Gyönyörű Zsigmond Ádám
V7DCHS
csütörtök 14:00

Rendszerközeli programozás program dokumentáció

(rövidített kiadás)

RKP main.c

Az RKP main.c program képes egy tömörítetlen TrueColor BMP képfájlból rejtett, kódolt szöveget kinyerni dekódoló megoldással. Mindezt egy HTTP POST metódussal el is küldi egy webszerverre. A projekt egy kurzushoz készült.

A fordítás menete

```
make clean
make
```

A `make clean` csak akkor szükséges, ha valami módosítást hajtottunk végre a programon korábban. Ekkor törlődnek az előző `make` parancs után legenerálódott segédfájlok. A Makefile a fordítás megkönnyítése érdekében jött létre.

Amennyiben **nem** a Makefile segítségével szeretnénk lefordítani a programot:

```
gcc main.c libbmpencoder.c -fopenmp -o go
```

Ahol a `go` nevű állomány lesz futtatható. Miután a fenti parancsot kiadtuk a program ugyanúgy fog viselkedni a különböző parancssori argumentumokra és egyéb kapcsolókra, mint amikor a Makefile segítségével fordítunk. (A fent említett futtatható állomány neve helyett megadhatunk valami mást is tetszés szerint természetesen.)
A Makefile a clang fordítót használja, de ahogy fent látható a gcc is remek alternatíva.

Használat

A program rendkívül informatív. Nem csak hiba esetén, hanem általánosan is.

Kapcsolók

`--help`	Információ a futtatás lehetséges opcióiról
`--version`	Kiíratásra kerül a program verziószáma, elkészültének dátuma és a fejlesztő neve (sorrendhelyesen)

Amennyiben argumentumként egy BMP képfájl nevét adjuk meg akkor a program megpróbálja megnyitni azt. Ettől függetlenül a két kapcsoló használható bármikor, első, második vagy akár harmadik argumentumként is. Nem okoz problémát a programnak, le van kezelve. Amennyiben a képfájl nevét elgépeltük, nincs

jogosultságunk megnyitni, illetve egyéb felmerülő hibák esetén tájékoztatást kapunk a program részéről.

Ha nem adunk meg képfájlt argumentumként és a két kapcsoló semelyikét sem használjuk, akkor egy karakteres formalitású tallózó felület jön be, ahol mi magunk kiválaszthatjuk a kívánt képfájlt.

Amennyiben itt is elgépelés történik, vagy a root könyvtár felé próbálunk lépni, esetleg a képfájlt nem sikerül megnyitni tájékoztatást kapunk.

A képfájl sikeres megtalálása után

Egy ideig nem leszünk képesek kilőni a programot a CTRL+C billentyűkombinációval de ne essünk kétségbe, amennyiben sikerült a küldés nyugtát fogunk kapni a terminálunkra „Sikeres üzenetküldés!”. Azonban, ha túl sokáig tart a dekódolás folyamat akkor ebben az esetben a program megteszi ezt helyettünk. Ez az időtartam pontosan 1 másodpercnek felel meg.

A sikeres küldés végeztével

Egy log.txt file készül, amely a dekódolási folyamat során végzett párhuzamos programozásról szól. A szálak sorszáma szerepel benne, illetve az, hogy a szálak mennyiszer dolgoztak külön-külön.

A fentiek ismerete elegendő az átlagfelhasználó számára. A továbbiakban részletesebben fogom tárgyalni a különböző programegységeket.

Lehetséges visszaadott értékek az operációsrendszer felé

Az egyes visszaadott értékek az EAX regiszterben lesznek találhatóak szokásosan.

Kód	Jelentés
1	Nem sikerült a memóriafoglalás
2	Nem sikerült a socketet létrehozni
3	A megadott host nem létezik
4	Nem sikerült csatlakozni (socket)
5	Nem sikerült üzenetet írni a socketre
6	Nem sikerült üzenetet olvasni
7	Nem sikerült üzenetet írni a szerverre
10	Rossz fájlformátum
11	Nem sikerült a .env fájlt megnyitni
12	Rossz parancssori argumentum
13	Nincs megfelelő jogosultság a fájlhoz
14	Nem sikerült fájlt megnyitni a tallózóban

Eljárások és függvények

Eljárások (nincs visszatérési érték)

Formázott hibakiírást megvalósító eljárás, ahol a(z)
const char *msg a kiírandó hibaüzenetet jelöli
int status pedig a hiba kódját

```
void error(const char *msg, int status) {
    fprintf(stderr, "%c[1m", ESC);
    fprintf(stderr, "RKP main.c: ");

    fprintf(stderr, "\033[0;31m");
    fprintf(stderr, "fatal error: ");
    fprintf(stderr, "%c[0m", ESC);

    fprintf(stderr, "\033[0m");

    fprintf(stderr, "%s\n", msg);

    exit(status);
}
```

A BrowseForOpen() függvény hibakiírója, ahol a(z)
int fp egy 32 bites egész, de egy fájleírónak felel meg valójában

```
void PrettyErrorChecker(int fp) {
    if (fp < 0) {
        char *error_msg = "error during file opening
(BrowseForOpen())\ncompilation terminated.\n";

        perror("Failed: ");

        error(error_msg, 14);
    }
}
```

SIGALRM és SIGINT jelekre készíti fel a főprogramot, ahol
int sig egy szignál

```
void WhatToDo(int sig) {
    if (sig == SIGALRM) {
        error("Hiba: a program túl sokáig futott, ezért leáll!\n", 7);
    } else if (sig == SIGINT) {
        pid_t pid;
        pid = fork();
        if (pid == 0) {
            printf("A program INTR (interrupt -> ctrl+c) karakterrel sem
állítható le!\n");
            kill(getpid(), SIGKILL);
        }
    }
}
```

