# *Rendszerközeli programozás*

*Dokumentáció*

*Tóth Zsolt – HL50I9*

## Fordítóprogram kapcsolói:

*Fordítóprogram: GCC Compiler 11.4.0*

## Kapcsoló:

- *-o*
- *-fopenmp*

## Rendszer követelmények:

*Linux operacios rendszer, Ubuntu (22.04)*

## Felhasználói útmutató:

*A programot a következő paranccsal lehet fordítani és futtatni:*

*gcc rkp.c main.c -o chart -fopenmp*

*A programunk egy bmp kiterjesztésű képet fog létrehozni. A programot kizárólag "chart" névvel lehet elindítani, egyéb néven nem fog működni (ezért kell a –o kapcsoló).*

*Miután sikeresen elindítottuk a programot, a felhasználó az alábbi parancssori argumentumokat használhatja:*

- *--version: A program verziójának, kiadásának és fejlesztőjének lekérdezése.*
- *--help: További súgókérésekhez.*
- *-send: fogadó üzemmód indításához*
- *-receive: küldő üzemmód indításához*
- *-file: file módba vált*
- *-socket: socket módba vált*

*Két terminálra lesz szükség, az egyik a küldő, a másik a fogadó.*
*Először elindítjuk a fogadót, vagy fájlkezeléssel vagy hálózaton keresztül.*

**Fogadó folyamat indítása:**

- *Fájl mód: ./chart  -file -receive*
- *Socket mód: ./chart -socket -receive*

**Küldő folyamat indítása:**

- *Fájl mód: ./chart -file -send*
- *Socket mód: ./chart -socket –send*

**A program kimenetei és azok jelentése:**

- *exit(0) (EXIT_SUCCES): A program sikeresen leállt.*
- *exit(1) (EXIT_FAILURE): A fájl alapú küldési szolgáltatás nem érhető el!*
- *exit(2): Socket létrehozási hiba!*
- *exit(3): Csatlakozási hiba!*
- *exit(4): Szerver időtúllépés!*
- *exit(5): Küldési hiba!*
- *exit(6): Fogadási hiba!*
- *exit(7): Méret hiba!*

# Alprogramok rövid leírása:

**void print_version_info():**

*Az alprogram célja, hogy kiírja a program verzióját, készítési dátumát, valamint a fejlesztő nevét és Neptun kódját.*

*Egy fontos részlet ebben az alprogramban az OpenMP szakaszok használata, amely lehetővé teszi a párhuzamos végrehajtást. Ezáltal a printf utasítások egymástól függetlenül, több szálon futnak.*

```
#pragma omp parallel sections

{

        #pragma omp section

                printf("Version: %s\n", VERSION_NUMBER);

        #pragma omp section
```

```
            printf("Date: %s\n", COMPLETION_DATE);

        #pragma omp section

            printf("Developer and neptun code: %s %s\n", DEVELOPER_NAME,
                                                        NEPTUN_CODE);
}
```

*Paraméterek: Nincsenek.*

*Visszatérési érték: Nincs.*

**void print_help_info():**

*Az eljaras program használati útmutatóját nyomtatja ki a konzolra.*

*A karakterláncok itt a parancsokat és azok rövid leírásait tartalmazzák, ami segíti a felhasználók megértését és az alkalmazás helyes használatát.*

*Paraméterek: Nincsenek.*

*Visszatérési érték: Nincs*

**void SignalHandler(int sig):**

*Kezeli a különböző szignálokat, amelyeket a program futása közben kap. Az if utasítások segítségével ellenőrzi, hogy melyik szignált kapta meg az eljaras:*

- *Ha a kapott szignál SIGINT, azaz a felhasználó interrupt szignált küldött (a Ctrl+C billentyűkombinációval), akkor a program kiírja, hogy a folyamat leáll, majd sikeresen kilép (EXIT_SUCCESS) az az exit(0).*

```
if (sig == SIGINT)
{
        puts("The process is being stopped");
        exit(EXIT_SUCCESS);
}
```

- *Ha a kapott szignál SIGUSR1, ami a felhasználó által definiált szignál, akkor a program kiírja a hibát a standard hiba kimenetre, majd hibakóddal kilép (EXIT_FAILURE) az az exit(1).*

```
else if (sig == SIGUSR1)
{
        fprintf(stderr, "ERROR! (Send via file service is not available)\n");
```

```
        exit(EXIT_FAILURE);
    }
```

- *Ha a kapott szignál SIGALRM, azaz az alarm szignált kapja, akkor a program kiírja, hogy a szerver nem válaszol, majd hibakóddal kilép exit(4).*

```
    else if (sig == SIGALRM)
    {
        fprintf(stderr, "ERROR! (The server is not responding)\n");
        exit(4);
    }
```

