Jegyzőkönyv

Operációs rendszerek BSC
2021 féléves beszámoló

Készítette:
Oláh Mózes
D9WPR7
mérnökinformatikus

Feladat:

17. Írjon egy olyan C programot, mely egy fájlból számpárokat kiolvasva meghatározza a legnagyobb közös osztóját. A feladat megoldása során használjon shared memory (osztott memória szegmens) IPC mechanizmust, valamint a kimenet kerüljön egy másik fájlba. A kimeneti fájl struktúrája kötött!
Példa a bemeneti és kimeneti fájl struktúrájára:
Bemeneti fájl: i (Ez jelzi a számpárok darabszámát) x y
Kimeneti fájl (Az x,y jelzi a bemeneti adatokat a z pedig a kimenet eredményét): x y z

Rövid leírás:

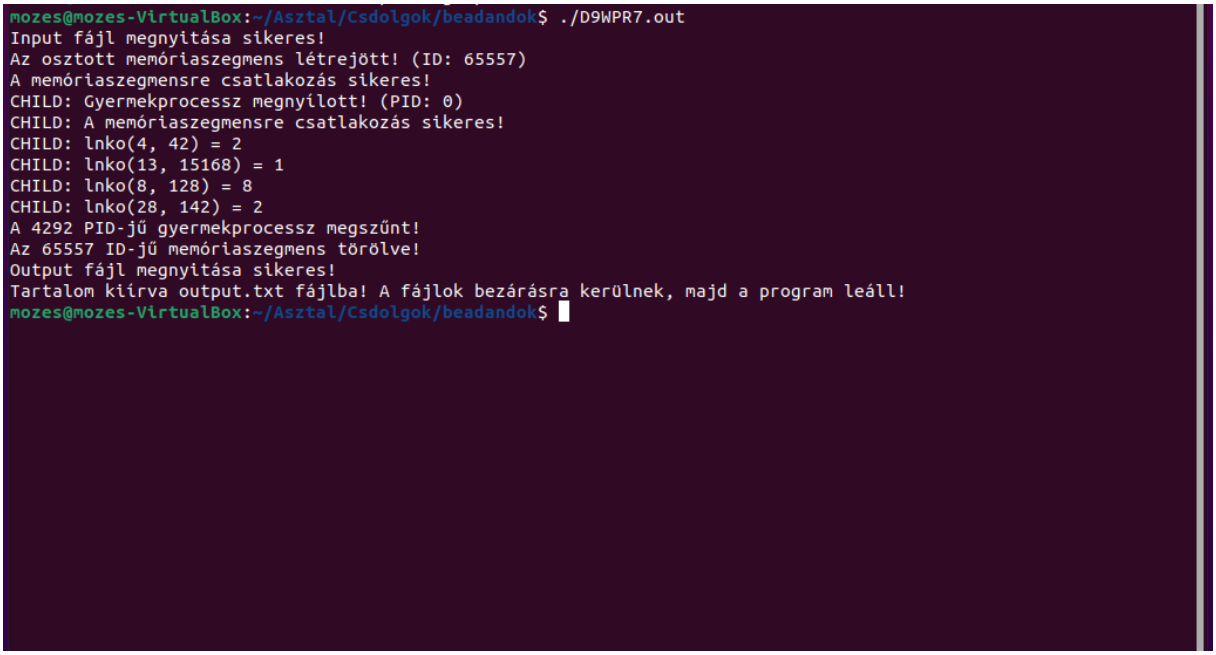
Egy olyan programot kellett írjak, mely egy fájlból beolvasva számpárokat meghatározza azok legnagyobb közös osztóját, illetve a megoldás során valahol osztott memóriaszegmenst használó IPC mechanizmust kellett alkalmazzak. Ezt én úgy oldottam meg, hogy egy ciklus soronként beolvassa a fájl tartalmát és azt egy kétdimenziós tömbben (gyakorlatilag stringek tömbjében) eltárolja. Egy másik ciklus, mint egy parserként ezeken a sorokon végigmegy és lebontja ezeket a sorokat a benne lévő szóközönként és a számokat (továbbra is string formában) eltárolja egymás után következve. Létrehoztam egy struktúrát ami megadja majd a memóriaszegmens struktúráját. A memóriaszegmens létrehozása után a program rácsatlakozik arra és létrehozza rajta a struktúrát, majd elteszi bele a fentebb említett tömb tartalmát, azonban itt már átalakítja azokat integerekké, így a memóriaszegmensen már integer tömbként létezik az input fájl tartalma. Ekkor létrehozódik egy gyermekprocessz, ami kiolvassa a memóriaszegmensből az adatokat, majd egy ciklus egy lnko algoritmust használó függvény
segítségével kiszámolja, majd visszaírja a memóriaszegmensen egy másik tömbbe az eredményeket. Ekkor a gyermek processz megszűnik (amire a szülő processz
várakozik), majd visszaíródnak a memóriaszegmensből egy tömbbe az adatok és a memóriaszegmens törlésre kerül. Ekkor a szülő processz létrehoz egy output fájlt, beleírja a megadott struktúra alapján az eredményeket és bezárja azt, majd felszabadítja a memóriát és leáll.

