COAL (EL-2003)

Spring-2025

Final Project Report

"Real-Time Clock with Alarm System using MASM"

Group Members:

Hafsa Rashid (23K-0064)

Syeda Sara Ali (23K-0070)

Instructor:

Sir Ubaidullah

Objective

The objective of this project was to design and implement a real-time alarm clock in x86 assembly language using Windows API. The program displays the current time, allows the user to set an alarm, and triggers an alarm sound at the specified time. It also supports features like snoozing and stopping the alarm.

Tools and Technologies

- **Language:** x86 Assembly (MASM Syntax)
- **Platform:** Windows
- **Assembler:** MASM32
- **Windows API Functions Used:**
    - GetLocalTime
    - GetStdHandle
    - SetConsoleCursorPosition
    - PlaySound

Key Features Implemented

Real-Time Clock Display

- Continuously displays the current time in HH:MM:SS format.
- Updates the display every second using a 1-second delay.
- Cursor positioning and formatted display implemented using SetConsoleCursorPosition and conditional formatting.

```
UpdateTimeDisplay PROC
    ; Save current cursor position
    push cursorPos.X
    push cursorPos.Y

    ; Set color for time display
    mov eax, COLOR_TIME
    call SetTextColor

    ; Set cursor position
    mov cursorPos.X, 6
    mov cursorPos.Y, 0
    INVOKE SetConsoleCursorPosition, hStdOut, cursorPos

    ; Display hours
    mov eax, currHour
    cmp eax, 10
    jae HourTwoDigit
    push eax
    mov eax, 0
    call WriteDec
    pop eax
HourTwoDigit:
    call WriteDec

    mov edx, OFFSET colonStr
    call WriteString
```

```
    ; Display minutes
    mov eax, currMin
    cmp eax, 10
    jae MinuteTwoDigit
    push eax
    mov eax, 0
    call WriteDec
    pop eax
MinuteTwoDigit:
    call WriteDec

    mov edx, OFFSET colonStr
    call WriteString

    ; Display seconds
    mov eax, currSec
    cmp eax, 10
    jae SecondTwoDigit
    push eax
    mov eax, 0
    call WriteDec
    pop eax
```

```
SecondTwoDigit:
    call WriteDec

    ; Restore cursor position
    pop cursorPos.Y
    pop cursorPos.X
    INVOKE SetConsoleCursorPosition, hStdOut, cursorPos

    ret
UpdateTimeDisplay ENDP
```

