

Operációs rendszerek BSc

9. Gyak.

2022. 04. 05.

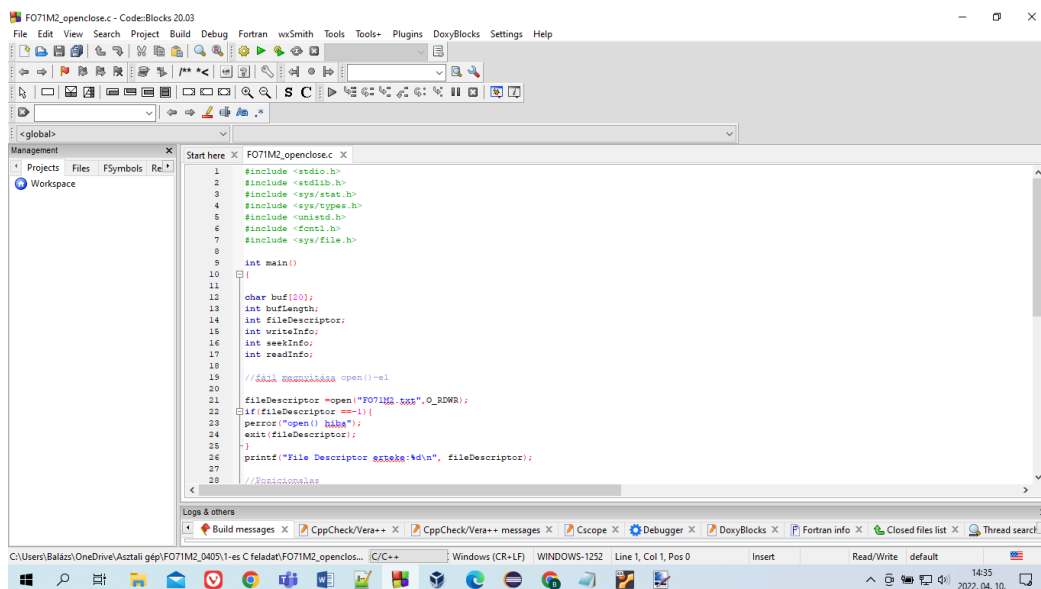
Készítette:

