

# **Operációs rendszerek BSc**

**9. Gyak.**

**2022. 04. 04.**

**Készítette:**

Martinák Mátyás Bsc

Programtervező informatikus

KLNSPG

**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.

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

#define FILE "KLNSPG.txt"

int main() {
    int fileHandle = open(FILE, O_RDWR);
    if(fileHandle == -1)
    {
        perror("Nem sikerult megnyitni a fajlt!");
        return 1;
    } else
    {
        printf("Megnyitottam a fajlt!");
        char tartalom[64];

        int olvasott = read(fileHandle, tartalom, sizeof(tartalom));
        printf("beolvasott tartalom: \"%s\" osszesen \"%i\" byte.\n", tartalom, olvasott);

        lseek(fileHandle, 0, SEEK_SET);

        char szoveg[] = "teszt";
        int irt = write(fileHandle, szoveg, sizeof(szoveg));
        printf("A fajlba irtuk a(z) \"%s\" szoveget. osszesen \"%i\" byte.\n", szoveg, irt);
        close(fileHandle);
        return 0;
    }
}
```

2. feladat - 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.

```
#include <stdio.h>
#include <unistd.h>
#include <signal.h>

void handleSignals(int signum);

int main() {
    void(*sigHandlerInterrupt)(int);
    void(*sigHandlerQuit)(int);
    void(*sigHandlerReturn)(int);
    sigHandlerInterrupt = sigHandlerQuit = handleSignals;
    sigHandlerReturn = signal(SIGINT, sigHandlerInterrupt);

    if (sigHandlerReturn == SIG_ERR) {
        perror("Signal error");
        return 1;
    }

    sigHandlerReturn = signal(SIGQUIT, sigHandlerQuit);

    if (sigHandlerReturn == SIG_ERR) {
        perror("Signal error");
        return 1;
    }
    for(;;) {
        printf("\nA program leállításához a következőket vegezze el: \n");
        printf("1. Nyisson meg egy másik terminált.\n");
        printf("2. Adja ki a parancsot: kill: %d \n", getpid());
        sleep(10);
    }
    return 0;
}
```

```
void handleSignals(int signum) {  
    switch(signum) {  
        case SIGINT:  
            printf("\n CTRL+C-t észlelt\n");  
            signal(SIGINT, SIG_DFL);  
            break;  
        case SIGQUIT:  
            printf("SIQUIT aktivalodott\n");  
            break;  
        default:  
            printf("\nFogadott jel szama: %d\n", signum);  
            break;  
    }  
    return ;  
}
```

3. feladat - 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):

FCFS	P1	P2	P3	P4
Érkezés	0	0	2	5
CPU idő	24	3	6	3
Indulás	0	24	27	33
Befejezés	24	27	33	36
Várakozás	0	24	25	28
SJF	P1	P2	P3	P4
Érkezés	0	0	2	5
CPU idő	24	3	6	3
Indulás	24	0	3	9
Befejezés	36	3	9	12
Várakozás	24	0	1	4
RR: 4 ms	P1	P2	P3	P4
Érkezés	0, 4, 15, 24, 28, 32	0	2, 11	5
CPU idő	24, 20, 16, 12, 8, 4	3	6, 2	3
Indulás	0, 11, 20, 24, 28, 32	4	7, 18	15
Befejezés	4, 15, 24, 28, 32, 36	7	11, 20	18
Várakozás	0, 7, 5, 0, 0, 0	4	5, 7	10