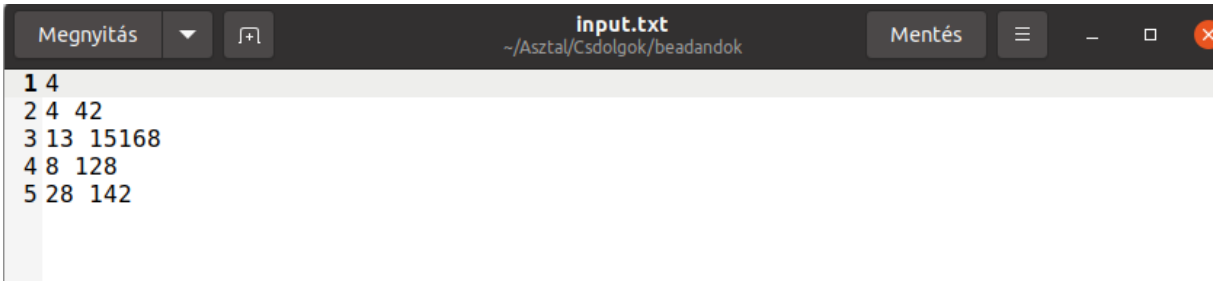
Csatolt fájlok:

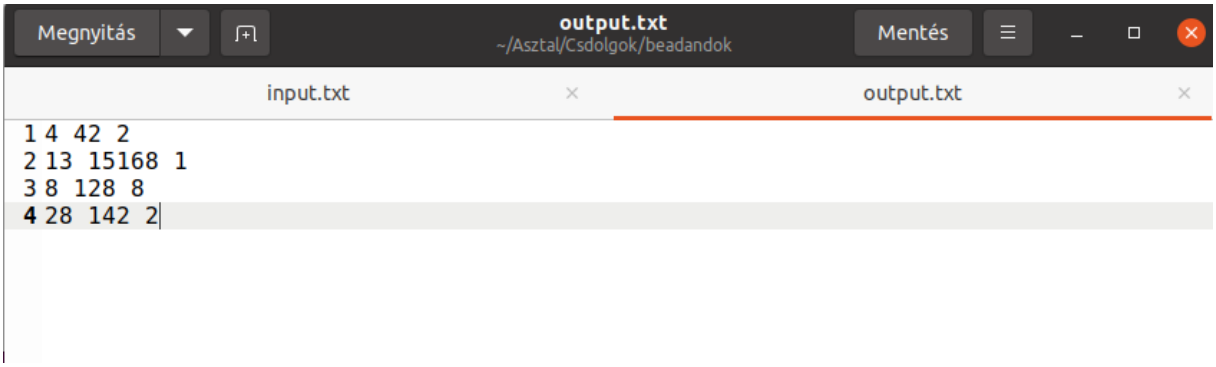
D9WPR7-17.c – program kódja
input.txt – egy tesztelés során használt input fájl

A program nem igényel a fordításnál semmilyen speciális paramétert, ellenben csak linux alatt képes futni.

Képek:


mozes@mozes-VirtualBox:~/Asztal/Csdolgok/beadandok$ ./D9WPR7.out
Input fájl megnyitása sikeres!
Az osztott memóriaszegmens létrejött! (ID: 65557)
A memóriaszegmensre csatlakozás sikeres!
CHILD: Gyermekprocessz megnyilott! (PID: 0)
CHILD: A memóriaszegmensre csatlakozás sikeres!
CHILD: lnko(4, 42) = 2
CHILD: lnko(13, 15168) = 1
CHILD: lnko(8, 128) = 8
CHILD: lnko(28, 142) = 2
A 4292 PID-jű gyermekprocessz megszűnt!
Az 65557 ID-jű memóriaszegmens törölve!
Output fájl megnyitása sikeres!
Tartalom kiírva output.txt fájlba! A fájlok bezárásra kerülnek, majd a program leáll!
mozes@mozes-VirtualBox:~/Asztal/Csdolgok/beadandok$


Megnyitás
input.txt
~/Asztal/Csdolgok/beadandok
Mentés
1 4
2 4 42
3 13 15168
4 8 128
5 28 142


