Чтобы подключиться к dll, надо оформить импорт вида

type

MFT\_REF = Packed Record

indexLow: dword;

indexHigh: word;

ordinal: word;

end;

function Get\_MFT\_EntryForPath (pcontext: ^pointer; path : PWideChar; pathlen:Integer; found: PMFT\_RECORD) : DWord; stdcall; external 'ntfs.dll';

function GetFileClusters(context:pointer; fileref:MFT\_REF; buflen:PDWORD; lcn\_len\_pairs:PDWORD):DWord; stdcall; external 'ntfs.dll';

function FreeNTFSContext(context:pointer;):DWord; stdcall; external 'ntfs.dll';

Обязательно использование stdcall, иначе возникнет конфликт машинного кода приложения и библиотеки.

Параметры Get\_MFT\_EntryForPath:

pcontext – адрес массива pointer[0..25], где каждой букве диска от A до Z сопоставлен свой контекст файловой системы. Например, для диска C это pcontext[2]. path – адрес Unicode строки с путём к файлу/каталогу, pathlen – длина пути, -1, если функция должна сама рассчитать длину null-terminated строки, PMFT\_REF – адрес структуры, куда функция вернёт номер записи в MFT.

Алгоритм поиска: если контекст диска не существует, то готовим его.

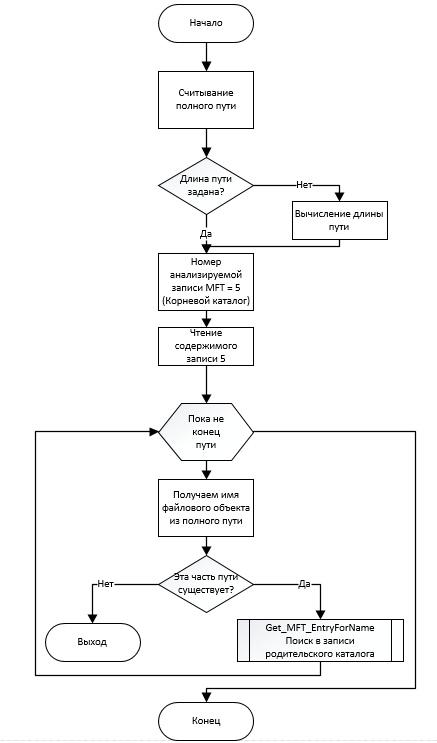
Если длина пути не задана, то вычисляем длину пути до нулевого символа.

Начинаем поиск первого каталога пути с файловой записи 5(Root directory).

Для этого в цикле передаём контекст диска, часть пути и файловую запись в функцию Get\_MFT\_EntryForName.

Разделяют путь на части символы «\» и «/». Для выделения части пути существуют индексы left и right. Индекс right проходит вправо пока не встретит разделитель или конец строки. Длина подстроки равна right-left. Передаём часть пути и текущую файловую запись в функцию Get\_MFT\_EntryForName. Если она вернула 0, значит есть ошибка выполнения. Если result=-1, то функция не нашла файловую запись. Если функция вернула 1, то переходим на следующую итерацию цикла, либо выводим номер файловой записи, если путь закончился. Для перехода на следующую часть пути надо сделать left=right+1, right=left.

Блок-схема для Get\_MFT\_EntryForPath:



Теперь рассмотрим Get\_MFT\_EntryForName

Функция Get\_MFT\_EntryForName имеет прототип function Get\_MFT\_EntryForName(context:pointer; filename:pwidechar; namelen:dword; result: PMFT\_REF):dword; stdcall;

Данная функция читает запись MFT по её номеру, затем находит атрибуты IR и IA. Если нет атрибута IR, там в функцию передали файловую запись не каталога и поиск в ней невозможен.

Далее атрибут IR передаётся в функцию FindInIndexRoot вместе с контекстом, именем файла/каталога, и длиной имени. Если FindInIndexRoot вернула 0, и не вернула в контекст диска vcn входа в IA, то считаем, что не нашли файловую запись. Если функция вернула 1, то файловая запись найдена и выводим её через параметр result.

Если IA отсутствует, то файловая запись не найдена, иначе вызываем FindInIndexAllocation и передаём ей контекст, IA атрибут, имя файла/каталога, и длину имени. Если функция вернула 1, то выводим найденную запись MFT, иначе выводим -1.

Блок-схема функции Get\_MFT\_EntryForName:

Теперь рассмотрим функцию FindInIndexRoot.

Она имеет прототип FindInIndexRoot(context:pointer; root:PMFT\_ATTR; filename:pwidechar; namelen:dword; result:PMFT\_REF):dword; stdcall.

Параметр root – адрес IR в MFT.

Алгоритм работы: переходим к индексным элементам и перебираем их в цикле. Если взведён нулевой бит флагов элемента, то выводим vcn в контекст. Если взведён первый бит флагов элемента, то это конечный элемент. Если нашли файл, то возвращаем 1 и выводим файловую запись, иначе возвращаем 0 и выводим -1.

