

# **Operációs rendszerek BSc**

9.Gyak

2022.04.04.

**Készítette:**

Urbán Olivér BSc

Szak:

Mérnökinformatikus

Neptunkód: HEPMIU

**2022.04.04.**

## Feladatok

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:

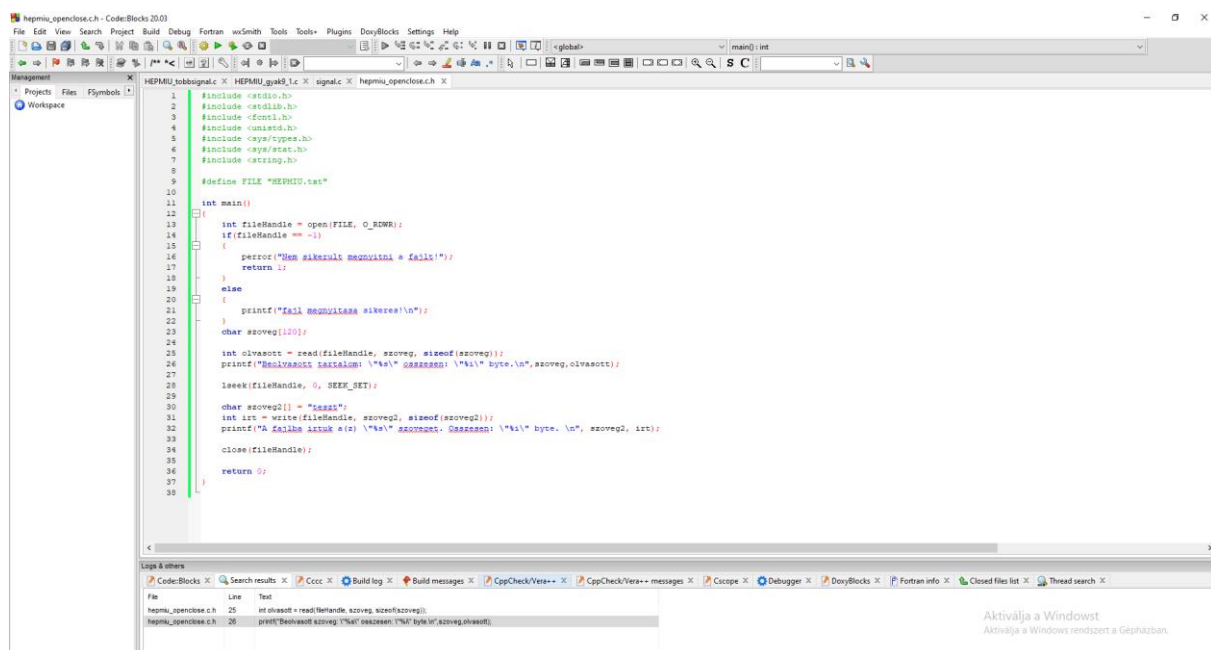
❑ olvassa be a neptunkod.txt fájlt, melynek attribútuma: O\_RDWR

❑ hiba ellenőrzést,

❑ write() - mennyit ír ki a konzolra.