Megnyitás
output.txt
~/Asztal/Csdolgok/beadandok
Mentés
input.txt
output.txt
1 4 42 2
2 13 15168 1
3 8 128 8
4 28 142 2

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#define SHMKEY 69905L //az osztott memóriaszegmes kulcsa

int lnko(int a, int b){  //euklideszi algoritmus a legnagyobb közös osztó meghatározására
    int c;
    while (b != 0){
        c = b;
        b = a%b;
        a = c;
    }
    return a;
}

int main() {

    FILE *input = fopen("input.txt", "r");
    char *line = malloc(1000*sizeof(char));
    char lines[15][20*sizeof(char)];
    char *token = malloc(1000*sizeof(char));
    char file_content[15][20*sizeof(char)];

    //ellenőrizzük a fájlmegnyitás sikerességét, sikertelen megnyitás esetén leállunk
    if(!input){
        printf("Input fájl megnyitása sikertelen!\n");
        return -1;
    }else{
        printf("Input fájl megnyitása sikeres!\n");
    }

    //beolvassuk a file tartalmát soronként egy lines nevű stringeket tartalmazó kétdimenziós tömbbe
    int line_counter = 0;
    while(fgets(line, 20*sizeof(char), input) != NULL){
        if (strlen(line) > 0 && line[strlen(line)-1] == '\n') line[strlen(line)-1] = '\0'; // a sor végén ottmaradt \n sortörést töröljük
        strcpy(lines[line_counter], line);
        line_counter++;
    }

    //a lines tömb stringjeit parseoljuk egy újabb stringeket tartalmazó kétdimenziós tömbbe, hogy a fájlból beolvasott számok egy tömb egyenkénti elemeiként legyenek tárolva
    int counter=0;
    for(int i=0; i<line_counter; i++){
        line=lines[i];
        token=strtok(line, " ");
        while(token != NULL){
            if (strlen(token) > 0 && line[strlen(token)-1] == '\n') token[strlen(token)-1] = '\0';
            strcpy(file_content[counter], token);
            counter++;
            token=strtok(NULL, " ");
        }
    }

    //az osztott memóriaszegmens struktúrája
    struct file_contents{
        int num;
        int nums[counter];
        int gcd[(int)(counter-1)/2];
    } *segm;

    int shmid;                                              // a memóriaszegmnes azonosítója
    key_t key = SHMKEY;                                 // kulcs a memóriához
```

```c
    //az osztott memóriaszegmens struktúrája
    struct file_contents{
        int num;
        int nums[counter];
        int gcd[(int)(counter-1)/2];
    } *segm;

    int shmid;                                              // a memóriaszegmnes azonosítója
    key_t key = SHMKEY;                                 // kulcs a memóriához
    int size = 512;                                         // a memóriaszegmens mérete byte-ban
    int shmflag;                                        // flag a jellemzőkhöz


    shmflag =0;
    if((shmid=shmget(key, size, shmflag)) < 0){
        shmflag = 00666 | IPC_CREAT;
        if((shmid=shmget(key, size, shmflag)) < 0){
            perror("Nem sikerült létrehozni az osztott memóriaszegmenst!\n");
            exit(-1);
        }else{
            printf("Az osztott memóriaszegmens létrejött! (ID: %d)\n", shmid);
        }
    }else{
        printf("A megadott KEY-jel már létezik osztott memóriaszegmens! %d\n", shmid);
        exit(-1);
    }

    shmflag =   00666 | SHM_RND;
    segm = (struct file_contents *)shmat(shmid, NULL, shmflag);
    if (segm == (void *)-1 ){
        perror("A memóriaszegmensre csatlakozás sikertelen!\n");
        exit(-1);
    }else{
        printf("A memóriaszegmensre csatlakozás sikeres!\n");
    }

    (segm->num) = file_content[0][0] - '0'; //char típus int típussá alakítva kerül a shm-be.
    for(int i=0;i<=counter-1; i++){
        (segm->nums[i]) = atoi(file_content[i+1]); //char típus int típussá alakítva kerül a shm-be.
    }

    //létrehozunk egy gyermekprocesszt, ami a szülő másolata lesz, de hogy ne csináljon végig minden ezutáni parancsot
    //a gyermekprocessz által végrehajtandó parancsokat egy feltételbe zárva gyakorlatilag a processzt is "bezárjuk"
    pid_t pid = fork();
    if ( pid == 0 ){
        printf("CHILD: Gyermekprocessz megnyílott! (PID: %d)\n", pid);

        shmflag =   00666 | SHM_RND;
        segm = (struct file_contents *)shmat(shmid, NULL, shmflag); //az előzőleg létrehozott struktúrát rákapcsoljuk az osztott memóriaszegmensre
        if (segm == (void *)-1 ){
            perror("CHILD: A memóriaszegmensre csatlakozás sikertelen!\n");
            exit(-1);
        }else{
            printf("CHILD: A memóriaszegmensre csatlakozás sikeres!\n");
        }