Alarm Setting

- Prompts the user to enter alarm hour and minute.
- Validates the input to ensure valid ranges (0–23 for hour, 0–59 for minute)
- Stores alarm values in global variables (alarmHour, alarmMin).

```
InputAlarmTime PROC
    ; Position cursor for input
    mov cursorPos.X, 0
    mov cursorPos.Y, 1
    INVOKE SetConsoleCursorPosition, hStdOut, cursorPos

    ; Set prompt color
    mov eax, COLOR_PROMPT
    call SetTextColor

HourInput:
    mov edx, OFFSET promptAlarmHour
    call WriteString
    call ReadInt
    jno ValidHour   ; Check for overflow
    jmp InvalidHour

ValidHour:
    cmp eax, 0
    jl InvalidHour
    cmp eax, 23
    jg InvalidHour
    mov alarmHour, eax
    jmp MinutePrompt
```

```
InvalidHour:
    ; Reset cursor position
    mov cursorPos.X, 0
    mov cursorPos.Y, 1
    INVOKE SetConsoleCursorPosition, hStdOut, cursorPos
    ; Clear the line
    mov ecx, 60
ClearHourLine:
    mov al, ' '
    call WriteChar
    loop ClearHourLine
    ; Reset cursor and try again
    mov cursorPos.X, 0
    mov cursorPos.Y, 1
    INVOKE SetConsoleCursorPosition, hStdOut, cursorPos
    mov eax, COLOR_PROMPT
    call SetTextColor
    jmp HourInput

MinutePrompt:
    ; Move to next line for minute input
    inc cursorPos.Y
    INVOKE SetConsoleCursorPosition, hStdOut, cursorPos
```

```
MinuteInput:
    mov edx, OFFSET promptAlarmMin
    call WriteString
    call ReadInt
    jno ValidMinute   ; Check for overflow
    jmp InvalidMinute

ValidMinute:
    cmp eax, 0
    jl InvalidMinute
    cmp eax, 59
    jg InvalidMinute
    mov alarmMin, eax
    jmp InputComplete

InvalidMinute:
    ; Reset cursor position
    mov cursorPos.X, 0
    mov cursorPos.Y, 2
    INVOKE SetConsoleCursorPosition, hStdOut, cursorPos
    ; Clear the line
    mov ecx, 60
ClearMinuteLine:
    mov al, ' '
    call WriteChar
    loop ClearMinuteLine
```

```
    loop ClearMinuteLine
    ; Reset cursor and try again
    mov cursorPos.X, 0
    mov cursorPos.Y, 2
    INVOKE SetConsoleCursorPosition, hStdOut, cursorPos
    mov eax, COLOR_PROMPT
    call SetTextColor
    jmp MinuteInput

InputComplete:
    ; Clear input lines
    mov cursorPos.X, 0
    mov cursorPos.Y, 1
    INVOKE SetConsoleCursorPosition, hStdOut, cursorPos
    mov ecx, 60
ClearLoop:
    mov al, ' '
    call WriteChar
    loop ClearLoop

    inc cursorPos.Y
    INVOKE SetConsoleCursorPosition, hStdOut, cursorPos
    mov ecx, 60
```

```
ClearLoop2:
    mov al, ' '
    call WriteChar
    loop ClearLoop2

    ; Reset cursor to time display position
    mov cursorPos.X, 6
    mov cursorPos.Y, 0
    INVOKE SetConsoleCursorPosition, hStdOut, cursorPos

    mov alarmFlag, 1
    mov snoozeFlag, 0

    ; Restore normal color
    mov eax, COLOR_NORMAL
    call SetTextColor

    ret
InputAlarmTime ENDP
```

Alarm Trigger and Sound

- Continuously checks if current time matches the alarm time.
- When matched, triggers a looping alarm sound (alarm.wav) using the PlaySound function.
- Displays a message prompting the user to snooze or stop the alarm.

```
CheckAlarm PROC
    cmp alarmFlag, 0
    je NoAlarm

    mov eax, currHour
    cmp eax, alarmHour
    jne NoAlarm

    mov eax, currMin
    cmp eax, alarmMin
    jne NoAlarm

    mov eax, currSec
    cmp eax, 0
    jne NoAlarm

    ; Alarm triggered!
    call TriggerAlarm

NoAlarm:
    ret
CheckAlarm ENDP
```

PlaySound usage:

```
    ; Play alarm sound
    INVOKE PlaySound, OFFSET alarmSoundFile, 0, SND_FILENAME + SND_ASYNC + SND_LOOP
```

```
    ; Stop the alarm sound
    INVOKE PlaySound, 0, 0, SND_PURGE

    ; Play confirmation beep
    INVOKE PlaySound, OFFSET beepSoundFile, 0, SND_FILENAME + SND_ASYNC
```

Snooze Feature

- If the user presses '1', the alarm is snoozed for 2 minutes.
- The new alarm time is calculated and set accordingly.
- Snooze sound (snooze.wav) is played once.