❑ read() - kiolvasa 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 <fcntl.h>
4 #include <unistd.h>
5 #include <sys/types.h>
6 #include <sys/stat.h>
7 #include <string.h>
8
9 #define FILE "neptunkod.txt"
10
11 int main()
12 {
13     int fileHandle = open(FILE, O_RDWR);
14     if(fileHandle == -1)
15     {
16         perror("Nem sikerült megnyitni a fájlt");
17         return 1;
18     }
19     else
20     {
21         printf("Fájl megnyitása sikeres\n");
22     }
23     char szoveg[100];
24     int olvasott = read(fileHandle, szoveg, sizeof(szoveg));
25     printf("Beolvasott szöveg: \"%s\" összesen: \"%i\" byte.\n", szoveg, olvasott);
26     lseek(fileHandle, 0, SEEK_SET);
27     char szoveg2[] = "szöveg";
28     int irt = write(fileHandle, szoveg2, sizeof(szoveg2));
29     printf("A fájlba írtuk a(z) \"%s\" szöveget. Összesen: \"%i\" byte. \n", szoveg2, irt);
30     close(fileHandle);
31     return 0;
32 }
```

Logs & others

File	Line	Text
neptunkod_openclose.c	25	int olvasott = read(fileHandle, szoveg, sizeof(szoveg));
neptunkod_openclose.c	26	printf("Beolvasott szöveg: \"%s\" összesen: \"%i\" byte.\n", szoveg, olvasott);

Aktiválja a Windowst  
Aktiválja a Windows rendszert a Gépházban.

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

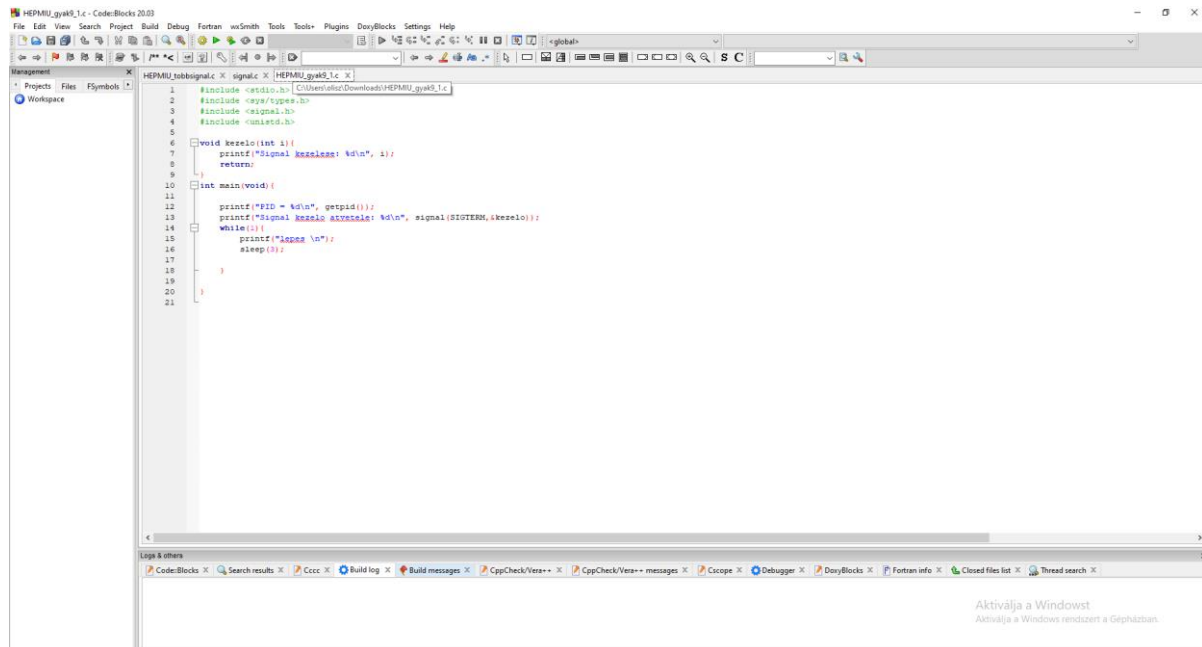
a.) Készítsen egy szignál kezelőt (handleSignals), amely a SIGINT (CTRL + C) vagy SIGQUIT (CTRL + \) jelek fogására vagy kezelésére képes.