        //kiszámoljuk a lnko-kat, majd kiírjuk őket a memóriaszegmensre
        int p = 0;
        for (int i = 0; i < (segm->num); i++){
            printf("CHILD: lnko(%d, %d) = %d\n", segm->nums[p], segm->nums[p+1], lnko(segm->nums[p],segm->nums[p+1]));
            segm->gcd[i] = lnko(segm->nums[p],segm->nums[p+1]);
            p=p+2;
        }
        exit(0); //lelőjük a gyermekprocesszt
    }
    pid = wait(NULL); //a szülő bevárja a(z összes) gyermekprocesszt, annak megszűnéséig várakozik
```

Megnyitás | D9WPR7-17.c ~/Asztal/Csdolgok/beadandok | Mentés

```
94      (segm->num) = file_content[0][0] - '0'; //char típus int típussá alakítva kerül a shm-be.
95      for(int i=0;i<=counter-1; i++){
96          (segm->nums[i]) = atoi(file_content[i+1]); //char típus int típussá alakítva kerül a shm-be.
97      }
98
99      //létrehozunk egy gyermekprocesszt, ami a szülő másolata lesz, de hogy ne csináljon végig minden ezutáni parancsot
100     //a gyermekprocessz által végrehajtandó parancsokat egy feltételbe zárva gyakorlatilag a processzt is "bezárjuk"
101     pid_t pid = fork();
102     if ( pid == 0 ){
103         printf("CHILD: Gyermekprocessz megnyílott! (PID: %d)\n", pid);
104
105         shmflag =   00666 | SHM_RND;
106         segm = (struct file_contents *)shmat(shmid, NULL, shmflag); //az előzőleg létrehozott struktúrát rákapcsoljuk az osztott memóriaszegmensre
107         if (segm == (void *)-1 ){
108             perror("CHILD: A memóriaszegmensre csatlakozás sikertelen!\n");
109             exit(-1);
110         }else{
111             printf("CHILD: A memóriaszegmensre csatlakozás sikeres!\n");
112         }
113
114         //kiszámoljuk a lnko-kat, majd kiírjuk őket a memóriaszegmensre
115         int p = 0;
116         for (int i = 0; i < (segm->num); i++){
117             printf("CHILD: lnko(%d, %d) = %d\n", segm->nums[p], segm->nums[p+1], lnko(segm->nums[p],segm->nums[p+1]));
118             segm->gcd[i] = lnko(segm->nums[p],segm->nums[p+1]);
119             p=p+2;
120         }
121         exit(0); //lelőjük a gyermekprocesszt
122     }
123     pid = wait(NULL); //a szülő bevárja a(z összes) gyermekprocesszt, annak megszűnéséig várakozik
124     printf("A %d PID-jű gyermekprocessz megszűnt!\n", pid);
125
126     //visszamásoljuk az eredményeket egy tömbbe a memóriaszegmensről
127     int lnko[segm->num];
128     for(int i=0; i < (segm->num); i++){
129         lnko[i]=segm->gcd[i];
130     }
131     int amount = segm->num;
132
133     //töröljük a memóriaszegmenst
134     shmctl(shmid, IPC_RMID, NULL);
135     printf("Az %d ID-jű memóriaszegmens törölve!\n", shmid);
136
137     //megnyitjuk az output fájlt; irási jogosultsággal, ami annyit tesz, hogy ha még nem létezik, akkor létrehozódik, ha már létezik a tartalma felülíródik
138     FILE *output = fopen("output.txt", "w");
139
140     //ellenőrizzük sikerült-e megnyitni
141     if(!output){
142         printf("Output fájl megnyitása sikertelen!\n");
143         return -1;
144     }else{
145         printf("Output fájl megnyitása sikeres!\n");
146     }
147
148     //kiírjuk a fájlba az eredményeket soronként
149     int p=1;
150     for(int i = 0; i < amount; i++){
151             sprintf(line, "%s %s %d\n", file_content[p], file_content[p+1], lnko[i]);
152             fputs(line, output);
153             p=p+2;
154     }
155
156     printf("Tartalom kiírva output.txt fájlba! A fájlok bezárásra kerülnek, majd a program leáll!\n");
157
158     //bezárjuk a fájlokat és a dinamikusan foglalt memóriákat felszabadítjuk
159     free(token);
```