Raport Laboratorium OiAK 2 Zajęcia nr 4

Data wykonania	26.05.2021
Termin zajęć	Czwartek TP 17:05
Autor	Mateusz Kusiak Indeks:252805

1 Treść zadania

Proszę napisać program w języku asemblera w architekturze 32 bit. Program powinien we wskazanym pliku BMP za pomocą techniki steganografii ukryć wiadomość tekstową podaną przez użytkownika. Powinien także umożliwić użytkownikowi odczyt ukrytej wiadomości z pliku BMP. Potrzebne są więc dwa tryby działania. Uwagi:

- Dane ukrywamy na najmłodszych bitach pikseli. Proszę zbadać ile najmłodszych bitów pikseli można wykorzystać, żeby zmiany nie zmieniły znacząco obrazu (empirycznie)
- Proszę pamiętać że pliki BMP mogą mieć różną liczbę bitów na piksel.
- Do wczytywania i zapisywania plików, tudzież wczytywania parametrów z linii poleceń można (ale nie trzeba) użyć kodu w C. Natomiast główny algorytm steganograficzny musi być zrobiony w asemblerze.
- Proszę pamiętać, że pliki BMP mają strukturę i metadane (nagłówki), do operowania na plikach BMP można używać kodu w C.

1.1 Przykład uruchomienia

```
>Prosze podac nazwe zdjecia (np. image.bmp): \image.bmp\
>Co chcesz zrobic?
>1. Zakoduj wiadomosc
>2. Odczytaj wiadomosc
>Opcja nr:

Dla opcji 1:
>Podaj wiadomosc do zapisania: \Ala ma kota\
>Prosze podac nazwe zdjecia po zakodowaniu wiadomosci (np: image2.bmp): \image2.bmp\
>(!) Wiadomosc zakodowana poprawnie!!!

Dla opcji 2:
(wyswietlana jest wiadomosc)
```

2 Objaśnienie

Poniżej znajduje się kod wraz z objaśnieniami:

```
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

//struktura przechowujaca dany pixel w pamieci
struct Pixel
 {
  unsigned char red;
  unsigned char green;
  unsigned char blue;
 };

char* bitMap = NULL;
int width = 0, height = 0, lineSize = 0, sizeOfData = 0;
//tablica przechowujaca 'naglowek zdjecia'
unsigned char header[54];
struct Pixel* photo_pixels = NULL;
```

Listing 1: Sekcja danych potrzebna do obsługi zdjęcia.

```
bool loadBMPfile(char* BmpfielName)
{

...
  //odczywywanie poszczegolnych atrybutow zdjecia za pomoca funkcji asemblerowych
  __asm
  {
    mov   esi, header_address
    // odczyt rozdzielczosci zdjecia - szerokosc
    mov   edi, width_address

    movzx eax, BYTE PTR[esi + 18]
    mov   BYTE PTR[edi], al

    movzx eax, BYTE PTR[esi + 19]
    mov   BYTE PTR[edi + 1], al

    movzx eax, BYTE PTR[esi + 20]
    mov   BYTE PTR[edi + 2], al

    movzx eax, BYTE PTR[esi + 21]
    mov   BYTE PTR[edi + 3], al

    // odczyt rozdzielczosci zdjecia - wysokosc
    mov   edi, height_address

    movzx eax, BYTE PTR[esi + 22]
    mov   BYTE PTR[edi], al

    movzx eax, BYTE PTR[esi + 23]
    mov   BYTE PTR[edi + 1], al

```

```
33      movzx eax, BYTE PTR[esi + 24]
34      mov   BYTE PTR[edi + 2], al
35
36      movzx eax, BYTE PTR[esi + 25]
37      mov   BYTE PTR[edi + 3], al
38
39      // rozmiar widocznego rozmiaru zdjecia (nie liczac naglowka)
40      mov   edi, sizeOfData_address
41
42      movzx eax, BYTE PTR[esi + 34]
43      mov   BYTE PTR[edi], al
44
45      movzx eax, BYTE PTR[esi + 35]
46      mov   BYTE PTR[edi + 1], al
47
48      movzx eax, BYTE PTR[esi + 36]
49      mov   BYTE PTR[edi + 2], al
50
51      movzx eax, BYTE PTR[esi + 37]
52      mov   BYTE PTR[edi + 3], al
53    }
54
55    int* lineSize_address = &lineSize;
56
57    //bajty wype niaj ce mog  by  dodawane na ko cu ka dego wiersza od liczby
        bajt w
58    //mno ymy szerokosc *3 poniewaz kazdy piksel to BGR (trzy kolory)
59    //obliczenie wielkosci dopelnienia w liniach przechowujacych dane o pikselach
60    __asm
61    {
62      mov   esi, width_address
63      mov   eax, [esi]
64      mov   ebx, 3
65      mul   ebx
66      add   eax, 3
67      not ebx
68      and eax, ebx
69      mov   ecx, lineSize_address
70      mov   DWORD PTR[ecx], eax
71    }
72  ...
```

Listing 2: Funkcja odczytująca informacje z pliku BMP.