Блок-схема FindInIndexRoot:

Теперь рассмотрим функцию FindInIndexAllocation:

Она имеет прототип FindInIndexAllocation(context:pointer; alloc: PMFT\_ATTR; filename:pwidechar; namelen:dword; result: PMFT\_REF).

Параметр alloc – адрес IA в MFT.

Теперь нужно распаковать IA runlist функцией ExpandRunlist.

Далее, имея vcn входа в дерево, читаем кластер lcn[vcn] в index record и передаём её в функцию FindInIndexRecord. Если FindInIndexRecord вернула 1, то выводим номер файловой записи и возвращаем 1. Если FindInIndexRecord вернула 0 и vcn=-1, то файловая запись отсутствует. Иначе продолжаем поиск в B-дереве пока не найдём файловую запись или не получим vcn=-1.

В сопоставлении lcn+vcn для index allocation есть некоторый момент. Например, если первый кластер с данными имеет номер 0x1000, а всего кластеров, например, 0х20, то если мы получили ссылку на кластер 0xA, то это кластер 0xA из 0x20 и далеко не всегда это кластер 0x100A. Например, если данные каталога разбросаны по фрагментам длиной 0x8, 0x14 и 0x4, а фрагменты начинаются по адресам 0x1000 0x1b00, 0x1c00, то в первый фрагмент номер не влезает, это 0x2 кластер второго фрагмента. Таким образом, на диске это кластер 0x1b02, а не 0x100A.

Блок-схема FindInIndexAllocation:

Теперь рассмотрим функцию FindInIndexRecord.

Её прототип BOOL FindInIndexRecord(context:pointer; lcnLow, lcnHigh:dword; filename:pwidechar; namelen:dword; result: PMFT\_REF):dword;stdcall.

Данная функция ищет файловую запись в индексной записи, расположенной в кластере lcnHigh.lcnLow. Она читает её в память, а далее проходит по ней аналогично алгоритму FindInIndexRoot.

Блок-схема функции:

Теперь рассмотрим функцию GetFileClusters:

Её прототип GetFileClusters(context:pointer; fileref:MFT\_REF; buflen, lcn\_len\_pairs:pdword):dword;stdcall.

Параметры GetFileClusters: context – контекст файловой системы для буквы диска текущего пути(для которого мы сейчас нашли ссылку в MFT), fileref – файловая запись, buflen – адрес длины массива в элементах вывода, lcn\_len\_pairs – адрес массива dword для вывода пар [lcnLow, lcnHigh, length].

Длина буфера в dword должна быть в 3 раза больше количества элементов, которые выведет функция. Например, если нужен буфер под 8 элементов вывода, то в массив должно помещаться 24 элемента dword.

Алгоритм вывода списка кластеров файла: определяем имеет ли запись MFT атрибут $DATA. Если имеет, то определяем резидентный ли он. Если резидентный, то выводим ошибку, что файл резидентный. Если атрибут $DATA нерезидентный, то считаем нужную длину буфера. Если она больше чем передано в buflen, то LastError:=ERROR\_INSUFFICIENT\_BUFFER, вывести в параметр buflen нужный размер буфера и вернуть 0.

Если длины буфера хватает, то выводим его отрезки файла в массив dword, длина которого кратна 3. Каждый i\*3 элемент – это lcnLow, каждый i\*3+1 элемент – это lcnHigh, каждый i\*3+2 элемент – это count of clusters, например, массив [0x256456, 0, 0x3461, 0x347612, 0, 0x126] – это отрезки файла, первый фрагмент которого начинается в кластере 0x256456 и содержит 0x3461 кластеров и второй фрагмент которого начинается в кластере 0x347612 и содержит 0x126 кластеров.

Если файловая запись содержит атрибут $ATTRIBUTE\_LIST, то она не содержит атрибутов $DATA, и $ATTRIBUTE\_LIST указывает на файловые записи с нерезидентным атрибутом $DATA. Чтобы найти ссылки на файловые записи с атрибутом $DATA, надо обойти тело $ATTRIBUTE\_LIST. Ниже приведена структура элемента тела $ATTRIBUTE\_LIST на языке Си:

typedef struct \_ATTRIBUTE\_LIST\_HEADER{

DWORD type;

USHORT size;

BYTE name\_size;

BYTE name\_offset;

DWORD data\_first\_vcn[2];

MFT\_REF file\_reference;

USHORT identifier;

}ATTRIBUTE\_LIST\_HEADER;

Таким образом, поле type определяет тип атрибута и надо найти запись со значением type=$80, а поле file\_reference указывает на файловую запись в MFT с атрибутом $DATA. Далее просто пройти по run list каждого найденного атрибута $DATA. Сначала мы обходим для подсчёта сколько нужно места под буфер, если места мало, то выводим по адресу buflen сколько надо, вызываем SetLastError(ERROR\_INSUFFICIENT\_BUFFER) и возвращаем 0. Если длина буфера достаточна, то заполняем массив lcn\_len\_pairs, по адресу buflen выводим сколько вывели элементов и возвращаем 1.