FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

Semestrální práce z předmětu

Programování v jazyce ANSI C

DIGITÁLNÍ STEGANOGRAFIE

4. ledna 2024

*A*utor:
Oleh Tabachynskyi
A22B0260P

*V*yučující:
Ing. Kamil Ekštein, Ph.D.
`kekstein@kiv.zcu.cz`

Obsah

1 Úvod **2**
1.1 Zadání 2

2 Použité Technologie a Problematika **3**
2.0.1 Struktura BMP Header 3
2.0.2 Struktura DIB Header 3
2.0.3 Struktura BMP File 4
2.1 Komprese 4
2.2 Ukryvání dat v obrázcích 5
2.2.1 BMP 5
2.2.2 PNG 6
2.3 Uložení obrázku 6
2.3.1 BMP 7
2.3.2 PNG 7
2.4 Popis extrakce 8
2.4.1 Extrakce z BMP obrázku 8
2.4.2 Extrakce z PNG obrázku 8

3 Testování a uživatelská příručka **9**
3.1 Testovací funkce 9
3.2 Příklady použití 9
3.3 Možná vylepšení 10

4 Závěr **11**

Kapitola 1

Úvod

1.1 Zadání

Cílem této semestrální práce je vytvořit konzolovou aplikaci v jazyce ANSI C, která bude schopna skrývat a obnovovat soubory s libovolným užitečným obsahem do/z digitálních snímků ve formátech PNG nebo BMP. Implementovaný algoritmus steganografie umožní skrytí zkomprimovaného payloadu do LSB (Least Significant Bit) modrého kanálu pixelů obrázku.

Aplikace bude schopna zpracovávat obrazová data ve formátu 24-bit RGB, což znamená, že bitmapa bude uložena jako sekvence 8-bitových hodnot intenzity červeného, zeleného a modrého kanálu. Před skrýváním obsahu do obrázku se program ujistí, že obrázek splňuje požadavky formátu.

Pro zajištění přenositelnosti by měl být program přeložitelný a spustitelný na platformách Win32/64 (Microsoft Windows NT/2000/XP/Vista/7/8/10/11) a běžných distribucích Linuxu (např. Ubuntu, Debian, Red Hat) s uvedenými specifikacemi pro operační systém Debian GNU/Linux 11.

Aplikace bude spustitelná z příkazové řádky s následujícím formátem:

`stegim.exe ⟨obrázek[.png|.bmp]⟩ ⟨-přepínač směru⟩ ⟨payload⟩`

Kde `⟨obrázek⟩` je název souboru s bitmapovým obrázkem ve formátu PNG nebo BMP, `⟨-přepínač směru⟩` určuje směr operace (skrývání nebo extrakce) a `⟨payload⟩` je soubor obsahující užitečný obsah k ukrytí nebo extrakci.

Kapitola 2

Použité Technologie a Problematika

Pro implementaci manipulace se soubory byla vytvořena samostatná reprezentace formátu BMP. K tomu byly vytvořeny tři struktury: BMPHeader, DIBHeader a BMPFile, které slouží k reprezentaci hlaviček a celkové struktury souboru BMP.

2.0.1 Struktura BMP Header

```
typedef struct bmp_header {
    unsigned char ID[ID_LENGTH];
    unsigned int filesize;
    unsigned int unused;
    unsigned int pixel_offset;
} BMPHeader;
```

Listing 2.1: Struktura BMP Header

2.0.2 Struktura DIB Header

```
typedef struct dib_header {
    unsigned int header_size;
    int width;
    int height;
    unsigned short color_planes;
    unsigned short bits_per_pixel;
    unsigned short comp;
    unsigned int data_size;
    unsigned int pwidth;
    unsigned int pheight;
    unsigned int colors_count;
    unsigned int imp_colors_count;
```

```
} DIBHeader;
```

Listing 2.2: Struktura DIB Header

2.0.3 Struktura BMP File

```
typedef struct bmp_file {
    BMPHeader bhdr;
    DIBHeader dhdr;
    unsigned char *data;
} BMPFile;
```

Listing 2.3: Struktura BMP File

Struktura `bmp_header` souboru obsahuje identifikátor formátu, celkovou velikost souboru, nepoužívaný prostor a offset, kde začínají data pixelů.