Stores alarm values in global variables (alarmHour, alarmMin).

```
HandleSnooze PROC
    ; Save current cursor position
    push cursorPos.X
    push cursorPos.Y

    ; Add snooze minutes to alarm time
    mov eax, alarmMin
    add eax, snoozeMin
    cmp eax, 60
    jl NoHourAdjust

    ; Adjust hour if minutes overflow
    sub eax, 60
    mov alarmMin, eax
    mov eax, alarmHour
    inc eax
    cmp eax, 24
    jl NoAdjust
    sub eax, 24
NoAdjust:
    mov alarmHour, eax
    jmp SnoozeDone
```

```
NoHourAdjust:
    mov alarmMin, eax

SnoozeDone:
    ; Play snooze sound
    INVOKE PlaySound, OFFSET snoozeSoundFile, 0, SND_FILENAME + SND_ASYNC

    ; Display snooze message
    mov cursorPos.X, 0
    mov cursorPos.Y, 3
    INVOKE SetConsoleCursorPosition, hStdOut, cursorPos

    mov snoozeFlag, 0

    mov cursorPos.X, 0
    mov cursorPos.Y, 3
    INVOKE SetConsoleCursorPosition, hStdOut, cursorPos
    mov ecx, 60
```

```
ClearSnooze:
    mov al, ' '
    call WriteChar
    loop ClearSnooze

    ; Restore cursor position
    pop cursorPos.Y
    pop cursorPos.X
    INVOKE SetConsoleCursorPosition, hStdOut, cursorPos

    ; Restore normal color
    mov eax, COLOR_NORMAL
    call SetTextColor

    ret
HandleSnooze ENDP
```

Stop Feature

- If the user presses '2', the alarm is turned off.
- A message "Alarm Stopped" is displayed.
- Any playing sounds are stopped using SND_PURGE.

Sound Support

- Sound files are played using the PlaySound API
- Any playing sounds are stopped using SND_PURGE.
- Constants like SND_ASYNC, SND_FILENAME, SND_LOOP, and SND_PURGE are defined to control playback behavior.

```
; Sound constants or flags
SND_ALIAS     EQU 00010000h     ;sound resorces
SND_ASYNC     EQU 00000001h     ;play sound without blockage
SND_FILENAME  EQU 00020000h     ;File name for sound
SND_LOOP      EQU 00000008h     ;play sound untill stoped
SND_PURGE     EQU 00000040h     ;stop all instances of sound

; Sound files
alarmSoundFile BYTE "alarm.wav",0
beepSoundFile  BYTE "beep.wav",0
snoozeSoundFile BYTE "alarm.wav",0
```

Console Formatting

- Used colors for better visual distinction.
- Cursor positioning ensures the time is always printed at the same location.
- Cleared screen lines after user input to avoid clutter

```
; Color constants
COLOR_NORMAL      = white + (black * 16)
COLOR_TIME        = lightGray + (black * 16)
COLOR_PROMPT      = cyan + (black * 16)
COLOR_ALARM       = lightRed + (black * 16)
COLOR_SNOOZE      = lightGreen + (black * 16)
COLOR_STOP        = lightBlue + (black * 16)
COLOR_ERROR       = red + (black * 16)
```

```
; Set initial cursor position
INVOKE SetConsoleCursorPosition, hStdOut, cursorPos
```

```
; Set cursor position
mov cursorPos.X, 6
mov cursorPos.Y, 0
INVOKE SetConsoleCursorPosition, hStdOut, cursorPos
```

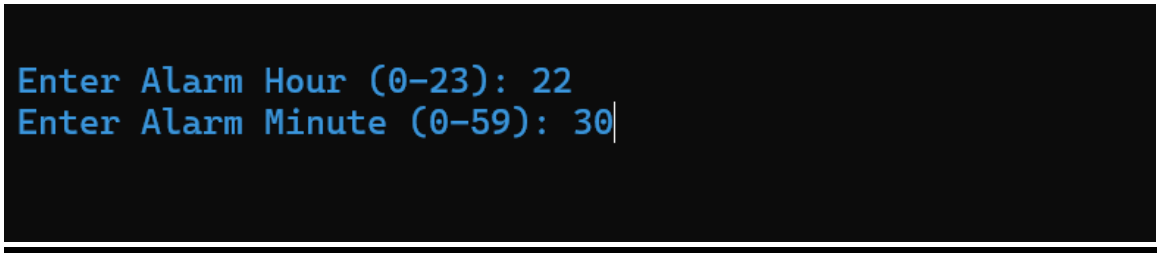
```
; Save current cursor position
push cursorPos.X
push cursorPos.Y
```

```
; Restore cursor position
pop cursorPos.Y
pop cursorPos.X
INVOKE SetConsoleCursorPosition, hStdOut, cursorPos
```