Funkcja ta jest potrzeba do wczytania zdjęcia o nazwie podanej przez użytkownika. Każdy plik BMP ma nagłowek składający się z: tablicy nagłówka mapy bitowej (14 bajtów)i tablicy informacji o mapie bitowej (40 bajtów). Każdy poszczególny piksel składa się z 3 bajtów reprezentujących każdy kolor R - czerwony, G - zielony, B - niebieski. Przyjmuje on wartości nieujemne 0-255. Zczytujemy z nagłówka informacje o rozmiarze (rozdzielczości) zdjęcia, która jest potrzebna przy odkodowywaniu i zakodowywaniu wiadomości w zdjęciu. Po odczytaniu nagłówka przechodzimy do odczytywania zawartości, czyli kolorów poszczególnych pikseli. Do przechowania informacji posłuży nam struktura Pixel, która zawiera trzy pola odpowiadające kolorom w piksleu.Po przeczytaniu nagłówka należy wykonać bardzo ważny krok, który polega na obliczeniu wypełnienia, ponieważ struktura pikseli jest często przechowywana z dodatkowym bajtem wypełnienia w całej tabeli kolorów.Odczytywanie następuje od lewego dolnego rogu.

```
extern "C" void make_bits(int& pixels_address, int& char_address)
{
  __asm
  {
    // pobranie adresow wiadomosci uzytkownika i adresu tablicy z informacjami o
    pikselach
    mov   esi, char_address
    mov   edi, pixels_address
    //pobranie 4 bitow mniej znaczacych
    movzx eax, BYTE PTR[esi]
    and eax, 0Fh
    movzx ebx, BYTE PTR[edi]
    // ustawianie 4 nizszych bitow w kolorze czerwonym
    and ebx, 0F0h
    or eax, ebx
    // nadpisanie znaku w tablicy pikseli
    mov   BYTE PTR[edi], al
    // pobranie 4 wyzszych bitow z znaku ASCII
    movzx eax, BYTE PTR[esi]
    and eax, 0F0h
    // ustawienie 4 mniej znaczacych bitow w kolorze zielonym
    shr   eax, 4
    movzx ebx, BYTE PTR[edi + 1]
    and ebx, 0F0h
    or eax, ebx
    // nadpisanie znaku w tablicy pikseli
    mov   BYTE PTR[edi + 1], al
  }
}
```

Listing 3: Funkcja przypisująca odpowiednie wartości ASCII czterem mniej znaczącym bitom pikseli.

Funkcja ta dzięki adresowi tablicy z wiadomością od użytkownika koduje wiadomość do zapisanyc kolorów pikseli w tablicy pixels.Działa ona w ten sposób:
Dla znaku A :
0100 0001, czyli w obrazie będzie to wyglądać:
GREEN = xxxx 0100
RED = xxxx 0001
Podstawą algorytmu jest kodowanie najmłodszych 4 bitów dwóch kolorów w pikselu wartościami znaku ASCII w postaci binarnej który ma 8 bitów, co sprawia że kolor niebieski pozostaje nietknięty. Funkcja odkodowywania działa bardzo podobnie.

```
__asm
  {
    // przygotowanie iterator w petli dla tablicy wiadomosci i tablicy struktur
    Pixel
    xor ecx, ecx
    xor edx, edx
    // na pierwszych 4 bajtach bedzie przechowywana dlugosc zakodowanej wiadomosci
    mov   edi, photo_pixels_address
    mov   ebx, msg_len
    mov   DWORD PTR[edi], ebx
    // przechodzimy dalej przez pierwsze dwa piksele
    add   edx, 6
    // poczatek petli kodowana znakow w zdjeciu
    start_loop:
    // jesli koniec wiadomosci to przerwij petle
    cmp   ecx, msg_len
      jge   end_loop
      mov   esi, msg_address
      mov   edi, photo_pixels_address
      // zapisanie adresu piksela do eax
      mov   eax, edi
      add   eax, edx
      // adres znaku do ebx
      mov   ebx, esi
      add   ebx, ecx
      push  ecx
      push  edx
      //kodowanie aktualnego znaku - wywolanie funkcji make_bits
      push  ebx
      push  eax
      call  make_bits
      add   esp, 8
      // wez rejestry ze stosu bo byly uzywane w funkcji make_bits
      pop   edx
      pop   ecx
      //przejdz do kolejnego znaku
      inc   ecx
      //przejdz do kolejnego piksela
      add   edx, 3
      // rozpocznij kodowanie dla kolejnego znaku od poczatku
      jmp   start_loop
      end_loop:
  }
```

