Kamil Skarżyński

Programowanie niskopoziomowe
Operacje na plikach i katalogach
Laboratorium 07

1. Wprowadzenie

Podczas laboratoriów zapoznamy się z:
1. Procedurami umożliwiającymi zarządzanie stukturą katalogów i plików,
2. Procedurami umożliwiającymi zarządzanie plikami.

2. Zadania

1. Utwórz progam, tworzący katalog: GrupaN - gdzie N jest numerem twojej grupy.
2. Napisz program, który tworzy plik <TwojeNazwisko>.dat. Zapisz do niego 20 losowych(lab05) liczb całkowitych z zakresu wprowadzanego z konsoli.
3. Napisz program wczytujące liczby z pliku z zadania 2 i wyświetlający je na konsolę.
4. Napisz program odczytujący liczby z pliku <TwojeNaziwsko>.dat (zadanie 2) i kopiujący je do pliku <TwojeNazwisko>.txt. Liczby w nowym pliku powinny być zapisane w formacie tekstowym i rozdzielone określonym separatorem z tabeli wariantów.

3. Pomoc

3.1. Pobranie ścieżki aktualnego katalogu

Funkcja API Win32 GetCurrentDirectoryA z biblioteki Kernel32.lib umożliwia pobranie bieżącego katalogu.

```
DWORD WINAPI GetCurrentDirectory(
_In_  DWORD nBufferLength,
_Out_ LPTSTR lpBuffer
);
```

Parametry:

1. nBufferLength - Długość bufora przeznaczonego na przechowanie ścieżki aktualnego katalogu. Długość bufora musi uwzględniać kończący string null,
2. lpBuffer - Wskaźnik do bufora w którym zostanie zapisany string z aktualnym katalogiem. String jest null-terminated i zawiera ścieżkę absolutną do aktualnego katalogu.

W celu określenia wymaganej długości bufora możemy ustawić oba parametry na 0. W przypadku poprawnego wykonania procedury do rejestru EAX zwracana jest ilość znaków zapisanych do bufora. Jeśli procedura się nie powiedzie, wartość zwracana wynosi 0. Możemy zapoznać się z kodem błędu za pomocą procedury GetLastError(3.9).

Przykład wykorzystania:
Prototyp:

```
GetCurrentDirectoryA PROTO :DWORD,:DWORD
```

Segment danych:

```
nBufferLength   DWORD   255
buffer                  BYTE    255 dup(0)
```

Segment kodu:

```
INVOKE GetCurrentDirectoryA, nBufferLength, OFFSET buffer
```

3.2. Utworzenie nowego katalogu

Funkcja API Win32 CreateDirectoryA z biblioteki Kernel32.lib tworzy katalogu.

```
BOOL WINAPI CreateDirectory(
_In_      LPCTSTR                 lpPathName,
_In_opt_  LPSECURITY_ATTRIBUTES   lpSecurityAttributes
);
```

Parametry:

1. lpPathName - Adres bufora z nazwą nowego katalogu(ciąg znaków zakończony 0), jeżeli chcemy utworzyć katalog w aktualnej lokalizacji programu możemy wykorzystać procedurę GetCurrentDirectoryA(3.1) i połączyć dwie ścieżki(3.3),

2. lpSecurityAttributes - Adres strutuktury opisującej prawa dostępu do katalogu, możemy przekazać 0.

W przypadku poprawnego wykonania procedury do rejestru EAX zwracana jest wartość niezerowa. Jeśli procedura się nie powiedzie, wartość zwracana wynosi 0. Możemy zapoznać się z kodem błędu za pomocą procedury GetLastError(3.9). Przykład wykorzystania:

Prototyp:

```
CreateDirectoryA PROTO :DWORD,:DWORD
```

Segment danych:

```
buffer                  BYTE      "c:\GrupaN",0
```

Segment kodu:

```
INVOKE CreateDirectoryA, OFFSET buffer, 0
```

3.3. Procedury pomocnicze do operacji na ciągach znaków

lstrcpyA do kopiowania łańcuchów znaków (ciągu znaków zakończonych zerem), lstrcatA konkatenacja , łączenie dwóch łańcuchów, lstrlenA do obliczenia długości łańcucha znaków. Funkcja lstrcpyA ma argumenty: lpString1 – adres bufora – miejsca przeznaczenia, lpString2 – adres wierszu do kopiowania.

Funkcja lstrcatA ma argumenty: lpString1 – adres pierwszego bufora – (tu także będzie umieszony wynikowy łańcuch znaków) lpString2 – adres drugiego bufora do połączenia.

Funkcja lstrlenA ma argument: lpString – adres bufora, którego długość jest wyznaczana.

