

Operációs rendszerek BSC

5. gyak.

2021. 03. 10.

Készítette:

Miliczki József Bsc

GÉIK - Programtervező Informatikus

Y86I0I

Miskolc, 2021

A feladatokban látható kódok megtalálhatóak külön file-ként is!

1.) Feladat: A system() rendszerhívással hajtson végre létező és nem létező parancsot, és vizsgálja a visszatérési értéket! Mentés: neptunkodgyak1.c

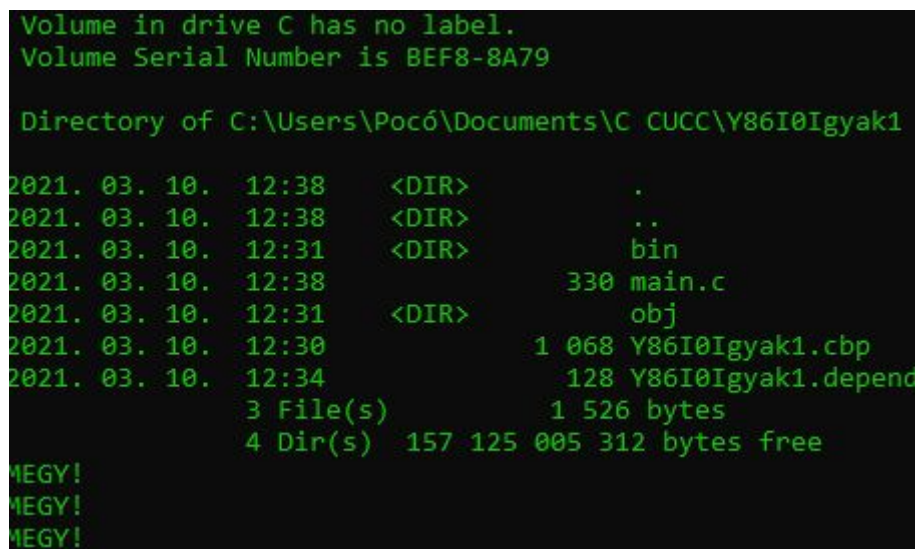
A Y86I0I.c tartalma:

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    if (!system("dir")){
        for (int i=0; i<3;i++){
            printf("MEGY!\n");
        }
        system("color a"); //Érdekesség, a szín megváltozott a dir command előtt!
        system("Ilyen nincs"); //Itt hibát jelez a rendszer, nem létező parancs!
        return 0;
    }
```

Az első paranccsal 2 legyet ütünk le egy csapással - kipróbáljuk milyen amikor működik egy parancs, valamint visszatérési értéket is vizsgálunk (Ha a system() 0-val tér vissza, akkor a megadott parancs lefut!)

Output:



```
Volume in drive C has no label.
Volume Serial Number is BEF8-8A79

Directory of C:\Users\Pocó\Documents\C CUCC\Y86I0Igyak1

2021. 03. 10. 12:38 <DIR>      .
2021. 03. 10. 12:38 <DIR>      ..
2021. 03. 10. 12:31 <DIR>      bin
2021. 03. 10. 12:38      330 main.c
2021. 03. 10. 12:31 <DIR>      obj
2021. 03. 10. 12:30      1 068 Y86I0Igyak1.cbp
2021. 03. 10. 12:34      128 Y86I0Igyak1.depend
                3 File(s)      1 526 bytes
                4 Dir(s)  157 125 005 312 bytes free

MEGY!
MEGY!
MEGY!
```

(A zöld színt hamarosan megmagyarázom!)

Ahogy láthatjuk lefutott a for ciklus tartalma, miután sikeresen végrehajtottuk a "dir" parancsot!

A `system("color a")` paranccsal kiadtam a konzolnak hogy zöld színű legyen, és ezzel az összes karakter (korábbi is) szín zölddé vált.

```
'Ilyen' is not recognized as an internal or external command,  
operable program or batch file.  
  
Process returned 0 (0x0)    execution time : 0.080 s  
Press any key to continue.
```

Az "Ilyen nincs" parancs nem létezik, ezért a shell nem tudja lefuttatni.

2.) Feladat: Írjon programot, amely billentyűzetről bekér Unix parancsokat és végrehajtja őket, majd kiírja a szabványos kimenetre. (pl.: amit bekér: date, pwd, who etc.; kilépés: CTRL-\\)

```
#include<stdio.h>  
#include<stdlib.h>  
#include<stdbool.h>  
int main(){  
  
    char Input[50];  
    do{  
        scanf("%s", Input);  
        system(Input);  
    }while(true);  
    return 0;  
}
```

Output:

