JEGYZŐKÖNYV

**Operációs rendszerek BSc**

**2022. tavasz**

**Féléves feladat**

**Készítette: Dávid Rebeka**

**Neptunkód: EQ4B3D**

1. **Feladat: IPC mechanizmus**

**A feladat leírása:**

Írjon egy olyan C programot, mely egy fájlból számpárokat kiolvasva meghatározza a legnagyobb közös osztóját. A feladat megoldása során használjon message queue (üzenetsoros) IPC mechanizmust, valamint a kimenet kerüljön egy másik fájlba.

A kimeneti fájl struktúrája kötött!

Bemeneti fájl:

i (Ez jelenti a számpárok darabszámát)

x y

Kimeneti fájl (x, y jelzi a bemeneti adatokat, z pedig a kimeneti eredményt):

x y z

**A feladat elkészítése:**

A feladat elkészítéséhez 2 txt kiterjesztésű fájlra, illetve egy C programkódra les szükségünk. A kódunk az egyik txt fájlból beolvassa a szóközzel elválasztott számokat, majd ezekből a számokból kiválasztva kettőt, megkeresi azoknak a legnagyobb közös osztóját, amit a második txt fájlba ír ki.

A gcd függvényben kiszámolom két szám legnagyobb közös osztóját, amit a későbbiekben felhasználok a fájlomba kiíratáshoz.

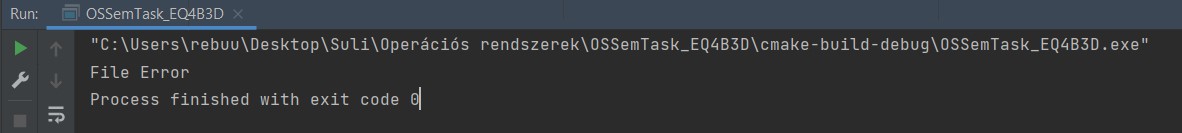
A main függvényben először meghívom a fájlt, amiből a kért számokat olvasom be, ha ez nem sikerül a program hibát ír ki. Ha minden rendben működik a main függvény felhasználja a korábbi gcd függvényt és kiszámolja a két beolvasott szám legnagyobb közös osztóját.

Ha sikerült kiszámolnia a két változót az output.txt fájlba kiírja az eredményt. Amennyiben a két számnak van közös osztója kiírja, amennyiben nincs visszatér 1-gyel.

Amennyiben nem sikerül beolvasni az output fájlt szintén hibaüzenettel tér vissza.

#include<stdio.h>  
#include<stdlib.h>  
  
int gcd(int a, int b) //algoritmus a legnagyobb közös osztó meghatározására  
{  
 if (a == 0)  
 return b;  
 return gcd(b % a, a);  
}  
int findGCD(int arr[], int n)  
{  
 int result = arr[0];  
 for (int i = 1; i < n; i++)  
 {  
 result = gcd(arr[i], result);  
  
 if(result == 1)  
 {  
 return 1;  
 }  
 }  
 return result;  
}  
int main()  
{  
 int numbers[50];  
 int i = 0;  
 FILE \*file;  
 if (file = fopen("input.txt", "r")) //beolvassuk a fájlt  
 {  
 while (fscanf(file, "%d ", &numbers[i]) != EOF)  
 {  
 i++;  
 }  
 fclose(file); //bezárjuk a fájl  
 }  
 else  
 {  
 printf("File Error"); // ha nem sikerül beolvasni a fájlt hibát ír kapunk  
 }  
 numbers[i] = '\0';  
  
 int n = sizeof(numbers) / sizeof(numbers[0]);  
 int result=findGCD(numbers,n);  
  
 FILE \*fp; //létrehozza a fáljt  
 if (fp = fopen("output.txt", "w")) //fájlba írja az eredményt  
 {  
 fprintf(fp, "GCD is :%d\n", result); // a megjelenítendő szöveg  
 fclose(fp); //bezárja a fájlt  
 }  
 else  
 {  
 printf("File Error"); // ha nem sikerül a fájlba írás hibaüzenetet kapunk  
 }  
 return 0;  
}

**A futtatás eredménye:**

****

1. **Feladat: OS algoritmusok**

**A feladat leírása:**

Adott egy igény szerinti lapozást használó számítógépesrendszer, melynek futás közben egy processz számára a következő laphivatkozással lehet hivatkozni: 6,8,3,8,6,0,3,6,3,5,3,6.

