Rendszerközeli Projektmunka
Dokumentáció

Tartalomjegyzék

A program két üzemmódban működik:

Az adatok előállítása és továbbítása a másik folyamatnak: küldő mód.
Az adatok fogadása és megjelenítése a fogadó folyamatban: fogadó mód.

Az adatok átvitele fájl vagy socket segítségével történik.

Használati utasítás

A program futtatása előtt a következő függőségek szükségesek:
gcc (C fordító) *[gcc (Ubuntu 11.3.0-1ubuntu1~22.04) 11.3.0]*
make (build eszköz) *[GNU Make 4.3]*

A program forráskódját letölthetjük a Github tárolóból, majd a forráskód könyvtárában a következő parancsokkal fordíthatjuk és futtathatjuk a programot:

```
make
./chart [-send|-receive] [-file|-socket]
```

Opcionális kapcsolók:

```
--help
--version
```

Program használata:

2 terminál megnyitása
A, Terminál
./chart -send [-file|-socket]
B, Terminál
./chart -receive [-file|-socket] //Az A, Terminálban kiválasztott mód

Ezek után a számítások el lesznek végezve, a fájlok továbbítv lesznek. Ezek után a program 0-ás hibakóddal leáll

Error kódok:

exit(1): Hibás parancssori argumentumok vagy programnév került megadásra. exit(2): Nem található receive módban futó chart program, ezért a művelet nem hajtható végre. exit(3): A megadott fájlt nem sikerült megnyitni. exit(4): Nem sikerült létrehozni a socketet, ami szükséges a kommunikációhoz. exit(5): Küldési hiba történt a Socketen keresztül. exit(6): Fogadási hiba történt a Socketen keresztül. exit(7): A kapott értékek nem egyeznek meg. exit(8): Nem sikerült bindelni a socketet, ami szükséges a kommunikációhoz.utyric ac exit(9): Nem sikerült kapcsolatot létrehozni a szerverrel. exit(10): A program nem találta a szükséges signált a végrehajtáshoz.

Használt alprogramok:

void BMPcreator(int *Values, int NumValues):

A megadott kód C-ben íródott, és egy BMPCreator nevű függvényt tartalmaz, amely egy BMP képfájlt hoz létre egy egész számokból álló tömb alapján.

A függvény először kiszámítja a szükséges fájlméretet a számok száma és a kép szélessége alapján, majd a megfelelő memóriaterületet lefoglalja a bitképnek a calloc segítségével, majd inicializálja a bitkép fejlécét és metaadatait.

Ezután a függvény végigmegy az egész számok tömbön, és beállítja a megfelelő bitet a bitkép pixeladataiban a szám pozíciója és eltolása alapján.

Végül a függvény megnyitja a chart.bmp nevű fájlt, írja a bitkép adatát a fájlba a write függvénnyel, majd bezárja a fájlt. A lefoglalt memóriaterületet pedig felszabadítja.

void write_help():

A program használati útmutatóját írja ki, illetve leírja a lehetséges hibakódokat és azok jelentését.

int namecheck(int argc, char *argv[]):

A függvény egy programparaméter-ellenőrzést hajt végre. Először megvizsgálja, hogy a futtatható állomány neve "chart"-e, ha nem, akkor hibát jelez és kilép. Ha a programnak vannak parancssori argumentumai, akkor végigiterál rajtuk és ellenőrzi, hogy azok megfelelőek-e. Ha bármelyik argumentum hibás, akkor hibát jelez és kilép a programból, miután kiírta a program használati útmutatóját. Ha az első argumentum "--version", "--help" vagy "--tutorial", akkor azokhoz tartozó információkat írja ki és kilép a programból. A függvény visszatérési értéke nincs megadva, valószínűleg a void típusú függvényekhez tartozik.

int checkdup(int argc, char *argv[])

A függvény azt ellenőrzi, hogy a parancssori argumentumok között van-e duplikáció. Végigmegy a parancssori argumentumokon, és minden argumentumot összehasonlít az utána következőkkel. Ha talál két azonos argumentumot, növeli a dup változó értékét eggyel. Ha a végén a változó értéke legalább 1, akkor visszatér 1-gyel, ami azt jelzi, hogy van duplikáció. Ha nincs duplikáció, akkor visszatér 0-val.

void signal_handeler(int sig):

Ez egy signal handler függvény, ami arra szolgál, hogy kezelje a különböző jeleket, amiket a processz kap. Ha a kapott jel SIGUSR1, akkor meghívja a ReceiveViaFile függvényt. Ha a jel SIGINT vagy SIGTERM, akkor kiírja a "Sikeresen leállítva!" üzenetet és kilép a programból. Ha a jel SIGALRM, akkor "Nem sikerült a kapcsolatot létrehozni!" üzenetet ír ki és kilép a programból a hibakóddal 9. Ha a kapott jel nem az előzőek egyike, akkor "A program nem talált signalt!" üzenetet ír ki és kilép a programból a hibakóddal 10.

void modecheck(int *modes, int argc, char *argv[]):

A függvény megvizsgálja, hogy a program milyen módban fut, és milyen kommunikációs módot használ.