Petró Balázs Bsc
Mérnökinformatikus
FO71M2

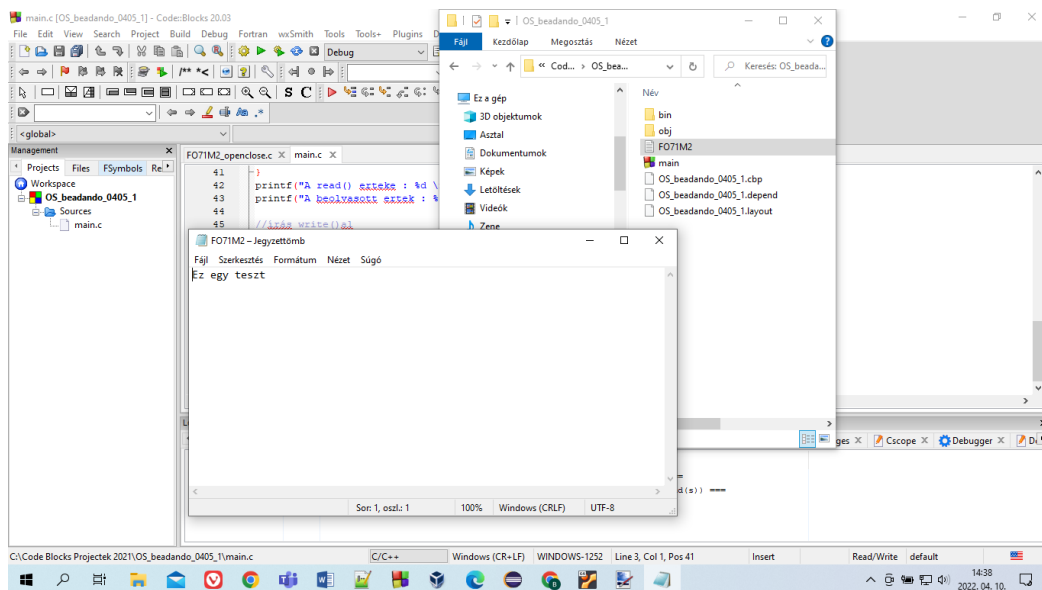
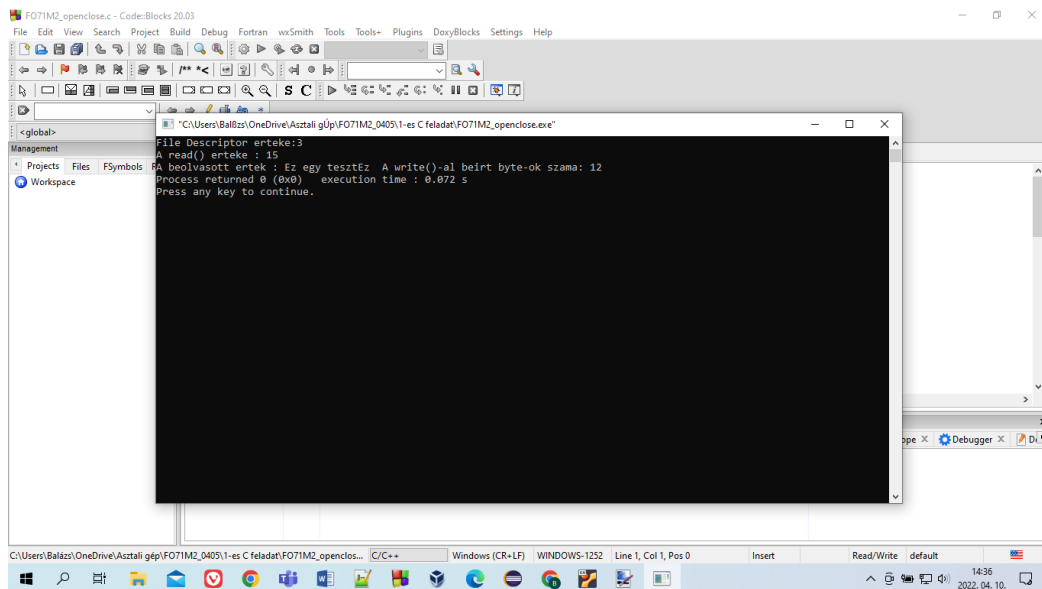
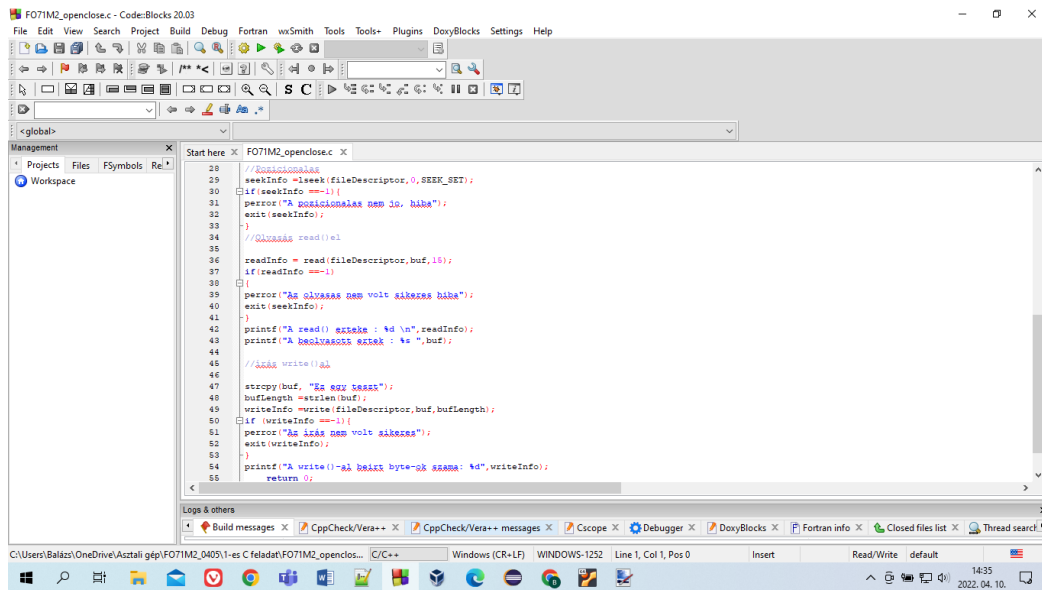
Miskolc, 2022