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

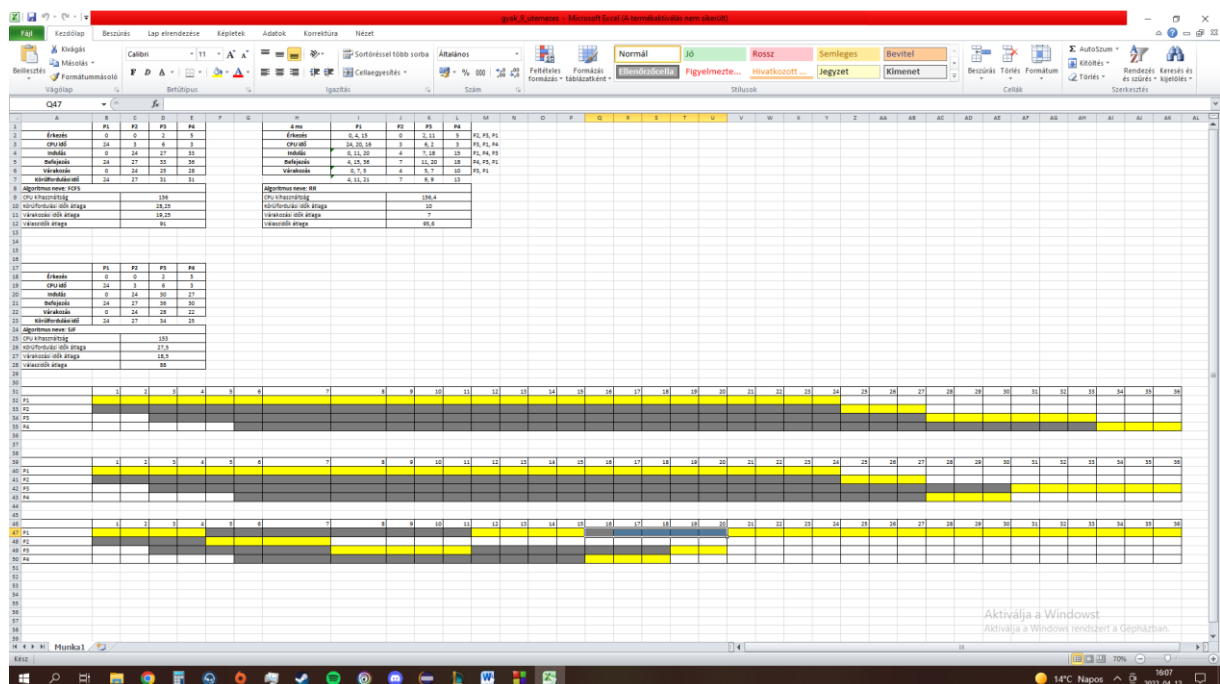
billentyűzetről a CTRL + C), akkor a jelet úgy módosítja, hogy a következő alkalommal

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



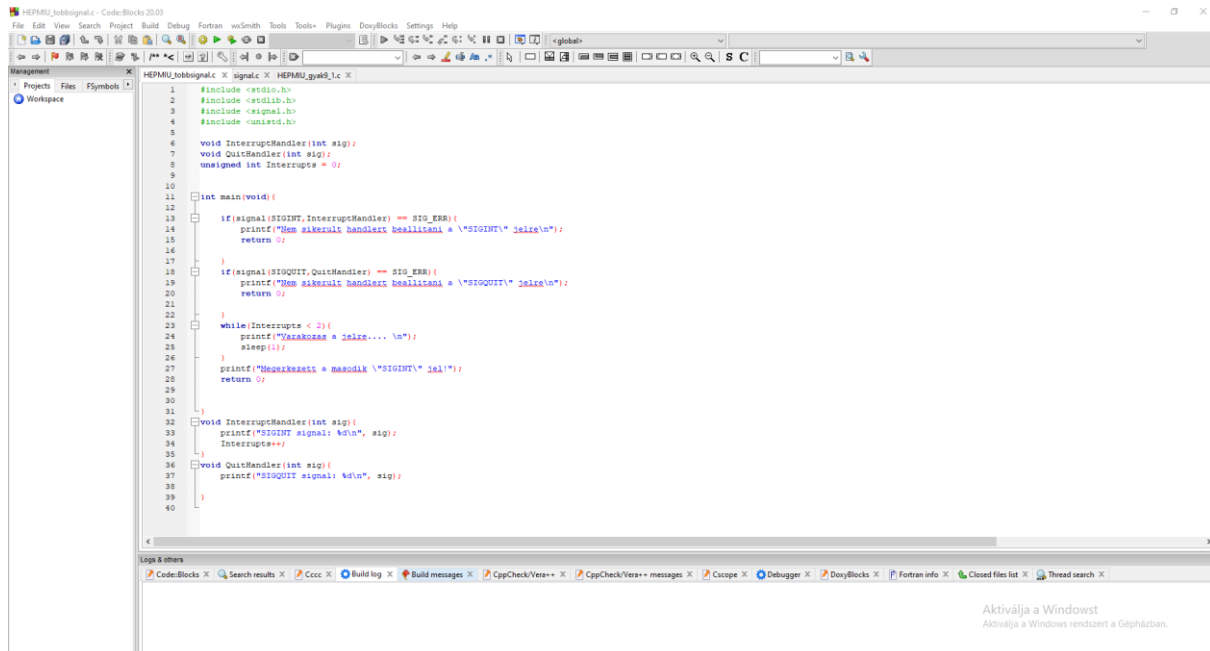
```
1 #include <stdio.h>
2 #include <ctype.h>
3 #include <signal.h>
4 #include <unistd.h>
5
6 void kezeles(int i)
7 {
8     printf("Signal kezeles: %d\n", i);
9     return;
10 }
11
12 int main(void)
13 {
14     print("PID = %d\n", getpid());
15     printf("Signal kezeles: %d\n", signal(SIGTERM, kezeles));
16     while(1)
17     {
18         printf("Aproha\n");
19         sleep(1);
20     }
21 }
```

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):



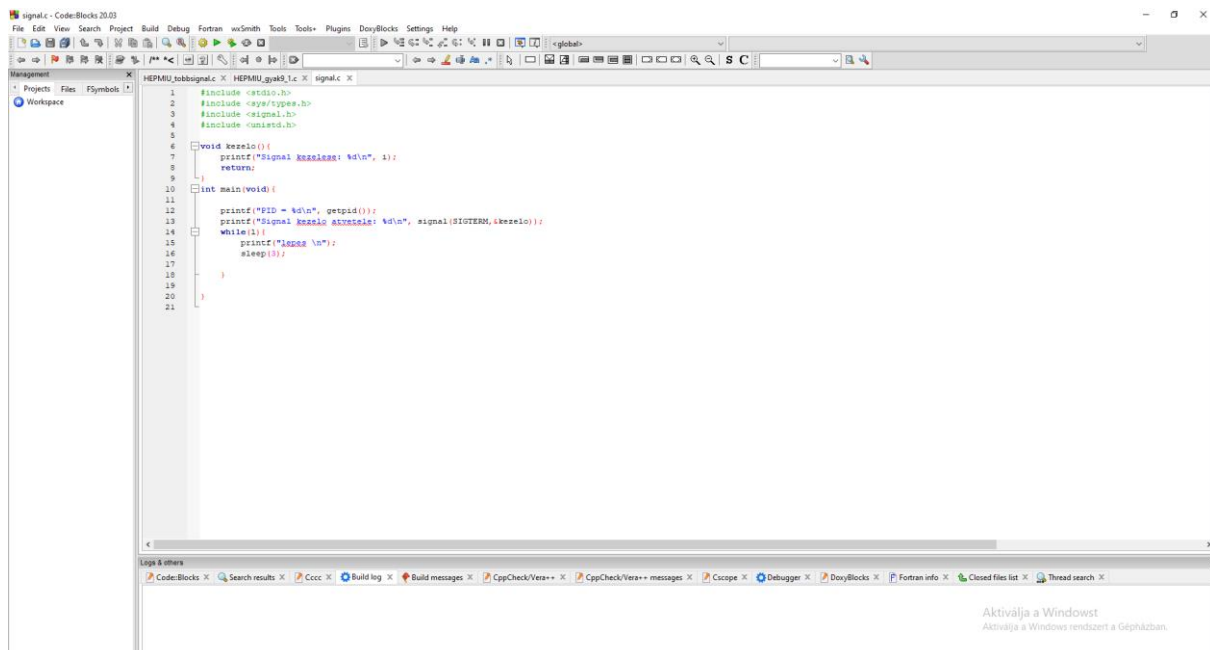
Idő	P1	P2	P3	P4
1	Ertes			
2	cpu id			
3	id			
4	id			
5	id			
6	id			
7	id			
8	id			
9	id			
10	id			
11	id			
12	id			
13	id			
14	id			
15	id			
16	id			
17	id			
18	id			
19	id			
20	id			
21	id			
22	id			
23	id			
24	id			
25	id			
26	id			
27	id			
28	id			
29	id			
30	id			

2. Írjon C nyelvű programot, amelyik kill() seg.-vel SIGALRM-et küld egy argumentumként megadott PID-u processznek, egy másik futó program a SIGALRM-hez rendeljen egy fv.-t amely kiírja pl. neptunkodot, továbbá pause() fv.-el blokkolódjon, majd kibillenés után jelezze, hogy kibillent és terminálódjon. Mentés. neptunkod\_gyak9\_1.c



