Time and WinAPI

Here's a brief description of each of the time and date-related functions in the Win32 API:

CompareFileTime:
Compares two 64-bit file times to determine their order.

DosDateTimeToFileTime:
Converts MS-DOS date and time values to a 64-bit file time, allowing easy compatibility with older date and time representations.

FileTimeToDosDateTime:
Performs the reverse operation, converting a 64-bit file time to MS-DOS date and time values.

FileTimeToLocalFileTime:
Converts a UTC (universal coordinated time) file time to a local file time, making it suitable for use in the local time zone.

FileTimeToSystemTime:
Converts a 64-bit file time to a SYSTEMTIME structure, providing detailed date and time information.

GetFileTime:
Retrieves the date and time when a file was created, last accessed, and last modified.

GetLocalTime:
Retrieves the current local date and time, useful for obtaining the current local system time.

GetSystemTime:
Retrieves the current system date and time in UTC format, allowing for consistent time information across different time zones.

GetSystemTimeAdjustment:
Determines whether the system is applying periodic time adjustments to its time-of-day clock, important for handling time adjustments like daylight saving time.

GetSystemTimeAsFileTime:
Retrieves the current system date and time in UTC format, providing a 64-bit file time.

GetTickCount:
Retrieves the number of milliseconds that have elapsed since the system was started, useful for measuring time intervals or system uptime.

GetTimeZoneInformation:
Retrieves the current time-zone parameters, allowing you to obtain information about the system's time zone.

LocalFileTimeToFileTime:
Converts a local file time to a file time based on UTC, enabling the conversion of local time to a more universal format.

SetFileTime:
Sets the date and time that a file was created, last accessed, or last modified, allowing for the modification of file timestamps.

SetLocalTime:
Sets the current local time and date on the system, making it useful for adjusting the system's local time settings.

SetSystemTime:
Sets the current system time and date, enabling adjustments to the system's time settings.

SetSystemTimeAdjustment:
Allows you to enable or disable periodic time adjustments to the system's time-of-day clock.

SetTimeZoneInformation:
Sets the current time-zone parameters, providing control over the system's time zone settings.

SystemTimeToFileTime:
Converts a SYSTEMTIME structure to a 64-bit file time, allowing for the transformation of detailed date and time information into a universal format.

SystemTimeToTzSpecificLocalTime:
Converts a UTC time to a specified time zone's corresponding local time, useful when you need to adjust time information to a specific time zone.

Here's the combined MASM program that includes the SYSTEMTIME structure, GetLocalTime, SetLocalTime, and GetTickCount functions, as well as a stopwatch timer:

```
0917 INCLUDE Irvine32.inc
0918 INCLUDE macros.inc
0919
0920 .data
0921 sysTime SYSTEMTIME <> ; SYSTEMTIME structure
0922 startTime DWORD ?     ; Start time for the stopwatch timer
0923
0924 .code
0925 main PROC
0926     ; Get the current local time.
0927     INVOKE GetLocalTime, ADDR sysTime
0928     ; Display the current local time.
0929     call DisplayTime
0930     ; Set the local time to a specific value.
0931     ; For example, you can set it to January 1, 2023, 12:00:00.
0932     mov sysTime.wYear, 2023
0933     mov sysTime.wMonth, 1
0934     mov sysTime.wDay, 1
0935     mov sysTime.wHour, 12
0936     mov sysTime.wMinute, 0
0937     mov sysTime.wSecond, 0
0938     INVOKE SetLocalTime, ADDR sysTime
0939     ; Get the current local time again after setting it.
0940     INVOKE GetLocalTime, ADDR sysTime
0941     ; Display the updated local time.
0942     call DisplayTime
0943     ; Start a stopwatch timer.
0944     INVOKE GetTickCount
0945     mov startTime, eax
0946     ; Perform some calculations to simulate a time-consuming operation.
0947     mov ecx, 10000100h
```

```
0948 L1:
0949     imul ebx
0950     imul ebx
0951     imul ebx
0952     loop L1
0953
0954     ; Get the current tick count and calculate elapsed time.
0955     INVOKE GetTickCount
0956     sub eax, startTime
0957
0958     ; Display the elapsed time.
0959     call WriteDec
0960     mWrite <" milliseconds have elapsed", 0dh, 0ah>
0961
0962     exit
0963 main ENDP
0964
```

```
0965 DisplayTime PROC
0966     ; Display the current local time.
0967     mWrite "Current Local Time: "
0968     call WriteDec, sysTime.wMonth
0969     mWrite <"/", 0>
0970     call WriteDec, sysTime.wDay
0971     mWrite <"/", 0>
0972     call WriteDec, sysTime.wYear
0973     mWrite <" ", 0>
0974     call WriteDec, sysTime.wHour
0975     mWrite <":", 0>
0976     call WriteDec, sysTime.wMinute
0977     mWrite <":", 0>
0978     call WriteDec, sysTime.wSecond
0979     mWrite <" (Day of Week: ", 0>
0980     call WriteDec, sysTime.wDayOfWeek
0981     mWrite <")", 0dh, 0ah>
0982     ret
0983 DisplayTime ENDP
0984 END main
0985
```

Here's an explanation of the program in paragraph format:

The program starts by defining a structure called SYSTEMTIME, which is used to hold information about date and time. It includes fields like year, month, day of the week, day of the month, hours, minutes, seconds, and milliseconds. This structure is essential for working with date and time-related functions in the Windows API.