Listing 4: Fuckcja posiadająca pętlę w której kodowane są bity w kolorach pikseli wzięte z wiadomości uzytkownika.

Funkcja ta pilnuje aby cała wiadomość została zakodowana w tablicy pikseli która została utwprzona podczas odczytu zdjęcia. Wykorzystuje ona wcześniejszą funkcje make bits która zajmuje sie już kodowaniem jednego znaku w pikselu.

```
void writeBMPfile(char* BmpfielName)
{
  char* photo = NULL;
  // tablica przechowujaca dane mapy bitowej
  photo = new char[sizeOfData];
  // tam gdzie nie sa potrzebne bity wypelniajace czyli gdzie szerokosc jest
    wielokrotnoscia 4
  if (width % 4 == 0)
  {
    //iterator dla kazdego piksela
    int it_px = 0;

    //wczytywanie danych o kolorach kazdeko piksela w mapie bitowej
    for (int pixel = 0; pixel < height * width; pixel++)
    {
      // zapisanie wartosci bajtu niebieskiego
      photo[it_px] = photo_pixels[pixel].blue;
      // zapisanie wartosci bajtu zielonego
      photo[it_px + 1] = photo_pixels[pixel].green;
      // zapisanie wartosci bajtu czerwonego
      photo[it_px + 2] = photo_pixels[pixel].red;

      //przejscie do kolejnych wartosci danego piksela
      it_px = it_px +3;
    }
  }
  //przypadek gdzie bity wypelaniajace sa konieczne
  else
  {
    int it_px = 0;
    int line_nr = 0;
    // przejscie po kazdym pikselu
    for (int pixel = 0; pixel < height * width; pixel++)
    {
      if (pixel == (width * (line_nr + 1)))
      {
        //przechodzienie do kolejnej linii pikseli
        line_nr++;
        // pominiecie bitow wypelnienia
        it_px = 0;
      }
      // zapisanie wartosci bajtu niebieskiego
      photo[(lineSize * line_nr) + (it_px)] = photo_pixels[pixel].blue;
      // zapisanie wartosci bajtu zielonego
      photo[(lineSize * line_nr) + (it_px + 1)] = photo_pixels[pixel].green;
      // zapisanie wartosci bajtu czerwonego
      photo[(lineSize * line_nr) + (it_px + 2)] = photo_pixels[pixel].red;
      // przejiscie do kolejnego pixela
      it_px = it_px + 3;
    }
  }
  // tworzenie nowego pilku bmp z wiadomoscia
  FILE* newBitmapFile;
  // otwarcie go w trybie zapisu binarnego
  fopen_s(&newBitmapFile, BmpfielName, "wb");
  // stworzenie naglowka BMP
  fwrite(&header, sizeof(char), 54, newBitmapFile);
  // przypisanie kolorow pikseli
```

```
58    fwrite(photo, sizeof(char), sizeOfData, newBitmapFile);
59    // zamkniecie pliku z zaszyforwana wiadomoscia
60    fclose(newBitmapFile);
61    delete[] photo;
62    photo = NULL;
63  }
```

Listing 5: Jest to funkcja zapisująca zdjęcie z zakodowaną wiadomością.