Struktura `dib_header` obsahuje informace o velikosti hlavičky, rozměrech obrazu, počtu barevných rovin, počtu bitů na pixel, typu komprese, velikosti datového bloku, rozlišení obrazu v pixelech a informace o použitých barvách.

Struktura `bmp_file` slouží k shlukování informací v hlavičkách a samotných datech pixelů.

Pro práci s formátem PNG byla využita knihovna libpng a její funkce pro vytvoření struktur, zápis a čtení informací o pixelech. Detailní informace o funkcích jsou dostupné v dokumentaci na oficiální stránce: `http://www.libpng.org/pub/png/libpng.html`.

2.1 Komprese

Pro implementaci komprese byl využit algoritmus LZW. Implementace položek tabulky je reprezentována strukturou `input`:

```
typedef struct input {
    int indexValue;     /**< Index hodnoty v tabulce. */
    char *word;         /**< Ukazatel na slovo v tabulce. */
} Table;
```

Listing 2.4: Struktura pro položky tabulky LZW

Algoritmus LZW začíná naplněním tabulky ASCII hodnotami (0-255) pomocí funkcí `allocate_table` a `initialize_table`.

```
Table *allocate_table(void);

int initialize_table(Table **table);
```

Listing 2.5: Inicializace a naplnění tabulky LZW

Následně jsou do tabulky přidány další položky podle principu "najdi nejdelší řetězec, který už je v tabulce, přidej symbol a přidej nový řetězec do tabulky". Tabulka je plněna a vytváří se pole typu `short`. Rozměr tabulky je konstantou 8192, což je standardní rozměr. Největší položka tabulky může být indexována maximálně 13 bity, a délka typu `short` je 2 byty. Takže libovolná položka tabulky se pohodlně vejde do tohoto pole.

Při kompresi se tabulka naplní a zároveň se naplní pole `short`. Při dekompresi máme opět stejné pole typu `short`, a algoritmus se snaží přidat nový kód a starý vypsat jako řetězec. Funkce místo přidání symbolu do pole nebo vypsání do konzole využívá výstupní soubor. Tato implementace není ideální z hlediska principů SOLID v programování. Bylo rozhodnuto se snažit splnit zadání, ale LZW algoritmus byl bohužel realizován nedostatečně kvůli nedostatku zkušeností a obtížné práci s pamětí v jazyce C.

2.2 Ukryvání dat v obrázcích

2.2.1 BMP

Pro ukrytí dat do BMP obrázku byla vytvořena funkce `insert_into_bmp`, která dostává ukazatel na BMP soubor, ukazatel na pole dat (již prošlých kompresí pomocí LZW) a počet prvků v tomto poli.

```
for (j = 0, k = 0; j < data_count; j++)
{
    for (i = BITS_IN_DATA; i >= 0; i--)
    {
        temp = (*(data + j) >> i) & 1;

        if (temp == 0)
        {
            write_bit(file->data + k, 0);
        }
        else
        {
            write_bit(file->data + k, 1);
        }
        k += 3;
    }
}
```

Listing 2.6: Ukryvání dat do BMP

Ukryvání dat do BMP obrázku je poměrně jednoduché. Funkce `insert_into_bmp` projde pole dat, které prošlo kompresí LZW, a každý bit z tohoto pole vloží do modrého kanálu BMP obrázku na jeho nejméně významné bity (LSB).