Az alapértelmezett üzemmód a küldő üzemmód, az alapértelmezett kommunikációs mód pedig a fájl.

void execute_commands(int modes[]):

Ez egy függvény, amely végrehajtja a parancsokat a programban a paraméterként kapott "modes" tömb alapján. A "modes" tömb egy négy elemű tömb, amelynek minden eleme egy bit (0 vagy 1), amely jelzi, hogy a program melyik módokat hajtja végre:

- modes[0] == 1: a program adatokat küld
- modes[1] == 1: a program adatokat fogad
- modes[2] == 1: a program fájlon keresztül kommunikál
- modes[3] == 1: a program socketen keresztül kommunikál

Ha a modes[0] és modes[2] egyaránt 1, akkor a program előállít egy értékkészletet a "Measurement" függvény segítségével, majd elküldi azt fájlon keresztül a "SendViaFile" függvény segítségével. Ha a modes[0] és modes[3] egyaránt 1, akkor a program ugyanazt az értékkészletet küldi el socketen keresztül a "SendViaSocket" függvény segítségével.

Ha a modes[1] és modes[2] egyaránt 1, akkor a program várakozik a SIGUSR1 szignálra (melyet a "signal_handeler" függvény kezel), majd a "pause" függvény segítségével felfüggeszti a program futását addig, amíg meg nem érkezik az adatok. Ha a modes[1] és modes[3] egyaránt 1, akkor a program fogadja az adatokat socketen keresztül a "ReceiveViaSocket" függvénnyel.

char *concat(const char *s1, const char *s2):

Két sztringet (s1 és s2) összefűz egy új sztringbe, és visszatér ezzel az új sztringgel.

int FindPID():

egy másik chart folyamat azonosítóját keresi a /proc fájlrendszerben. A függvény először megnyitja a /proc könyvtárat, majd végigiterál a benne található fájlokon és könyvtárakon.

void write_int(char *p, int value):

Egy egész számot (value) ír ki egy karaktertömbbe (p) little-endian byte rendezéssel.

int calculate_size(int min, int sec):

Kiszámolja egy tömb méretét, amelyet a függvényhívó létre fog hozni. A függvény két bemeneti értéket vár: az első az eltelt percek száma (min), a második pedig az eltelt másodpercek száma (sec). A függvény az eltelt percek számától függően különböző méreteket ad vissza a tömbnek.

int Measurement(int **p_values):

A függvény létrehoz egy dinamikus tömböt (a méretét a rendszeróra pillanatnyi állapotától függően határozza meg), majd feltölti ezt a tömböt véletlenszerűen generált értékekkel

void SendViaFile(int *Values, int NumValues):

A függvény először meghívja a "FindPID" nevű függvényt, hogy megkapja egy diagram program folyamat azonosítóját, amely várakozó módban fut.

void ReceiveViaFile(int sig):

A függvény megnyitja a "Measurement.txt" fájlt, amelyben egész számok találhatók. Az adatokat egy dinamikus memóriaterületen tárolja és kiírja a konzolra. Ezután meghívja a "BMPcreator" függvényt, amely egy BMP képet készít az adatokból. Végül felszabadítja a dinamikusan lefoglalt memóriát.

void SendViaSocket(int *Values, int NumValues):

A "SendViaSocket" függvény egy UDP socketet használva küldi el az egész számokat tartalmazó tömböt a "127.0.0.1" IP-című szervernek. A kapcsolatot az adatok számával kezdi, majd vár a szerver válaszára. Ha a válasz tartalmazza a helyes értéket, elküldi az adatokat a szervernek. Ha bármilyen hiba történik, akkor hibaüzenetet ír ki és kilép a programból.

void ReceiveViaSocket():

Ez egy UDP Socket program, ami adatokat küld és fogad a kliens és a szerver között. A kliens küld egy tömböt a szervernek, majd a szerver ezt a tömböt fogadja és létrehoz belőle egy BMP fájlt. A kommunikációhoz a program a socket API-t használja.