

# Operációs rendszerek BSc

## 9. Gyak.

2022. 04. 04.

**Készítette:**

Juhász Balázs Bsc  
Mérnökinformatikus  
ZUYISF

**Miskolc, 2022**

### **1. feladat –**

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.

```
int main()
{
    int fileHandle = open(FILE, O_RDWR);
    if(fileHandle == -1)
    {
        perror("Nem sikerult megnyitni a fajlt!");
        return 1;
    }
    else
    {
        printf("Sikeres volt a fajl megnyitasa!\n");
    }
    char tartalom[120];
    int olvasott = read(fileHandle, tartalom, sizeof(tartalom));
    printf("Beolvasott tartalom: \"%s\" osszesen: \"%i\" byte.\n", tartalom, olvasott);

    lseek(fileHandle, 0, SEEK_SET);

    char text[] = "teszt";
    int irt = write(fileHandle, text, sizeof(text));
    printf("A fajlba irtuk a(z) \"%s\" szoveget. Osszesen: \"%i\" byte.\n", text, irt);

    close(fileHandle);

    return 0;
}
```

## 2. feladat –

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

- Készítsen egy szignál kezelőt (`handleSignals`), amely a `SIGINT` (`CTRL + C`) vagy `SIGQUIT` (`CTRL + \`) jelek fogására vagy kezelésére képes.
- Ha a felhasználó `SIGQUIT` jelet generál (akár `kill` paranccsal, akár billentyűzetről a `CTRL + \`) a kezelő egyszerűen kiírja az üzenetet visszatérési értékét – a konzolra.
- 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.



[illegible]

### Gyakorló feladat:

**2. feladat** – 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.

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

void handleSigalarm();

int main()
{
    printf("A program pidje: %d\n", getpid());
    signal(SIGALRM, handleSigalarm);
    pause();
    printf("Kibillent\n");
    exit(0);
    return 0;
}

void handleSigalarm()
```

**3. feladat** – 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.”

```
DLWGQZ_gyak9_2.c
1  #include <stdio.h>
2  #include <sys/types.h>
3  #include <signal.h>
4  #include <unistd.h>
5
6  void kezelo(int i)
7  {
8      printf("Signal kezelese: %d\n", i);
9      return;
10 }
11
12
13 int main()
14 {
15     printf("PID: %d\n", getpid());
16     printf("Signal kezelo atvetele: %d\n", signal(SIGTERM, &kezelo));
17     while(1)
18     {
19         printf("lepes\n");
20         sleep(3);
21     }
22 }
23
```