A kép bitmap header mező adatait írja ki formázottan, ahol
bitmap_header container az adatokat szolgáltató struktúra

```
void pretty_header_print(bitmap_header container) {
    printf("Kép aláírása: %s\n", container.signature);
    printf("Fájlméret (bájtokban): %d\n", container.file_size);
    printf("Rejtett karakterek száma a képben: %d\n",
container.number_of_hidden_chars);
    printf("Eltolás (offset) mérete: %d\n",
container.pixel_array_offset);
}
```

Függvények (van visszatérési érték)

Visszatérési értéke logikai igaz (nem nulla egész szám), amennyiben reguláris fájlról van szó. Logikai hamis (azaz nulla szám szerint) amennyiben egy könyvtár.
const char *path az út, amin a vizsgálandó fájl vagy könyvtár található

```
int IsRegularFile(const char *path) {
    struct stat path_stat;
    stat(path, &path_stat);
    return S_ISREG(path_stat.st_mode);
}
```

A következő függvények elég részletesek, emiatt csak a formális paraméterlista kerül mutatásra. Egyes függvények csak tömören kerülnek bemutatásra, a formalitás betartása végett (rövidített változat).

A függvény a parancssori argumentumokat kezeli. Ezek alapján ad utasítást. Ahol a(z)
int argc a parancssori argumentumok számát jelöli (argv külső tömbjének mérete)
char **argv egy kétdimenziós char tömb, amely a parancssori argumentumokat tárolja.

Lehetséges return értékei a fő programegység felé:
111 -> Browser függvényt kell meghívni a továbbiakban
222 -> Képet adtunk meg parancssori argumentumként
1337 -> Ha valami nem volt rendben akkor a fájl nem kerül megnyitásra és ez az érték térül vissza. (ez a main()-ben van lekezelve)
Kép indexe -> Ha minden rendben volt akkor az argv tömbön belül visszaadja a kép indexét a főprogram felé

```
int HandleArgv(int argc, char **argv);
```

Karakteresen megvalósított fájltallózó függvény szabványos UNIX clear paranccsal kiegészítve. (Azaz windows-zal nem kompatibilis.)
Visszatérési értéke egy olyan 32 bites egész, ami file leíróként funkcionál.

```
int BrowseForOpen();
```

A HTTP metódusok egyikét megvalósító függvény. A HTTP POST-ot adatküldésre használjuk egy szerver felé, vagy valamilyen erőforrás kreálására, frissítésére. A függvény 0-val tér vissza, ha sikeres volt a küldés (erről nyugta is készül kiíratás formájában).

char *neptunID Neptun azonosító
char *message a küldeni kívánt üzenet
int NumCh a fent lévő message sztring (karaktertömb) mérete

```
int Post(char *neptunID, char *message, int NumCh);
```

Visszatérési értéke egy bitmap_header struktúra.
const u_char *array korábban binárisan beolvasott bájtokat tartalmaz. Ha ezt a függvényt kombináljuk a **void pretty_header_print**(bitmap_header container); eljárással akkor egy rendkívül szép, formázott eredményt találunk az alapértelmezett kimeneten a BMP képfájlról.

```
bitmap_header pretty_formatter(const u_char *array);
```

Komplex, generikus párhuzamos programozás során használt folyamatokkal ellátott függvény. Visszatérési értéke egy olyan karaktertömb, amely az elküldésre kész kikódolt üzenetet tartalmazza.
char *Pbuff nyers pixel_array tömb.
int NumCh kódolt karakterek száma. (A méret ennek a számnak a háromszorosa, hisz 3 szín alkot egy pixelt (és a padding, ha van)).
Az 1-es hibakód minden esetben a memória allokálás hibáját jelzi. A háttérben elkészül egy log.txt is, amely a párhuzamos programozás hitelességét jelzi.

```
char *Unwrap(char *Pbuff, int NumCh);
```

Bináris olvasást végző függvény. Visszatérési értéke a nyers pixel_array.
int f 32 bites egész, ami fájleíróként funkcionál.
int *NumCh kódolt karakterek száma, outputként működik.
A függvény az operációs rendszer felé 0-val tér vissza, ha nem tartalmaz rejtett szöveget a kép.

```
char *ReadPixels(int f, int *NumCh);
```

Az alábbi Makefile-t használom fordításra. Itt jól látható, hogy fordítónak a clang van beállítva.

```
CC=clang
CFLAGS=-Wall -g -fopenmp
BINS=main libbmpencoder.so
all: $(BINS)

libbmpencoder.o: libbmpencoder.c bmpencoder.h
	$(CC) $(CFLAGS) -c libbmpencoder.c

libbmpencoder.so: libbmpencoder.c bmpencoder.h
	$(CC) $(CFLAGS) -fPIC -shared -o $@ libbmpencoder.c -lc

main: main.c libbmpencoder.o
	$(CC) $(CFLAGS) -o $@ $^

clean:
	rm *.o *.so main
```

Az alábbi .env fájlt használom az adatok kiíratására. Azért nem alkalmaztam sima kiíratást, mert úgy a forráskódba lettek volna beleégetve ezen adatok, illetve így titkosítani is lehet őket. Ez a megfelelő függvényben tokenizálásra kerül.

```
VERSION_NUMBER=1.0.0
PROJECT_FINISHED_DATE=2021 May 2.
AUTHOR=ZSIGMOND Adam Gyonyoru
```

Köszönöm, hogy megtekintette a programom dokumentációját!