3.4. Tworzenie lub otwarcie pliku

Funkcja API Win32 CreateFileA z biblioteki Kernel32.lib tworzy plik i pobiera do niego uchwyt lub pobiera uchwytu do istniejącego pliku.

```
HANDLE WINAPI CreateFile(
_In_      LPCTSTR                lpFileName,
_In_      DWORD                  dwDesiredAccess,
_In_      DWORD                  dwShareMode,
_In_opt_  LPSECURITY_ATTRIBUTES  lpSecurityAttributes,
_In_      DWORD                  dwCreationDisposition,
_In_      DWORD                  dwFlagsAndAttributes,
_In_opt_  HANDLE                 hTemplateFile
);
```

1. lpFileName - Adres nazwy pliku lub urządzenia które ma być utworzone lub otwarte.
2. dwDesiredAccess - Tryb dostępu do pliku: GENERIC_READ - do odczytu, GENERIC_WRITE-do zapisu, można połączyć obie maski za pomocą operatora OR.

3. dwShareMode - Opcje dostępu do pliku przez inne aplikacje(możemy ustawić 0),
4. lpSecurityAttributes- Adres strutuktury opisującej prawa dostępu do katalogu, możemy przekazać 0.
5. dwCreationDisposition - Tryb otwarcia pliku: CREATE_ALWAYS - kreacja nowego, OPEN_EXISTING - otwarcie istniejącego.
6. dwFlagsAndAttributes - Dodatkowe atrybuty i flagi, możemy przekazać 0.
7. hTemplateFile - Deskryptor pliku tymczasowego, możemy przekazać 0.

Funkcja zwraca przez rejestr EAX uchwyt do pliku(handle), który możemy wykorzystywać w funkcjach operujących na plikach.

Przykład wykorzystania:
Stałe:

```
GENERIC_READ                    equ 80000000h
GENERIC_WRITE                   equ 40000000h
CREATE_NEW                      equ 1
CREATE_ALWAYS                   equ 2
OPEN_EXISTING                   equ 3
OPEN_ALWAYS                     equ 4
```

Prototyp:

```
CreateFileA PROTO :DWORD,:DWORD,:DWORD,:DWORD,:DWORD,:DWORD,:DWORD
```

Segment danych:

```
path BYTE "D:\potato.txt",0
```

Segment kodu:

```
INVOKE CreateFileA, OFFSET path, GENERIC_READ OR GENERIC_WRITE
,0,0,CREATE_ALWAYS,0,0
```

3.5. Zamknięcie pliku

Funkcja API Win32 CloseHandle z biblioteki Kernel32.lib pozwala na zamknięcie pobranego uchwytu.

```
BOOL WINAPI CloseHandle(
_In_ HANDLE hObject
);
```

1. hObject - Uchwyt przeznaczony do zamknięcia.

W przypadku poprawnego wykonania procedury do rejestru EAX zwracana jest wartość niezerowa. Jeśli procedura się nie powiedzie, wartość zwracana wynosi 0. Możemy zapoznać się z kodem błędu za pomocą procedury GetLastError(3.9). Przykład wykorzystania:

Prototyp:

```
CloseHandle PROTO :DWORD
```

Segment danych:

```
handle                  DWORD   0
```

Segment kodu:

```
INVOKE CloseHandle, handle
```

3.6. Zapis do pliku

Funkcja API Win32 WriteFile z biblioteki Kernel32.lib pozwala na zapis do pliku.

```
BOOL WINAPI WriteFile(
_In_        HANDLE       hFile,
_In_        LPCVOID      lpBuffer,
_In_        DWORD        nNumberOfBytesToWrite,
_Out_opt_   LPDWORD      lpNumberOfBytesWritten,
_Inout_opt_ LPOVERLAPPED lpOverlapped
);
```

1. hFile - Uchwyt do pliku pobrany za pomocą funkcji CreateFileA(3.4).
2. lpBuffer - Adres do bufora z danymi do zapisania.
3. nNumberOfBytesToWrite - Ilość bajtów którą chcemy zapisać z bufora,
4. lpNumberOfBytesWritten - Adres podwójnego słowa do którego zapisana zostanie faktyczna liczba zapisanych znaków.
5. lpOverlapped - Adres struktury z informacjami o nadpisanych danych. Możemy ustawić na 0.

Funkcja zwraca przez rejestr EAX ilość faktycznie zapisanych bajtów.