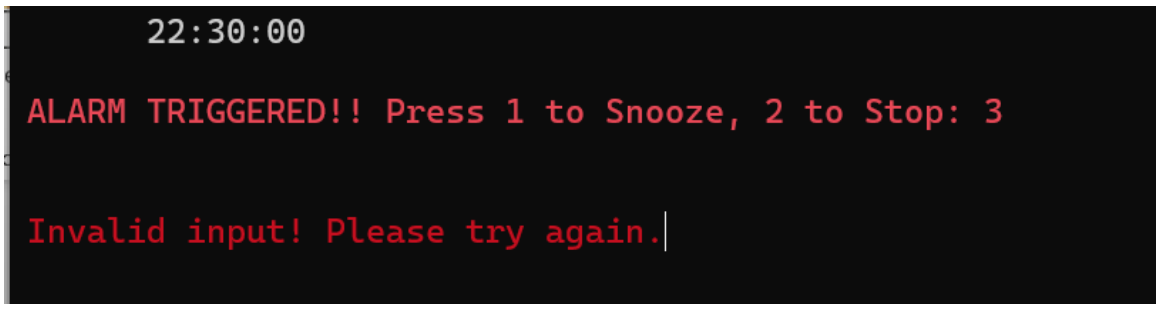
Achievements

- Gained experience working with Windows API in low-level programming
- Integrated real-time time tracking and conditional sound playback.
- Developed features like **snooze**, **stop**, and **sound control**.
- Achieved an interactive and visually clean console interface.

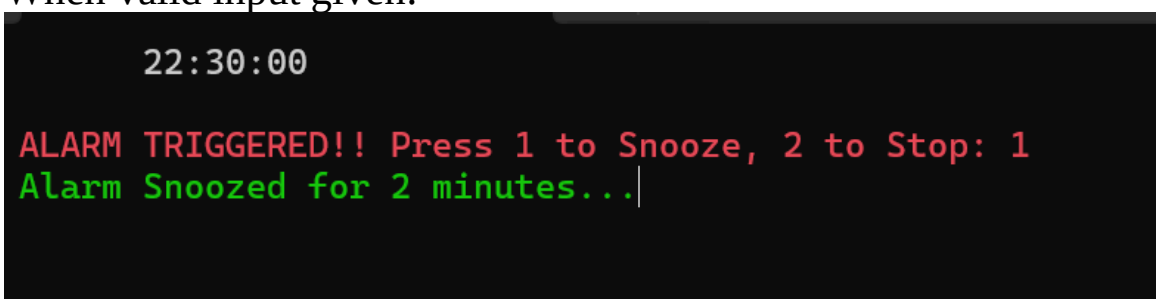
Output on Console

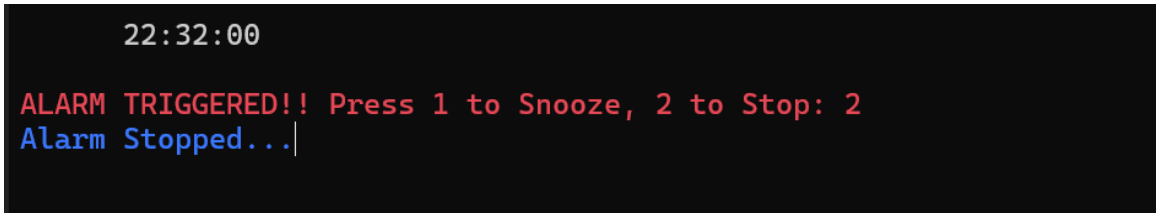


If invalid input given:



When valid input given:





Conclusion

This project successfully demonstrated how an alarm clock system can be implemented in x86 assembly language using Windows API. The final product includes a real-time clock, alarm setting, sound alerts, snooze capability, and interactive user control through keyboard input. It was a valuable learning experience in low-level programming and system interaction with external APIs.