2.2.2 PNG

Práce s PNG je složitější, neboť formát PNG je komprimovaný a vyžaduje práci s knihovnou. Extrahovaná data jsou reprezentována dvourozměrným polem. Pro ukrytí dat byla vytvořena funkce, která prochází řádek po řádku a do každého třetího bytu vkládá jeden bit dat.

```
while (1)
{
    if (i > data_count)
    {
        break;
    }
    temp = (*(data + i) >> k) & 1;
    if (temp == 0)
    {
        write_bit(&image[y][x], 0);
    }
    else
    {
        write_bit(&image[y][x], 1);
    }
    k--;
    if (k < 0)
    {
        k = BITS_IN_DATA;
        i++;
    }
    x += 3;
    if (x > width)
    {
        y++;
        x = 0;
    }
}
```

Listing 2.7: Ukryvání dat do PNG

Ukryvání dat do PNG obrázku vyžaduje více cyklů kvůli jeho formátu. Funkce pracuje s dvourozměrným polem reprezentujícím obrázek a vkládá bity dat do třetího kanálu (každého třetího bytu) tohoto pole.

2.3 Uložení obrázku

Pro uložení obrázku je vytvořen jednoduchý postup, kde obrázky jsou ukládány do souboru se stejným formátem a názvem "new_file". Zápis je proveden podle následujících pravidel:

2.3.1 BMP

Pro BMP obrázky je zápis jednoduchý. Je třeba otevřít soubor v režimu zápisu a zapsat do něj kompletní strukturu BMP souboru.

```
FILE *output_file = fopen("new_file.bmp", "wb");
fwrite(&bmp_file, sizeof(BMPFile), 1, output_file);
fclose(output_file);
```

Listing 2.8: Uložení BMP obrázku

2.3.2 PNG

Pro PNG obrázky je nutné vytvořit strukturu z knihovny libpng. Struktura z knihovny je naplněna odpovídajícími daty, a poté je přesměrován výstup na otevřený soubor, kam se zapíše každá část PNG struktury.

```
FILE *fp = fopen(file_name, "wb");
png_structp png;
png_infop info;
png = png_create_write_struct(PNG_LIBPNG_VER_STRING, NULL,
   NULL, NULL);
info = png_create_info_struct(png);
png_init_io(png, fp);

png_set_IHDR(
        png,
        info,
        width, height,
        8,
        color_type,
        PNG_INTERLACE_NONE,
        PNG_COMPRESSION_TYPE_DEFAULT,
        PNG_FILTER_TYPE_DEFAULT);
png_write_info(png, info);

png_write_image(png, row_pointers);

png_write_end(png, NULL);

png_destroy_write_struct(&png, &info);

fclose(fp);
```

Listing 2.9: Uložení PNG obrázku

Takto jsou uloženy BMP a PNG obrázky do souboru "new_file"ve stejném formátu jako původní.

2.4 Popis extrakce

Extrakce dat z BMP a PNG obrázků je navržena tak, aby byla co nejjednodušší. Zde jsou popisovány dvě metody extrakce pro oba typy obrázků.

2.4.1 Extrakce z BMP obrázku

Extrakce dat z BMP obrázku je poměrně jednoduchá. BMP soubor se načte a provede se kontrola, zda v nevyužitých bytech existují nějaká data. Pokud ano, přečte se počet informací uložený v těchto bytech. Poté jsou postupně čteny pouze nejnižší bity modrého kanálu a po dosažení 13 bitů jsou zapsány do pole dat. Následně je provedena dekomprese a výsledek je zapsán do souboru, který je určen třetím parametrem. Pseudokód:

```
BMPFile bmp_file = read_bmp("input.bmp");

if (bmp_file.bhdr.unused > 0) {
    size_t data_count = bmp_file.bhdr.unused;
    short *extracted_data = extract_from_bmp(&bmp_file,
   data_count);

    decompress(extracted_data, data_count, "output.txt");
}
```