1. A tanult rendszerhívásokkal (open(), read()/write(), close()) - ők fogják a rendszerhívásokat tovább hívni - írjanak egy neptunkod_openclose.c programot, amely megnyit egy fájlt – neptunkod.txt, tartalma: hallgató neve, szak , neptunkod. A program következő műveleteket végezze:

- olvassa be a neptunkod.txt fájlt, melynek attribútuma: O_RDWR
- hiba ellenőrzést,
- write() - mennyit ír ki a konzolra.
- read() - kiolvassa a neptunkod.txt tartalmát és mennyit olvasott ki (byte), és kiírja konzolra.
- lseek() – pozícionálja a fájl kurzor helyét, ez legyen a fájl eleje: SEEK_SET, és kiírja a konzolra.

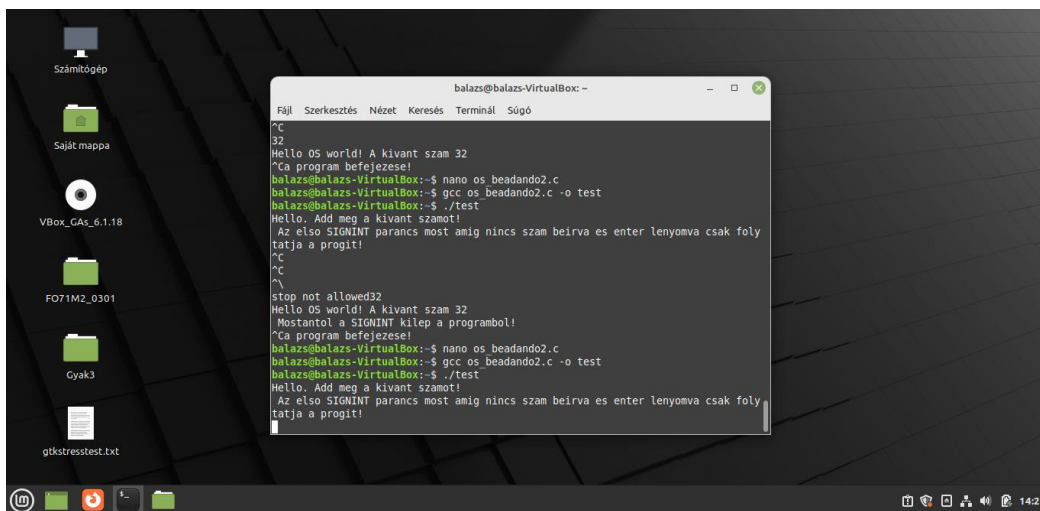


```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/stat.h>
4 #include <sys/types.h>
5 #include <unistd.h>
6 #include <fcntl.h>
7 #include <sys/file.h>
8
9 int main()
10 {
11
12     char buf[100];
13     int bufLength;
14     int fileDescriptor;
15     int writeInfo;
16     int seekInfo;
17     int readInfo;
18
19     //Ez a feladat megoldása
20
21     fileDescriptor = open("F071M2.txt", O_RDWR);
22     if (fileDescriptor == -1) {
23         perror("open() hiba");
24         exit(fileDescriptor);
25     }
26     printf("File Descriptor %d\n", fileDescriptor);
27
28     //Bontás
```