Funkcja tworzy nowy plik BMP, który jest stworzony po zmodyfikowaniu kolorów jego pikseli, kótóre już teraz zawierają kod ASCII na pierwszych 4 bitach.

```
1  extern "C" void read_bits(int& pixels_address, int& char_address)
2  {
3    // funkacja w coalosci wykonana w kodzie asemblera
4    __asm
5    {
6      // zaladowanie pustej pamieci na znaki oraz informacje o kolorach
7      // pikseli z ktorych te znaki beda pozyskiwane
8      mov   esi, pixels_address
9      mov   edi, char_address
10     // pobranie 4 mniej znaczacych bitow z koloru czerwonego
11     // ustawienie 4 mniej znaczacych bitow znaku
12     // zapisanie w tablicy znakow
13     movzx eax, BYTE PTR[esi]
14     and eax, 0Fh
15     movzx ebx, BYTE PTR[edi]
16     and ebx, 0F0h
17     or eax, ebx
18     mov   BYTE PTR[edi], al
19     // pobranie 4 mniej znaczacych bitow z koloru zielonego
20     // ustawienie 4 znaczacych bitow znaku
21     // zapisanie w tablicy znakow w miejcu bitow znaczacych
22     movzx eax, BYTE PTR[esi + 1]
23     and eax, 0Fh
24     shl   eax, 4
25     movzx ebx, BYTE PTR[edi]
26     and ebx, 0Fh
27     or eax, ebx
28     mov   BYTE PTR[edi], al
29   }
30 }
```

Listing 6: Funkcja jest odpowiednikiem make bits, różnicą jest jej odwrotne działanie.

Funkcja pobiera 4 najmłodsze bity dwóch kolorów opiksela i łączy je w 8 bitowy znak ASCII
RED = xxxx 0001
GREEN = xxxx 0100

Po przetworzeniu:
0100 0001 = 'A'

```
extern "C" char* readfrom_BMP_message(Pixel * photo_pixels_address)
{
  // stworznie zmiennych potrzbnych w funkcji
  photo_pixels_address = photo_pixels;
  int pixels_array_size = sizeOfData;

  //stworznie tablicy przechowujacej odczytywana wadomocs
  char* msg_address = new char[pixels_array_size];
  int msg_len;

  // kod asemblerowy odczytujacy kazdy pojedynczy piksel
  __asm
  {
    // przygotowanie iterator w petli dla tablicy wiadomosci i tablicy struktur
    Pixel
    xor ecx, ecx
    xor edx, edx
    // zczytanie pierwszych czterech bajtow z dlugoscia wiadomosci z pikseli zdecia
    mov   esi, photo_pixels_address
    mov   ebx, DWORD PTR[esi]
    mov   msg_len, ebx
    // pominiecie dwoch pierwszych pikseli
    add   ecx, 6
    // poczatke petli w ktorej odczytywanie bedzie kazdy piksel
    start_loop :
    // jezeli przeszlismy po wszystkich to koniec operacji
    cmp   edx, msg_len
      jge   end_loop

      mov   esi, photo_pixels_address
      mov   edi, msg_address
      // zapisanie adresu znaku do rejestru ebx i piksela do eax
      mov   ebx, edi
      add   ebx, edx
      mov   eax, esi
      add   eax, ecx
      // zabezpieczenie wartosci relestrow na stosie
      push  ecx
      push  edx
      // odkodowanianie wiadomosci
      push  ebx
      push  eax
      call  read_bits
      add   esp, 8
      pop   edx
      pop   ecx
      // przjescie do kojelnego naku i kolejnego piksela w tablicy
      inc   edx
      add   ecx, 3
      jmp   start_loop
      end_loop :
    mov   BYTE PTR[edi + edx], 0
  }

  return msg_address;
}
```

Listing 7: Fuckcja posiadająca pętlę w której odkodowywane są bity ASCII zapisne w kolorach

pikseli.

Funkcja ta pilnuje aby cała wiadomość została odczytana ze zdjęcia, korzysta z funkcji read bits która odkodowywuje i ustawia odpowiednio bity kodu ASCII z pojedynczego piksela. Jeżeli znak został odczytany, wtedy następuje przeskok do kolejnego.

3 Wnioski

W trakcie wykonywania tego programu na laboratorium miałem problemy z implementacją kodu Asemblera w jezyku C. Po czytaniu i szukaniu informacji stwirdziłem, że łatwiej będzie go napisać na systemie windows. Dzieki składni Intela dużo łatwiejsze okazało się korzystanie ze zmiennych stworzonych w języku C bezpośrednio za pomocą wstawki asemblerowej co okazało się niezbedne podczas wykonywania tego zadania. Nauczyłem się operować na plikach graficznych w sposób binarny i uzywać asemblera w otoczeniu wysokopoziomowego języka C++. Cały algorytm nie jest może najbardziej optymalny, ponieważ lepsze byłoby kodowanie znaku ASCII na najmłodszym bicie wartości przechowującej kolor piksela, jednak okazało się to bardzo trudne ze względu na odkodowywanie i zmianę kolejności poszczególnych bitów, tak aby wyświetlany był prawidłowy znak kodu ASCII.