Listing 2.10: Extrakce z BMP obrázku

2.4.2 Extrakce z PNG obrázku

Extrakce dat z PNG obrázku je trochu složitější kvůli struktuře dat v dvojrozměrném poli. Při extrakci jsou načteny 13 bitů, které jsou následně dekomprimovány a zapsány do souboru. Pseudokód:

```
PNGImage png_image = read_png("input.png");

size_t data_count = calculate_data_count(png_image.width,
   png_image.height);
short *extracted_data = extract_from_png(&png_image,
   data_count);

decompress(extracted_data, data_count, "output.txt");
```

Listing 2.11: Extrakce z PNG obrázku

Takto jsou popsané postupy pro extrakci dat z BMP a PNG obrázků, které jsou následně dekomprimovány a uloženy do výsledného souboru.

Kapitola 3

Testování a uživatelská příručka

V této kapitole jsou prezentovány testovací funkce, které byly vytvořeny pro ověření funkcionality programu. Kromě toho jsou poskytnuty informace o jednotlivých funkcích, včetně popisu a příkladů použití.

3.1 Testovací funkce

Pro ověření správného chování programu byly implementovány testovací funkce, které vypisují do standardního výstupu informace o obrazcích, vytvoření struktur, alokaci a uvolnění paměti. Každá testovací funkce je stručně popsána, ale vzhledem k dlouhé době psaní kódu může docházet k neshodám v signaturách různých funkcí.

Přestože jsou komentáře v kódu, které umožňují porozumět, jak se jednotlivé funkce používají, některé funkce neposkytují dostatečné informace o průběhu operace. Tato neshoda ve vracených hodnotách a nedostatečné informace o průběhu operace mohou být považovány za oblasti k možnému vylepšení v budoucích verzích programu.

3.2 Příklady použití

Pro ukázku použití několika funkcí ve vašem programu jsou zde uvedeny příklady s komentáři. Je doporučeno pečlivě studovat komentáře k funkcím pro správné použití.

```
void test_create_bmp() {
    BMPFile bmp_file = create_bmp(800, 600);
    print_bmp_info(bmp_file);
    free_bmp(&bmp_file);
}
```

Listing 3.1: Příklad použití testovací funkce pro vytvoření BMP souboru

Tento příklad ukazuje použití testovací funkce pro vytvoření BMP souboru, vypsání informací o něm a následné uvolnění paměti. Podobně jsou implementovány další

testovací funkce pro různé části programu.

3.3 Možná vylepšení

V budoucích verzích programu by bylo vhodné zvážit sjednocení signatur funkcí, přidání více informací o průběhu operací a zlepšení konzistentnosti návratových hodnot. Tyto vylepšení by přispěla ke snazšímu porozumění a používání programu.

Kapitola 4

Závěr

Implementovaný program a jeho kód, jak byly popsány v předchozích sekcích, nesplňují všechny požadavky semestrální práce. Některé nedostatky jsou způsobeny nedostatkem zkušeností a obtížností práce s vysokými nároky na implementaci. Například nebylo dosaženo cíle umožnit uživatelům umístit přepínače na libovolnou pozici pro kódování a dekódování. Metody pro označení přítomnosti skrytých dat v obraze a kvantifikaci těchto dat nebyly implementovány. Navíc implementace LZW algoritmu není optimální.

Na pozitivní stranu lze poznamenat, že práce s BMP soubory byla popsána poměrně detailně. LZW algoritmus byl úspěšně rozdělen na jednoduché bloky. Nicméně kvůli cizímu kódu v knihovně pro práci s formátem PNG se nám nedařilo dosáhnout uspokojivých výsledků. Bylo obtížné nalézt kvalitní a stručnou dokumentaci popisující funkce této knihovny.

Přestože implementovaný program splňuje minimální požadavky stanovené pro semestrální práci, existuje prostor pro vylepšení. V budoucích verzích by bylo vhodné lépe zdokumentovat kód, přidat podrobnější popisy funkcí a dosáhnout vyšší kvality implementace. Tato zkušenost poskytla cenné poznatky, které lze využít při práci na podobných projektech.