2. Készítse el a következő feladatot, melyben egy szignálkezelő több szignált is tud kezelni:

- a.) Készítsen egy szignál kezelőt (handleSignals), amely a SIGINT (CTRL + C) vagy SIGQUIT (CTRL + \) jelek fogására vagy kezelésére képes.
- b.) Ha a felhasználó SIGQUIT jelet generál (akár kill paranccsal, akár billentyűzetről a CTRL + \) a kezelő egyszerűen kiírja az üzenetet visszatérési értékét – a konzolra.
- c.) Ha a felhasználó először generálja a SIGINT jelet (akár kill paranccsal, akár billentyűzetről a CTRL + C), akkor a jelet úgy módosítja, hogy a következő alkalommal alapértelmezett műveletet hajtson végre (a SIG_DFL) – kiírás a konzolra.
- d.) Ha a felhasználó másodszor generálja a SIGINT jelet, akkor végrehajt egy alapértelmezett műveletet, amely a program befejezése - kiírás a konzolra. Mentés: neptunkod_tobbsignal.c



```
balazs@balazs-VirtualBox: ~  
^C  
32  
Hello OS world! A kívánt szám 32  
^Ca program befejezése!  
balazs@balazs-VirtualBox:~$ nano os_beadando2.c  
balazs@balazs-VirtualBox:~$ gcc os_beadando2.c -o test  
balazs@balazs-VirtualBox:~$ ./test  
Hello. Add meg a kívánt számot!  
Az első SIGINT parancs most amíg nincs szám beírva és enter lenyomva csak folytatja a progit!  
^C  
^C  
stop not allowed32  
Hello OS world! A kívánt szám 32  
Mostantol a SIGINT kilep a programbol!  
^Ca program befejezése!  
balazs@balazs-VirtualBox:~$ nano os_beadando2.c  
balazs@balazs-VirtualBox:~$ gcc os_beadando2.c -o test  
balazs@balazs-VirtualBox:~$ ./test  
Hello. Add meg a kívánt számot!  
Az első SIGINT parancs most amíg nincs szám beírva és enter lenyomva csak folytatja a progit!
```

A .c File futtatása után ezt a képet kapjuk . Innen lehet Ctrl+C illetve Ctrl+ \ parancsokat adni.


```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <signal.h>
4 #include <unistd.h>
5
6 void handleSignals (int sig){
7     if (sig == SIGQUIT)
8         printf("stop not allowed");
9
10    if (sig == SIGINT)
11    {
12        SIG_DFL;
13    }
14 }
15
16 void handleSignalsSIGINT_AFTER (int sig){
17     printf("a program befejezese\n");
18     exit(0);
19 }
20
21 int main(int argc ,char* argv[]){
22     struct sigaction sa;
23     int x;
24
25     sa.sa_handler = &handleSignals;
26     sa.sa_flags = SA_RESTART;
27     sigaction(SIGQUIT,&sa,NULL);
28     sigaction(SIGINT,&sa,NULL);
29     sa.sa_handler = &handleSignalsSIGINT_AFTER;
30
31     printf("Hello. Add meg a kivant szamot!\n Az elso SIGINT parancs most amig nincs szam beirva es enter lenyomva csak folytatja a progit!\n");
32     scanf("%d",&x);
33     printf("Hello OS world! A kivant szam %d\n Mostantol a SIGINT kilap a programbol!\n",x);
34     sigaction(SIGINT,&sa,NULL);
35     while(1)
36     {
37         sleep(1);
38         return 0;
39     }
}

```

A program kódja.

3. Adott a következő ütemezési feladat, amit a FCFS, SJF és Round Robin (RR: 4 ms) ütemezési algoritmus alapján határozza meg következő teljesítmény értékeket, metrikákat (külön-külön táblázatba):

	P1	P2	P3	P4
Érkezés	0	0	2	5
CPU idő	24	3	6	3
Indulás				
Befejezés				
Várakozás				

Külön táblázatba számolja a teljesítmény értékeket!

CPU kihasználtság: számolni kell a cs: 0,1(ms) és sch: 0,1 (ms) értékkel is.

Algoritmus neve	
CPU kihasználtság	
Körülfordulási idők átlaga	
Várakozási idők átlaga	
Válaszidők átlaga	

The screenshot displays an Excel spreadsheet with a Gantt chart and a summary table. The Gantt chart at the top shows tasks P1, P2, P3, and P4 with their durations and start/end times. The summary table below provides detailed scheduling metrics for each task.

SJF	Érkezés	CPU idő	Indulás	Befejezés	Várakozás	Körf idő	Sorrend
P1	0	24	0	24	0	24	P1,P2,P4,P3
P2	0	8	24	27	24	27	
P3	2	6	30	36	28	34	
P4	5	3	27	30	22	25	

Algoritmus neve	SJF
CPU kihasználtság	~98,901%
körülfordulási idő	110/4=27,5
átlag várakozási idő	74/4=18,5
Válaszidő átlaga	(0+24+30+27)/4=

Körülfordulási idő (ICPU idő + Várakozási/n CPU kihasználtság)	cs 0,1ms context switch	sch 0,1ms ütemezés	CPU kihasználtság: $\sum \text{rcPU} / \text{hasznos munka}$	*100%	(36,4-0,4)/36,4*100% = 36/36,4*100% = ~98,901%
			$\sum \text{rcPU}$		4 cs-je van