```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <signal.h>
4 #include <unistd.h>
5
6 void InterruptHandler(int sig);
7 void QuitHandler(int sig);
8 unsigned int Interrupts = 0;
9
10
11 int main(void) {
12     if (signal(SIGINT, InterruptHandler) == SIG_ERR) {
13         printf("Hem sikeresen beallitani a SIGINT-t jelre\n");
14         return 0;
15     }
16     if (signal(SIGQUIT, QuitHandler) == SIG_ERR) {
17         printf("Hem sikeresen beallitani a SIGQUIT-t jelre\n");
18         return 0;
19     }
20     while (Interrupts < 2) {
21         printf("Várakozás a jelre... \n");
22         sleep(2);
23     }
24     printf("Megkezdett a második SIGINT jel!\n");
25     return 0;
26 }
27
28 void InterruptHandler(int sig) {
29     printf("SIGINT signal: %d\n", sig);
30     Interrupts++;
31 }
32
33 void QuitHandler(int sig) {
34     printf("SIGQUIT signal: %d\n", sig);
35 }
```

3. Írjon C nyelvű programot, amelyik a SIGTERM-hez hozzárendel egy fv.-t., amelyik kiírja az int paraméter értéket, majd végtelen ciklusban fusson, 3 sec-ig állandóan blokkolódva elindítás után egy másik shell-ben kill parancssal (SIGTERM) próbálja terminálni, majd SIGKILL-el.” Mentés. neptunkod\_gyak9\_2.c



```
1 #include <stdio.h>
2 #include <sys/types.h>
3 #include <signal.h>
4 #include <unistd.h>
5
6 void kezeles(i) {
7     printf("Signal kezelési: %d\n", i);
8     return;
9 }
10
11 int main(void) {
12     printf("PID = %d\n", getpid());
13     printf("Signal kezelési ATOMos: %d\n", signal(SIGTERM, kezeles));
14     while (1) {
15         printf("Várakozás \n");
16         sleep(3);
17     }
18     printf("Várakozás \n");
19     return 0;
20 }
```