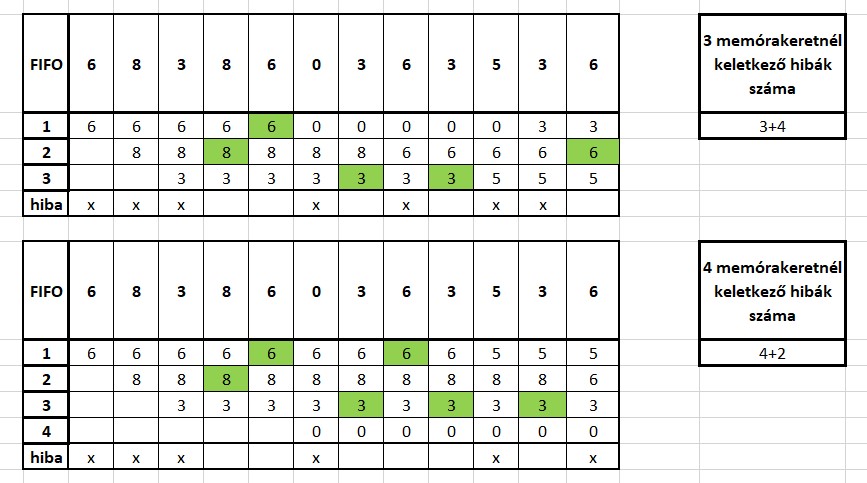
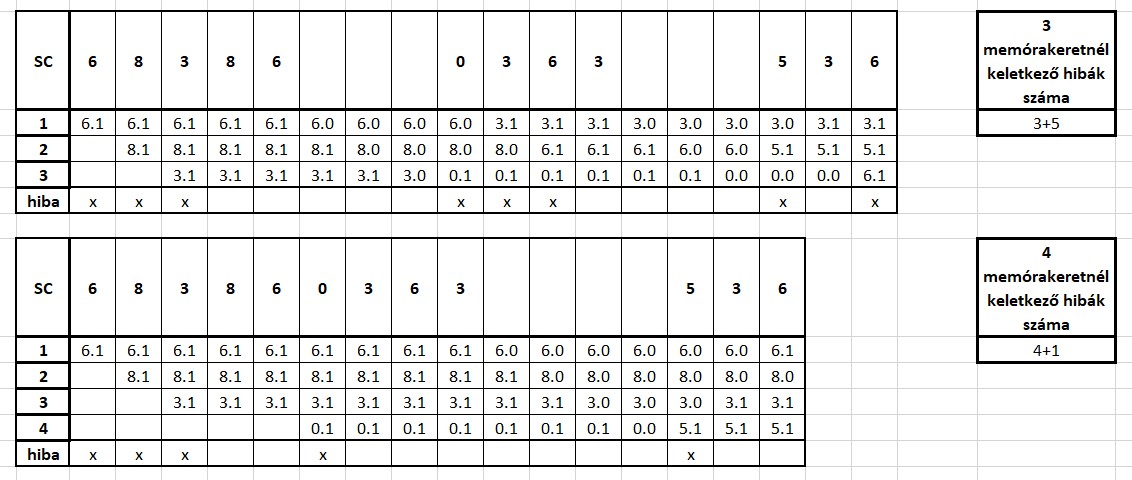
Memóriakeret (igényelt lapok): 3, ill. 4 memóriakeret.

Készítse el a laphivatkozások betöltését külön-külön táblázatba 3, ill. 4 memóriakeret esetén.

Mennyi laphiba keletkezik az alábbi algoritmusok esetén: FIFO, SC?

Hasonlítsa össze és magyarázza az eredményeket!

**A feladat elkészítése:**

* **FIFO:** Hibák szempontjából 4 memóriakeret esetén hatékonyabb
* **SC:** Hibák szempontjából 4 memóriakeret esetén hatékonyabb

**A FIFO és Second Chance algoritmust összehasonítva a legeredményesebb a Second Chance algoritmus 4 memóriakeret esetén, a legeredménytelenebb pedig a Second Chance 3 memóriakeret esetén.**