*Paraméterek:*

- *int sig: A kapott szignál.*

*Visszatérési érték: Nincs.*

**int Commands(int* send_mode, int* file_mode, int argc, char *argv[]):**

*Ez a függvény parancssori argumentumok feldolgozását végzi el, hogy beállítsa a program módját és kommunikációs protokollját.*

*Paraméterek:*

- *int* send_mode: A küldési mód (send vagy receive) beállítását tároló pointer.*
- *int* file_mode: A kommunikációs mód (fájl vagy socket) beállítását tároló pointer.*
- *int argc: A parancssori argumentumok száma.*
- *char *argv[]: A parancssori argumentumok tömbje.*

*Visszatérési érték: Sikeres végrehajtás esetén EXIT_SUCCESS, exit(0), egyébként EXIT_FAILURE, exit(1).*

- *A függvény először megvizsgálja, hogy vannak-e parancssori argumentumok.*
- *Ezután az előre definiált parancssori argumentumokat összehasonlítja a kapott argumentumokkal, és beállítja a megfelelő módokat és protokollokat (send, receive, file, vagy socket).*
- *Hibás kombinációk vagy érvénytelen argumentumok esetén hibaüzenetet ír ki, és kilép hibakóddal.*

- *Ha a felhasználó --help vagy --version opciót használ, akkor meghivja a void print_version_info(); es a void print_help_info(); eljárásokat, amelyek kiírják a segítséget vagy a verzióinformációkat, majd sikeresen kilép belőlük.*

**int Measurement(int **Values):**

*Ez a függvény méréseket generál egy 3 állapotú 1 dimenziós random "lépkedést" implementál, ami véletlenszerű egész számokat hoz létre egy véges sorozatban. A szomszédos elemek közötti különbség abszolút értéke mindig 1.*

- *Az aktuális idő lekérdezése time() és localtime() függvényekkel.*
- *A mérési adatok számának kiszámítása az aktuális perc és másodperc alapján.*
  *int measurementCount = (currentTime -> tm_min * 60) + currentTime -> tm_sec;*
- *A mérések számát 100-ra korlátozza, ha az meghaladná ezt a számot.*
  // Korlátozás: maximum 100 meres
  if (measurementCount > 100)
  {
  measurementCount = 100;
  }
- *A rand() függvény inicializálása az aktuális idő alapján.*
- *Dinamikus memóriafoglalás a mérési adatok tárolására.*
  *Values = (int *)malloc(measurementCount * sizeof(int));
  (*Values)[0] = 0;
- *A mérési adatok generálása, ahol a számok véletlenszerűen növekedhetnek, csökkenhetnek vagy változatlanok maradhatnak az előző értékhez képest.*
  for(int i = 1; i < measurementCount; i++)
  {
  double rnd = (double)rand()/((unsigned)RAND_MAX+1);
  if(rnd <= 0.428571) // 42.8571%-os valószínüséggel növekszik
  {
  (*Values)[i] = (*Values)[i - 1] + 1;
  }
  else if (rnd <= 0.783409) // 78.3409%-os valószínüséggel csökken
  {
  (*Values)[i] = (*Values)[i - 1] - 1;
  }
  else
  {
  (*Values)[i] = (*Values)[i - 1];
  }

}

*Paraméterek:*

- *int **Values: Egy pointer a mérések tárolására szolgáló tömbre.*

*Visszatérési érték:*

- *A generált mérések számát adja vissza.*

**void BMPcreator(int *Values, int NumValues):**

*Létrehozza vagy megnyitja a chart.bmp nevű BMP fájlt írásra, majd a kapott adatok alapján létrehozza a megfelelő képet.*

- *A függvény létrehozza vagy megnyitja a chart.bmp fájlt a megfelelő paraméterekkel a fájl manipulációs függvények segítségével.*
  int fileHandle = open("chart.bmp", O_CREAT | O_WRONLY | O_TRUNC, S_IRUSR| S_IROTH | S_IWUSR);
- *A BMP fájl fejlécét előkészíti és kiírja a fájlba.*
- *A képméret kiszámítása a pixeladatok alapján, figyelembe véve a sorok hosszát is, amelyeket 32-bites határra kerekít.*
  unsigned int imageSize = (imageHeight * imageWidth/8)+BMP_HEADER_SIZE;
- *A kép pixeladatainak létrehozása és beállítása a kapott adatok alapján.*
- *A kép adatokat a fájlba írja, majd felszabadítja a dinamikusan foglalt memóriát és bezárja a fájlt.*

*Paraméterek:*

- *int *Values: Az adatokat tartalmazó tömb.*
- *int NumValues: A kapott adatok száma.*