The program utilizes the GetLocalTime function, a Windows API function that retrieves the current local date and time according to the system's clock. It takes a single parameter, a pointer to a SYSTEMTIME structure, where it stores the current date and time values. This function is essential for obtaining the current time for further processing.

On the other hand, the SetLocalTime function is another Windows API function used to set the system's local date and time. It also takes a SYSTEMTIME structure as a parameter, but this time, it contains the desired date and time values. By calling this function, you can modify the system's date and time settings programmatically.

The program also incorporates the GetTickCount function, which is used to measure the number of milliseconds that have passed since the system started. This function doesn't require any parameters and returns the elapsed time in the EAX register. It's particularly useful for timing operations and determining the duration of processes.

There's a custom procedure in the program called DisplayTime, which serves to display the various components of a SYSTEMTIME structure, such as the year, month, day, hour, minute, second, and day of the week. This procedure uses different write functions to display these components on the console.

The program's main procedure is the entry point. It first calls GetLocalTime to retrieve and display the current local time. After that, it sets the local time to a specific value, allowing you to modify the date and time as needed. The program then calls GetLocalTime again to retrieve and display the updated local time. To simulate a time-consuming operation, the program performs calculations in a loop. Before and after this loop, it uses GetTickCount to measure the elapsed time and displays it.

In summary, this program showcases how to work with date and time in a Windows environment using the SYSTEMTIME structure and relevant Windows API functions. It demonstrates retrieving and displaying the

current local time, setting the local time, and measuring elapsed time using GetTickCount. You can adjust the date and time values as necessary for your specific requirements.

==========

The **Sleep function** is a part of the Win32 API that allows programs to introduce pauses or delays. This can be useful for controlling the timing of various operations in a program.

The function takes a parameter that specifies the length of time to sleep, and then it puts the processor into a low-power state until the specified time has elapsed.

The **GetDateTime procedure** is a convenient utility to retrieve date and time information. It returns the number of 100-nanosecond intervals that have elapsed since January 1, 1601.

The procedure generally follows these steps:

It calls a function like GetLocalTime, which populates a SYSTEMTIME structure with the current date and time information.

It converts this SYSTEMTIME structure to a FILETIME structure using the SystemTimeToFileTime function. Then, it copies the resulting FILETIME structure to a 64-bit quadword.

The FILETIME structure is used to divide a 64-bit quadword into two doublewords.

The GetDateTime procedure, which receives a pointer to a 64-bit quadword variable as an argument, is responsible for storing the current date and time in the specified variable in the FILETIME format used by Win32.

In simpler terms, the **Sleep functio**n allows programs to pause for a specified period of time, while the **GetDateTime procedure** allows programs to retrieve the current date and time.

Both functions are useful for controlling the timing of various operations in Win32 applications.

```
0988 ; Sleep Function
0989 Sleep PROTO,
0990 dwMilliseconds:DWORD
0991
0992 ; GetDateTime Procedure
0993 GetDateTime PROC,
0994 pStartTime:PTR QWORD
0995 LOCAL sysTime:SYSTEMTIME, flTime:FILETIME
0996
0997 ; Get the system local time
0998 INVOKE GetLocalTime,
0999 ADDR sysTime
1000
1001 ; Convert the SYSTEMTIME to FILETIME
1002 INVOKE SystemTimeToFileTime,
1003 ADDR sysTime,
1004 ADDR flTime
1005
1006 ; Copy the FILETIME to a 64-bit integer
1007 mov esi, pStartTime
1008 mov eax, flTime.loDateTime
1009 mov DWORD PTR [esi], eax
1010 mov eax, flTime.hiDateTime
1011 mov DWORD PTR [esi+4], eax
1012 ret
1013 GetDateTime ENDP
```

The Sleep function allows you to introduce time delays, and the GetDateTime procedure retrieves the current date and time and stores it in a 64-bit quadword. This code can be integrated into your assembly programs as needed.

```
; Sleep for 1 second
mov eax, 1000 ; 1000 milliseconds = 1 second
call sleep

; Continue execution
```

The eax register is used to specify the length of time to sleep. The call sleep instruction then calls the sleep function. Once the sleep function has returned, the program will continue execution.

It is important to note that the sleep function can be interrupted by certain events, such as a timer interrupt. If this happens, the program will resume execution immediately, even if the specified sleep time has not yet elapsed.

Here are some additional things to keep in mind when using the sleep function in MASM:
The sleep function is typically implemented as a system call.

This means that it must be executed in a privileged mode, such as kernel mode. The sleep function can be blocked by other processes.

This means that if another process is holding the kernel lock, the sleep function will not be able to execute until the other process releases the lock.

The sleep function can cause the processor to enter a low-power state. This can save power, but it can also delay the execution of other programs.

Overall, the sleep function is a powerful tool that can be used to control the execution of a program. However, it is important to be aware of the limitations of the function and to use it carefully.