Unit 4: Memory Management

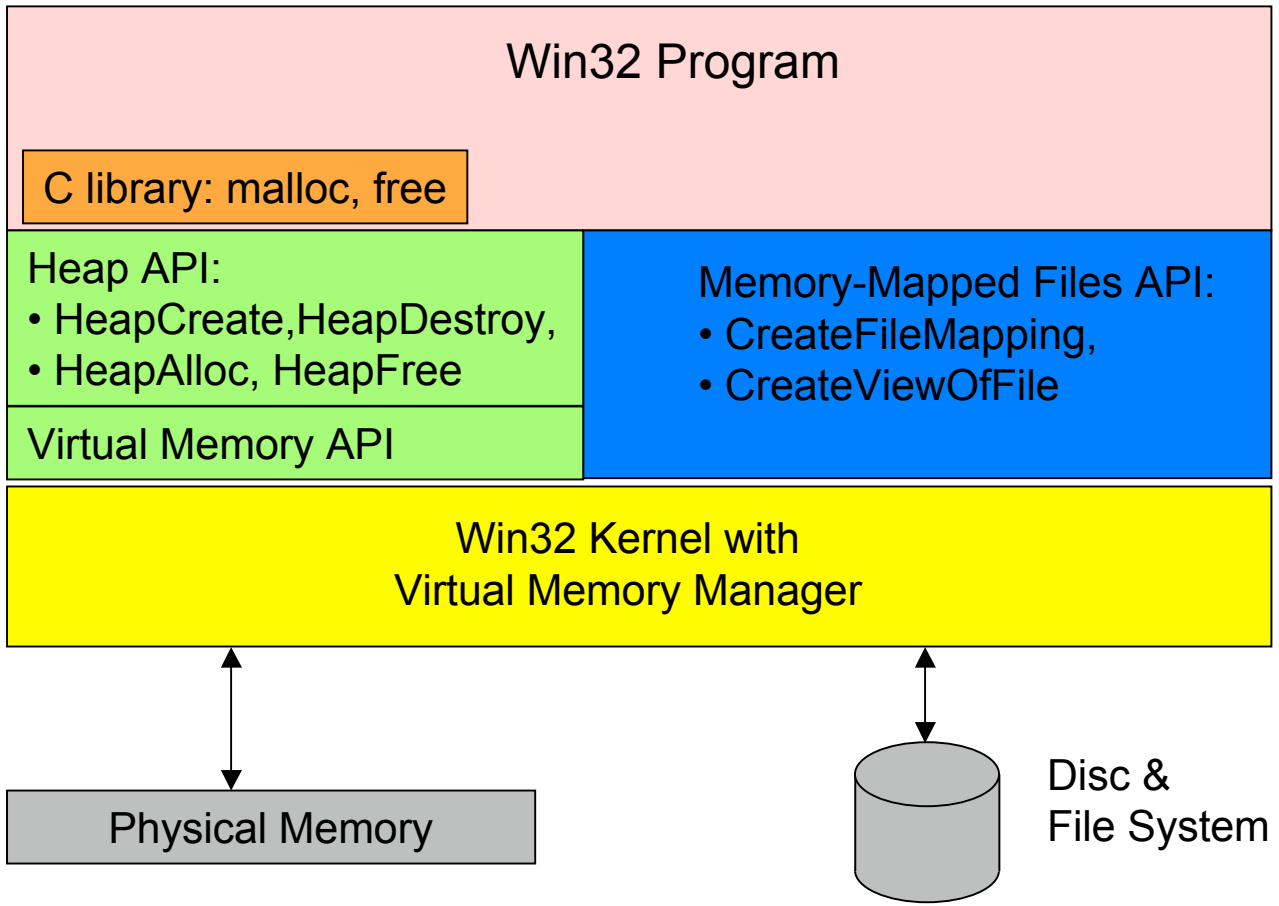
4.4. Win32 Memory Management

Win32 Memory Management

- Win32 maintains pools of memory in heaps
- A process can contain several heaps
  - C library functions manage default heap: malloc, free, calloc
- Heaps are Win32 objects – have handle
  - Each process has own default heap
  - Return value of NULL indicates failure (instead of INVALID_HANDLE_VALUE)

```
HANDLE GetProcessHeap( VOID );
HANDLE HeapCreate (DWORD floptions,
            DWORD dwInitialSize,
            DWORD dwMaximumSize);
BOOL HeapDestroy( HANDLE hHeap );
```

Win32 Mem. Manag. Arch.