*Visszatérési érték: Nincs.*

**int FindPID():**

*Megkeresi a "chart" nevű folyamat PID-jét a rendszeren.*

- *A /proc mappába való belépés.*
  d = opendir("/proc");
- *A függvény végigmegy az összes folyamaton a /proc mappában, és megvizsgálja a status fájlt, hogy ellenőrizze a folyamat nevét és PID-jét.*
- *Ha talál "chart" nevű folyamatot, és annak PID-je nem azonos a saját PID-vel, akkor elmenti ezt a PID-t.*
- *Végül visszaadja a talált PID-t, vagy -1-et, ha nem talált "chart" nevű folyamatot.*

*Paraméterek: Nincsenek.*

*Visszatérési érték:*

- *Az adott PID, ha találja a "chart" nevű folyamatot.*
- *Ha nem találja, akkor -1.*

**void SendViaFile(int *Values, int NumValues):**

*Adatokat küld a "Measurement.txt" nevű fájlba.*

- *Először a felhasználói könyvtár elérési útját kéri le a HOME környezeti változóból.*
- *Ezután összeállítja a teljes elérési utat a fájlhoz.*
- *Megnyitja a fájlt írásra, és beírja az adatokat a fájlba.*
- *Bezárja a fájlt és felszabadítja az elérési út memóriaterületét.*
- *Megkeresi a fogadó folyamat PID-jét a FindPID() függvénnyel, és kiírja a megtalált PID-t.*
- *Ellenőrzi, hogy sikerült-e megtalálni a fogadó folyamatot. Ha nem, akkor hibaüzenetet ír ki és kilép hibakóddal.*
- *Ha talált fogadó folyamatot, akkor jelzi, majd próbálkozik a fogadó folyamatnak küldeni egy jelzést (SIGUSR1), azonban megjegyzi, hogy ez a rész még nem működik helyesen.*

*Paraméterek:*

- *int *Values: A méréseket tartalmazó tömb.*
- *int NumValues: A méréseket tartalmazó tömb mérete.*

*Visszatérési érték: Nincs.*

**void ReceiveViaFile(int sig):**

*Ez az eljaras fogadja a „Measurement.txt-be érkező adatokat, majd feldolgozza és létrehozza belőle a megfelelő BMP képet.*

- *Kiírja a saját PID-jét, majd jelzi, hogy megkezdődött a fájl fogadása.*
- *Lekérdezi a felhasználói könyvtár elérési útját a HOME környezeti változóból.*
- *Összeállítja a teljes elérési utat a fogadott fájlhoz.*
- *Addig vár, amíg a fájl létezik (3 másodpercenként ellenőrizve).*

- *Megnyitja a fájlt olvasásra, majd soronként beolvassa és konvertálja az adatokat egész számokká.*
- *Létrehoz egy dinamikusan foglalt tömböt az adatok tárolására, majd beolvasza az összes sort és azokat eltárolja.*
- *Ezután meghívja a BMPcreator() függvényt a kapott adatokkal, hogy létrehozza a BMP képet.*
- *Törli a fogadott fájlt, felszabadítja a dinamikusan foglalt memóriát, majd kész jelzést küld a felhasználónak.*

*Paraméterek:*

- *int sig: A kapott jelzés, amely a fájl fogadását indítja.*

*Visszatérési érték: Nincs.*

**void SendViaSocket(int *Values, int NumValues):**

*Mérési adatokat küldi egy UDP socketen keresztül a megadott szervernek.*

- *Létrehoz egy UDP socketet (SOCK_DGRAM), és beállítja a szerver címét és portját.*
- *Elküldi az adatok méretét először a szervernek, majd vár egy válaszra a méret helyességének ellenőrzéséhez.*
- *Ha a szerver nem válaszol időben, akkor megszakítja az átvitelt egy időkorlát beállításával (SIGALRM).*
- *Ellenőrzi, hogy a kapott válasz mérete megegyezik-e az elküldött mérettel, és ha igen, akkor elküldi az adatokat.*
- *Vár egy válaszra a szerverről, amely tartalmazza az elküldött adatok méretét, majd ellenőrzi, hogy a fogadott és elküldött adatok mérete megegyezik-e.*
- *Ha minden rendben volt, kiírja a sikeres fájlküldés üzenetet.*

*Paraméterek:*

- *int *Values: A mérési adatokat tartalmazó tömb.*
- *int NumValues: A mérési adatok tömbjének mérete.*

*Visszatérési érték: Nincs.*

**void ReceiveViaSocket():**

*Ez az eljaras felelős a meresi adatok fogadásáért egy UDP socketen keresztül.*

- *Létrehoz egy UDP socketet (SOCK_DGRAM), és beállítja a szerver címét és portját.*
- *Várja a bejövő adatokat.*
- *Miután megkapta az adatok méretét, ellenőrzi és visszaigazolja az elküldött méretet.*
- *Megkapja az adatokat, majd visszaigazolja a fogadott adatok méretét.*
- *Létrehozza a BMP fájlt az adatok alapján.*
- *Ez a folyamat folyamatosan fut, míg a szerver aktív.*

*Paraméterek: Nincsenek.*

*Visszatérési érték: Nincs.*