяти, распределенной или зарезервированной процессом, а в случае неудачи – NULL. При этом отметим такую деталь, если распределение виртуальной памяти функцией *VirtualAlloc* завершается успешно, то выделенная память автоматически инициализируется нулями.

После завершения работы с виртуальной памятью её необходимо освободить, используя функцию

```
BOOL  VirtualFree (
        LPVOID      lpAddress,          // адрес области виртуальной памяти
        SIZE_T      dwSize,             // размер области
```

```
        DWORD       dwFreeType          // тип операции
);
```

которая в случае успешного завершения возвращает значение TRUE, а в случае неудачи – FALSE.

7.3. Блокирование виртуальных страниц в реальной памяти.

Если некоторая область виртуальной памяти будет часто использоваться процессом, то можно запретить системе выгружать эти виртуальные страницы из реальной памяти, иначе говоря, запереть или блокировать эти виртуальные страницы в реальной памяти. Для этого нужно использовать функцию

```
BOOL  VirtualLock (
        LPVOID      lpAddress,          // адрес области
        SIZE_T      dwSize              // размер области
);
```

которая в случае успешного завершения возвращает значение TRUE, а в случае неудачи – FALSE.

Для отмены блокировки виртуальных страниц в реальной памяти используется функция

```
BOOL  VirtualUnlock (
        LPVOID      lpAddress,          // адрес области
        SIZE_T      dwSize              // размер области
);
```

которая в случае успешного завершения возвращает значение TRUE, а в случае неудачи – FALSE.

7.4. Изменение атрибутов доступа к виртуальной странице.

Изменить атрибуты доступа к области виртуальной памяти можно при помощи вызова функции

```
BOOL  VirtualProtect (
        LPVOID      lpAddress,      // адрес области памяти
        SIZE_T      dwSize,         // размер области памяти
        DWORD       flNewProtect,   // флаги новых атрибутов доступа
        PDWORD      lpflOldProtect  // указатель на старые атрибуты доступа
);
```

которая в случае успешного завершения возвращает значение TRUE, а в случае неудачи – FALSE.

7.5. Управление рабочим множеством страниц процесса.

Узнать о количестве страниц, которые входят в рабочее множество процесса, можно посредством вызова функции

```
BOOL  GetProcessWorkingSetSize (
        HANDLE      hProcess,           // дескриптор процесса
        PSIZE_T     lpMinWorkingSetSize,    // минимальный размер рабочего множества
        PSIZE_T     lpMaxWorkingSetSize     // максимальный размер рабочего множества
);
```

которая в случае успешного завершения возвращает значение TRUE, а в случае неудачи – FALSE.

Минимальный и максимальный размеры рабочего множества страниц процесса можно изменить посредством вызова функции

```
BOOL  SetProcessWorkingSetSize (
        HANDLE      hProcess,               // дескриптор процесса
        SIZE_T      dwMinWorkingSetSize,    // минимальный размер рабочего множества
        SIZE_T      dwMaxWorkingSetSize     // максимальный размер рабочего множества
```

```
);
```

которая в случае успешного завершения возвращает значение – TRUE, а в случае неудачи – FALSE.

7.6. Инициализация и копирование блоков виртуальной памяти.

Чтобы заполнить блок памяти определенным значением, используется функция

```
VOID  FillMemory (
        PVOID       Destination,    // адрес блока памяти
        SIZE_T      Length,         // длина блока
        BYTE        Fill            // символ заполнитель
);
```

которая не возвращает никакого значения. Эта функция заполняет блок памяти, длина в байтах и базовый адрес которого задаются соответственно параметрами Length и Destination, символом, заданным параметром Fill. Если блок памяти необходимо заполнить нулями, то для этого можно использовать функцию

```
VOID  ZeroMemory (
        PVOID       Destination,    // адрес блока памяти
        SIZE_T      Length,         // длина блока
);
```

с аналогичными параметрами, исключая символ-заполнитель.

Для копирования блока виртуальной памяти используется функция

```
VOID  CopyMemory (
        PVOID       Destination,    // адрес области назначения
        CONST VOID  *Source,        // адрес исходной области
        SIZE_T      Length          // длина блока памяти
);
```

которая не возвращает никакого значения. Эта функция копирует блок памяти, длина в байтах и базовый адрес которого задаются соответственно параметрами Length и Source, в область памяти по адресу Destination. Отметим, что результат выполнения функции непредсказуем, если исходный и результирующий блоки памяти перекрываются. Чтобы этот случай отрабатывался правильно, нужно использовать для копирования функцию

```
VOID  MoveMemory (
        PVOID       Destination,    // адрес области назначения
        CONST VOID  *Source,        // адрес исходной области
        SIZE_T      Length          // длина блока памяти
);
```

параметры которой полностью совпадают с параметрами функции *CopyMemory*.

7.7. Определение состояния памяти.

Определить состояние области виртуальной памяти процесса можно при помощи вызова функции

```
DWORD  VirtualQuery (
        LPCVOID     lpAddress,                          // адрес области
        PMEMORY_BASIC_INFORMATION   lpBuffer,           // буфер для информации
        DWORD       dwLength                            // длина буфера
);
```