Managing Heap Memory

```
LPVOID HeapAlloc( HANDLE hHeap,
                  DWORD dwFlags,
                  DWORD dwBytes );
```

- dwFlags:
  - HEAP_GENERATE_EXCEPTION,
    - raise SEH on memory allocation failure
    - STATUS_NO_MEMORY, STATUS_ACCESS_VIOLATION
  - HEAP_NO_SERIALIZE:
    - no serialization of concurrent (multithreaded) requests
  - HEAP_ZEROC_MEMORY: initialize allocated memory to zero
- dwSize:
  - Block of memory to allocate
  - For non-growable heaps: 0x7FFF8 (0.5 MB)
- HeapFree(), HeapReAlloc(),
- HeapCompact(), HeapValidate() – NT only

HeapLock(), HeapUnlock():
Manage concurrent accesses
to heap – NT only

Excerpt:
Sorting with Binary Search Tree

```
#define NODE_HEAP_ISIZE 0x8000
 for (iFile = iFirstFile; iFile < argc; iFile++) __try {
            /* Open the input file. */
            hIn = CreateFile (argv [iFile], GENERIC_READ, 0, NULL,
                        OPEN_EXISTING, 0, NULL);
            if (hIn == INVALID_HANDLE_VALUE)
                        ReportException (_T ("Failed to open input file"),
                                                SORT_EXCEPTION);
            /* Allocate the two heaps. */
            hNode = HeapCreate (
                        HEAP_GENERATE_EXCEPTIONS | HEAP_NO_SERIALIZE,
                        NODE_HEAP_ISIZE, 0);
            hData = HeapCreate (
                        HEAP_GENERATE_EXCEPTIONS | HEAP_NO_SERIALIZE,
                        DATA_HEAP_ISIZE, 0);
            /* Process the input file, creating the tree, actual search. */
            pRoot = FillTree (hIn, hNode, hData);
```

Heap Management Example (contd.)

```
        /* Display the tree in Key order. */
        if (!NoPrint) {
                _tprintf (_T ("Sorted file: %s"), argv [iFile]); Scan (pRoot);
        }
        /* Destroy the two heaps and data structures. */
        HeapDestroy (hNode); hNode = NULL;
        HeapDestroy (hData); hData = NULL;
        CloseHandle (hIn);
} /* End of main file processing loop and try block. */

__except (EXCEPTION_EXECUTE_HANDLER) {
        if (hNode != NULL) HeapDestroy (hNode);
        if (hData != NULL) HeapDestroy (hData);
        if (hIn != INVALID_HANDLE_VALUE) CloseHandle (hIn);
}
return 0;
```

- UNIX C library uses only a single heap
- UNIX sbrk() can create a Process‘ address space – no general-purpose MM
- UNIX does not generate signals on memory alloc.

Memory-Mapped Files

- No need to perform direct file I/O (read/write)
- Data structures will be saved – be careful with pointers
- Convenient & efficient in-memory algorithms:
  - Can process data much larger than physical memory
- Improved performance for file processing
- No need to manage buffers and file data
  - OS does the hard work: efficient & reliable
- Multiple processes can share memory
- No need to consume space in paging file

File Mapping Object

```
HANDLE CreateFileMapping (HANDLE hFile,
                          LPSECURITY_ATTRIBUTES lpsa,
                          DWORD fdwProtect,
                          DWORD dwMaximumSizeHigh,
                          DWORD dwMaximumSizeLow,
                          LPCTSTR lpszMapName );
```

Parameters:

- hFile:
  - hFile: handle to open file with compatible access rights (fdwProtect)
  - hFile == 0xFFFFFFFF: paging file, no need to create separate file
- fdwProtect:
  - PAGE_READONLY, PAGE_READWRITE, PAGE_WRITECOPY
- dwMaximumSizeHigh, dwMaximumSizeLow:
  - Zero: current file size is used
- lpszMapName:
  - Name of mapping object for sharing between processes or NULL

Shared Memory

```
HANDLE OpenFileMapping (HANDLE hFile,
        DWORD dwDesiredAccess,
        BOOL bInheritHandle,
        LPCTSTR lpName );
```

- Open an existing mapping object
  - Name comes from previous CreateFileMapping() call
  - First process creates mapping, subsequent processes open mapping
- dwDesiredAccess: same as fdwProtect
- lpName: name created with CreateFileMapping()
- CloseHandle() destroys mapping handles

Mapping Process Address Space to Mapping Objects

UNIX:
4.3BSD/SysV.4
have mmap() call;

See also
shmget(),shmctl(),
shmat(),shmdt()

```
LPVOID MapViewOfFile(    HANDLE hMapObject,
                         DWORD fdwAccess, DWORD dwOffsetHigh,
                         DWORD dwOffsetLow, DWORD cbMap );
BOOL UnmapViewOfFile ( LPVOID lpBaseAddress );
```

- Allocate virtual memory space and map it to a file through a mapping object
  - Similar to HeapAlloc – much coarser granularity
  - Pointer to allocated block is returned (file view)
- Parameters:
  - FILE_MAP_WRITE, FILE_MAP_READ, FILE_MAP_ALL_ACCESS flag bits for fdwAccess
  - cbMap: size; entire file if zero
- FlushViewOfFile(): create consistent view

Limitation:
2GB virtual
Address space

Example: File Conversion with Memory Mapping

```
            /* Open the input file. */
hIn = CreateFile (fIn, GENERIC_READ, 0, NULL,
            OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
if (hIn == INVALID_HANDLE_VALUE)  ReportException (_T ("Failure opening input file."), 1);
            /* Create a file mapping object on the input file. Use the file size. */
hInMap = CreateFileMapping (hIn, NULL, PAGE_READONLY, 0, 0, NULL);
if (hInMap == INVALID_HANDLE_VALUE) ReportException (_T ("Failure Creating input map."), 2);

pInFile = MapViewOfFile (hInMap, FILE_MAP_READ, 0, 0, 0);
if (pInFile == NULL) ReportException (_T ("Failure Mapping input file."), 3);
            /* The output file MUST have Read/Write access for the mapping to succeed. */
hOut = CreateFile (fOut, GENERIC_READ | GENERIC_WRITE,
                        0, NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
if (hOut == INVALID_HANDLE_VALUE) ReportException (_T ("Failure Opening output file."), 5);
hOutMap = CreateFileMapping (hOut, NULL, PAGE_READWRITE, 0, 2 * FsLow, NULL);
if (hOutMap == INVALID_HANDLE_VALUE) ReportException (_T ("Failure creating output map."), 7);

pOutFile = MapViewOfFile (hOutMap, FILE_MAP_WRITE, 0, 0, 2 * FsLow);
if (pOutFile == NULL) ReportException (_T ("Failure mapping output file."), 8);
```

Example (contd.)

```
        pIn = pInFile;                          /* actual file conversion */
        pOut = pOutFile;
        while (pIn < pInFile + FsLow) {
                *pOut = (WCHAR) *pIn; pIn++; pOut++;
        }

        /* Close all views and handles. */
        UnmapViewOfFile (pOutFile); UnmapViewOfFile (pInFile);
        CloseHandle (hOutMap); CloseHandle (hInMap);
        CloseHandle (hIn); CloseHandle (hOut);
        Complete = TRUE; return TRUE;
}

_except (EXCEPTION_EXECUTE_HANDLER) {
                /* Delete the output file if the operation did not complete successfully. */
        if (!Complete)
                DeleteFile (fOut);
        return FALSE;
}
```