```
mili@mili-VirtualBox:~/Desktop$ ./Y86I0I
dir
gst-release Y86I0I Y86I0I.c
pwd
/home/mili/Desktop
cd
ls
gst-release Y86I0I Y86I0I.c
ls
gst-release Y86I0I Y86I0I.c
^C
```

Kihasználjuk a linux beépített billentyűkombinációját amely egy futó programot leállít. Ez a CTRL+C. Addig olvasunk be parancsokat és futtatjuk őket a shell által, amíg a felhasználó ki nem adta a fenti kombinációt.

3.) Feladat: Készítsen egy parent.c és a child.c programokat. A parent.c elindít egy gyermek processzt, ami különbözik a szülőtől. A szülő megvárja a gyermek lefutását. A gyermek szöveget ír a szabványos kimenetre (5-ször) (pl. a hallgató neve és a neptunkód)!

parent.c

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/wait.h>

int main(){
pid_t pid = fork();
if (pid < 0){
    printf("Fork fail");
    exit(0);
}
else if(pid == 0){
    execl("./child", "child", (char *)NULL);
}
waitpid(pid, NULL, 0);
return 0;
}
```

child.c

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main(){

for (int i=0; i<5; i++){
printf("Miliczki Jozsef - Y86I0I\n");
sleep(1);
}
return 0;
}
```

Output:

```
mili@mili-VirtualBox:~/Desktop$ ./parent
Miliczki Jozsef - Y86I0I
Miliczki Jozsef - Y86I0I
Miliczki Jozsef - Y86I0I
Miliczki Jozsef - Y86I0I
Miliczki Jozsef - Y86I0I
```

A szülőben `fork()`-al kreálunk egy szülővel identikus processz-t, ő lesz a gyermek. A `pid_t` típusú `pid` változóból tudjuk megállapítani hogy milyen processzről van szó:

`pid = -1` Nem sikerült a gyermek létrehozása

`pid = 0` A gyermek működik

`pid > 0` (a szülő PID-je) A szülőnél van az irányítás

Ekkor még ugyan azt a feladatot látják el, viszont amint kiadom az `execl` parancsot, kijelölöm hogy melyik program foglalja el a gyermek helyét, és innentől kezdve mondhatjuk azt, hogy a C programunk egy másik C programot futtat. A `child.c` lefut, kiírja a megadott stringet és befejeződik a program utána.

4. Feladat: A `fork()` rendszerhívással hozzon létre egy gyerek processz-t és abban hívjon meg egy `exec` családbeli rendszerhívást (pl. `execlp`). A szülő várja meg a gyerek futását!

Ezt a 3-as feladatban már megvalósítottam, ott az `execl` parancsot alkalmaztam rendszerhívásra!

5. Feladat: A `fork()` rendszerhívással hozzon létre gyerekeket, várja meg és vizsgálja a befejeződési állapotokat (gyerekekben: `exit`, `abort`, nullával való osztás)!

A 3. Feladat kódját fogom átalakítani ennek a feladatnak a megvalósításához! A `parent` kód vége ezzel módosult:

```
int status;

waitpid(pid, &status, 0);

if (WIFEXITED(status))
{
    int exit_status = WEXITSTATUS(status);
    printf("A gyermek kilepesi erteke: %d\n", exit_status);
}
```

Konkrétan megvizsgáljuk a befejezési státuszt, és átadjuk az `exit_status` változóba. A `child` program először lefut simán:

```
mili@mili-VirtualBox:~/Desktop$ ./parent
Miliczki Jozsef - Y86I0I
Miliczki Jozsef - Y86I0I
Miliczki Jozsef - Y86I0I
Miliczki Jozsef - Y86I0I
Miliczki Jozsef - Y86I0I
A gyermek kilepesi erteke: 0
```

Most a `child`-ban kiadok egy `exit(10)` parancsot a ciklus elé!

```
mili@mili-VirtualBox:~/Desktop$ ./parent
A gyermek kilepesi erteke: 10
```

Nézzük meg mit kapunk ha 0-val osztunk!

Megjegyzés: Ezt nem tudjuk kipróbálni, mivel maga a fordító nem engedelmeskedik a 0-val való osztásnak. Szabvány szerint nincs ez az osztás definiálva, ezért hibába ütközünk.

Abort parancsra nem kapunk exit kódot, rögtön kidobja a processzet a CPU.

A tesztelésre módosított fájlok child5.c és parent5.c ként fognak szerepelni a mappában!

6.) Feladat: Adott a következő ütemezési feladat, ahol a RR ütemezési algoritmus használatával készítse el:

Határozza meg a

a.) Ütemezze az adott időszület alapján az egyes processzek paramétereit (ms)!

RR: 5 ms	Round Robin				
	P1	P2	P3	P4	P5
Érkezés	0	1,8	3	9,18	12
CPU idő	3	8,3	2	20,15	5
Indulás	0	3,10	8	13,23	18
Befejezés	3	8,13	10	18,38	23
Várakozás	0	2,2	5	4,5	6
Átlag					24/6 = 4 ms

b.) A rendszerben lévő processzek végrehajtásának sorrendjét?

p1 -> p2(1) -> p3 -> p2(bef.) -> p4(1) -> p5 -> p4(bef.)

c.) Ábrázolja Gantt diagram segítségével az aktív/várakozó processzek futásának menetét