Przykład wykorzystania:
Prototyp:

```
WriteFile PROTO :DWORD,:DWORD,:DWORD,:DWORD,:DWORD
```

Segment danych:

```
dataToWrite BYTE "ziemniaczek",0
dataToWriteLen DWORD $-dataToWrite
dataWritten DWORD 0
fileHandle DWORD 0
```

Segment kodu:

```
;najpierw pobieramy do zmiennej fileHandle uchwyt do pliku
;nastepnie wykorzystujemy uchwyt w procedurze do zapisu danych
INVOKE WriteFile, fileHandle, OFFSET dataToWrite,
dataToWriteLen, OFFSET dataWritten,0
```

3.7. Odczyt z pliku

Funkcja API Win32 WriteFile z biblioteki Kernel32.lib pozwala na odczyt danych z pliku.

```
BOOL WINAPI ReadFile(
_In_          HANDLE       hFile,
_Out_         LPVOID       lpBuffer,
_In_          DWORD        nNumberOfBytesToRead,
_Out_opt_     LPDWORD      lpNumberOfBytesRead,
_Inout_opt_   LPOVERLAPPED lpOverlapped
);
```

1. hFile - Uchwyt do pliku pobrany za pomocą funkcji CreateFileA(3.4).
2. lpBuffer - Adres do bufora do którego zapisane zostaną odczytane dane.
3. nNumberOfBytesToRead - Ilość bajtów którą chcemy odczytać z pliku,
4. lpNumberOfBytesRead - Adres podwójnego słowa do którego zapisana zostanie faktyczna liczba odczytanych znaków.
5. lpOverlapped - Adres struktury z informacjami o nadpisanych danych. Możemy ustawić na 0.

W przypadku poprawnego wykonania procedury do rejestru EAX zwracana jest wartość niezerowa. Jeśli procedura się nie powiedzie, wartość zwracana wynosi 0. Możemy zapoznać się z kodem błędu za pomocą procedury GetLastError(3.9). Wskaźnik w którym miejscu w pliku się znajdujemy jest zapisywany automatycznie.

Przykład wykorzystania:
Prototyp:

```
ReadFile PROTO :DWORD,:DWORD,:DWORD,:DWORD,:DWORD
```

Segment danych:

```
buffor BYTE 255 dup(0)
dataToReadLen DWORD 255
dataRad DWORD 0
fileHandle DWORD 0
```

Segment kodu:

```
;najpierw pobieramy do zmiennej fileHandle uchwyt do pliku
;nastepnie wykorzystujemy uchwyt w procedurze do zapisu danych
INVOKE ReadFile, fileHandle, OFFSET dataToRead,
dataToReadLen, OFFSET dataRad,0
```

3.8. Przemieszczanie się po pliku

Do przemieszczenia w pliku służy funkcja API Win32 SetFilePointer z biblioteki kernel32.lib.

```
DWORD WINAPI SetFilePointer(
_In_          HANDLE hFile,
_In_          LONG   lDistanceToMove,
```

```
_Inout_opt_ PLONG  lpDistanceToMoveHigh,
_In_        DWORD  dwMoveMethod
);
```

1. hFile - Uchwyt do pliku pobrany za pomocą funkcji CreateFileA(3.4).
2. lDistanceToMove - Odległość (w bajtach) o którą chcemy się przesunąć .
3. lpDistanceToMoveHigh - ten argument musi być równy 0, jeśli rozmiar pliku jest mniejszy niż (2^{32}–2). Jeśli rozmiar pliku jest większy niż (2^{32}–2), to ten argument musi być adresem 32-bitowej zmiennej, która razem z argumentem lDistanceToMove tworzy 64-bitową odległość.
4. dwMoveMethod - opcja wskazująca na regułę liczenia odległości: FILE_BEGIN – odległość jest liczona od początku pliku, FILE_CURRENT – odległość jest liczona od aktualnej pozycji, FILE_END – odległość jest liczona od końca pliku.

Funkcja zwraca (przez rejestr EAX) pozycję wskaźnika pliku. Jeśli rozmiar pliku jest większy niż (2^{32}–2), to argument lpDistanceToMoveHigh wskazuje na 32-bitową zmienną, która razem z zawartością rejestru EAX tworzy 64-bitową pozycję.

Przykład wykorzystania:
Stałe:

```
FILE_BEGIN                    equ 0h
FILE_CURRENT                  equ 1h
FILE_END                      equ 2h
```

Prototyp:

```
SetFilePointer PROTO :DWORD,:DWORD,:DWORD,:DWORD
```

Segment danych:

```
buffor BYTE 255 dup(0)
dataToReadLen DWORD 255
dataRad DWORD 0
fileHandle DWORD 0
```

Segment kodu:

```
;najpierw pobieramy do zmiennej fileHandle uchwyt do pliku
;nastepnie wykorzystujemy uchwyt w funkcji
INVOKE SetFilePointer, fileHandle, 10, 0, FILE_BEGIN
```

3.9. Sprawdzenie ostatniego kodu błędu

```
.DATA
formErr     DB "%d=%xh",0Dh,0Ah,0
nErr DD (?)
bufor DB 128 dup (0)

rbuf DD (?)
```

```
rout DD (?)
hout DD (?)
.CODE
call   GetLastError
mov    nErr,EAX
INVOKE   wsprintfA,OFFSET bufor,OFFSET formErr,nErr,nErr
mov    rbuf, EAX
push 0
push OFFSET rout
push rbuf
push OFFSET bufor
push hout
call WriteConsoleA
```