**Rendszerközeli programozás**

**Projekt Dokumentáció**

Készítette: Berencsi Csaba

Neptun-kód: YKS33Y

Dátum: 2021.04.25.

A projektem három állományból épül fel:

1. project.c
2. myHeader.h
3. Projekt dokumentáció(Berencsi\_Csaba\_YKS33Y).docx

A projektet **Linux** operációs rendszer alatt**, VsCode**-ban fejlesztettem.

*Hasznos/szükséges kapcsolók*:

* -fopenmp(!)

gcc project.c -o project -fopenmp

* -o

gcc feladat1.c -o feladat1 🡪 ./feladat1

**A projekt feladat több módon is futtatható**:

* „--help” 🡪 Kezelési útmutató.
* „--version” 🡪 A projekt verziószámának, a készítés/frissítés dátumának, illetve a fejlesztő nevének lekérdezése.
* filename.bmp 🡪 A program könyvtárában található bmp képfájl.
* Ha nem adunk meg parancssori argumentumot, akkor a futtatott program a fájltallózót indítja el.

A program futtatása során(az utóbbi 2 esetben) a képfájl beolvasásra kerül, és a képfájlban lévő szöveget a program dekódolja.

A projekt függvényeit/eljárásait egy külön header állományba, a „myHeader” nevű állományba szerveztem ki. A main.c pedig a főprogram kódjait tartalmazza.

**A főprogram(projekt.c) felépítése**:

A főprogram fejléce adott: int main(int argc, char\* argv[])

A main függvény kezeli a parancssori argumentumokat, és egyetlen egy állományt inkludál(myHeader.h).

Ha semmi hiba nem keletkezett futáskor, akkor 0-ás értékkel tér vissza a program. Ha viszont hiba keletkezett futtatáskor, akkor 1-es értékkel tér vissza.

Továbbá a főprogram 1 másodperces időzítőt állít be, amit a lefutást követően ki is kapcsol. Ha ez a lefutás nem fejeződik be adott idő alatt, akkor a SIGALRM nevű szignál hatására leáll, 1-es értéket adva vissza.

**Az alprogramok(myHeader.h)felépítése**:

A myHeader.h összesen 5 alprogramból áll:

* BrowseForOpen
* ReadPixels
* Unwrap
* Post
* WhatToDo

1. int BrowseForOpen()

Ez az alprogram kilistázza a HOME könyvtár tartalmát, illetve bekéri az adott könyvtár/fájl nevét. Ha könyvtár nevet ad meg a felhasználó, akkor abba a könyvtárba be is lép, és kilistázza az adott könyvtár tartalmát. A fájltallózó akkor zárul be, ha a felhasználó megtalálja, és begépeli az adott .bmp fájlt. Ezt a .bmp fájlt csak olvasásra, binárisan nyitja meg.

1. char\* ReadPixels(int f, int\* NumCh)

A fájl beolvasását követően ez a függvény meghívásra kerül. Az alprogram a fájlban kódolt karakterek számát határozza meg, illetve ezen karaktereknek szükséges memóriaterületet foglal le. A visszatérési értéke a lefoglalt(és feltöltött) memóriaterület címe.

1. char\* Unwrap(char\* Pbuff, int NumCh)

A ReadPixels függvény lefutása után hívódik meg. A Pbuff pointer a ReadPixels által feltöltött memóriaterület címére mutat. A NumCh pedig a kódolt karakterek száma. A kapott memóriacímen található adatot bitműveletek segítségével feldolgozza, majd az új memóriaterületre menti. Ezután a Pbuff által mutatott memóriaterületet felszabadítja. Visszatérési értéke a számítás által kapott memóriaterület címe.

1. int Post(char\* neptunID, char\* message, int NumCh)

Feladata, hogy a webszerverrel felvegye a kapcsolatot. Ennek alapja a socket programozás. Egy TCP kapcsolatot hoz létre a webszerverrel, és egy üzenetet küld a címre. Sikeres fogadás/küldés esetén a socketet bezárja. Visszajelzésként értesíti a felhasználót a sikeres/sikertelen üzenetküldésről.

1. void WhatToDo(int sig)

Szignálkezelő eljárás. A szignálok alapértelmezett módon működnek, beérkező szignálok esetén a feldolgozni kívánt fájl beolvasása után hívható meg.

Két fajta szignált képes kezelni: SIGALRM és SIGINT. A ***SIGALRM*** jelez a felhasználónak, hogy a képfájl feldolgozására lejárt az időkorlát, és 1-es értékkel megszakítja a folyamatot. A ***SIGINT*** esetén forkolást alkalmazva elindít egy gyermek folyamatot(értesíti a felhasználót, hogy a program nem állítható le ctrl + c billenytűkombinációval). A gyermek folyamat ezt követően a saját futását leállítja a ***SIGKILL*** szignállal.