Operációs rendszerek BSC

7. gyak.

2021. 03. 24.

Készítette:

Miliczki József Bsc

GÉIK - Programtervező Informatikus

Y86I0I

Miskolc, 2021

1. Feladat: RR Nélkül
   - A,B,C,D Processz ebben a sorrendben
   - p_uspri = 60
   - A,B,C p_nice = 0, D p_nice = 5
   - p_cpu = 0
   - 1-től 201-ig

p_cpu = p_cpu * KF, ahol KF értéke ½
p_pri = P_USER + p_cpu / 4 + 2 * p_nice
P_USER konstans, értéke 50
RR Nélkül: RR_nelkul.pdf
RR algoritmussal: RR.pdf

2. Feladat: A tanult rendszer hívásokkal (open(), read()/write(), close()) írjanak egy neptunkod_openclose.c programot, amely megnyit egy fájlt – neptunkod.txt, tartalma: hallgató neve, szak, neptun kód. A program következő műveleteket végezze el:
   - olvassa be a neptunkod.txt-t, attribútuma: O_RDWR
   - hiba ellenőrzés
   - write()
   - read()
   - lseek() - SEEK_SET-et a fájl elejére állítva

A következő sorokban a program működését és egyes komponenseit fogom elemezni. A magyarázatokhoz képeket is csatolok amelyek a kód megvalósítását mutatják. A program teljes forráskódja az Y86I0I_openclose.c állományban található.

Az fcntl.h könyvtár

Mivel file-al dolgozunk, ezért meg szeretnénk nyitni azt. Erre az open() parancsot fogjuk használni, amelyhez az **fcntl.h** könyvtárat kell beágyazni a programunk Header állományába. Az **fcntl.h** könyvtárból nyerjük ki a feladatban leírt O_RDWR zászlót amely olvasásra és írásra nyit meg egy fájlt, valamint magát az open() parancsot definiálja. Programunk lefut a könyvtár nélkül is, ám jelzi hogy implicit deklarációt hajtunk végre, így “illik” elhelyezni a könyvtárat mint forrást a funkcióhoz.

```
#include <fcntl.h>
```

open() és a fileDescriptor változó

Az open() parancs 2 argumentumot fogad: A file elérési útvonalát és a file kezelési módszerét (olvasás, írás, “toldás” {append}, írás/olvasás stb.). Jelenleg a kezelési módszert az O_RDWR zászló adja meg, amely az olvasás és írást jelenti.
Az open() parancs visszatérési értéke (helyes működés esetén) egy nemnegatív egész, amely a legkisebb elérhető azonosítót adja vissza az argumentumban megadott file-hoz, amit a process elér. Hiba esetén a visszatérési érték -1. A feladatban a visszatérési értéket a fileDescriptor változóban tároljuk el, amely egész típusú.

```
int fileDescriptor;
fileDescriptor = open("Y86I0I.txt",O_RDWR);
```

Hibakezelés

Az open() funkció visszatérési értékét (innentől kezdve fileDescriptor) megvizsgáljuk a fenti hiba kimenetelre, a -1-re. Ha -1-et kapunk, jelezzük a felhasználónak hogy a fájl megnyitása sikertelen, és egy exit(-1) paranccsal terminálják a processzt. Ha a fileDescriptor értéke nem -1 - tehát megkapjuk a file processz alatti azonosítóját - haladunk tovább.

```
//Fájl megnyitása open()-el
fileDescriptor = open("Y86I0I.txt",O_RDWR);
if (fileDescriptor == -1){ //Hiba ellenőrzés
    perror("open() hiba:");
    exit(fileDescriptor);
}
printf("File Descriptor erteke: %d\n", fileDescriptor);
```

Fájl kurzor pozícionálás lseek() metódussal

Az lseek() a **unistd.h** könyvtárban található, ezért ezt a header állományt hozzá kell fűzni a programhoz:

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

Az lseek metódus 3 argumentumot fogad:

- Egy egész szám, amely a fileDescriptor tartalma, természetesen azért, hogy referáljunk a fájlra amin dolgozunk.
- Egy egész szám, amellyel megadjuk, hogy melyik byte-ra irányítsa a kurzort az lseek(). Ez **offset**-nek hívjuk innentől.
- Egy egész szám, amely az lseek() működését definiálja. Ez 3 konstansban ürül ki:
    - SEEK_SET: A kurzort **offset** byte-ra állítjuk.
    - SEEK_CUR: A kurzort a jelenlegi pozícióhoz képest **offset** byte-al eltoljuk.
    - SEEK_END: A kurzort a fájl végéhez teszi, majd eltoljuk még **offset** byte-al.

Az lseek() visszatérési értéke egy egész: -1 ha nem sikerült a pozícionálás, egyébként visszaadja a kurzor helyét a fileDescriptor által definiált fájlban.

Tesztelés és hibakezelés érdekében létrehozok egy seekInfo egész változót, amelyben eltárolom majd az lseek() visszatérési értékét.

```
//Pozícionálás
seekInfo = lseek(fileDescriptor, 0, SEEK_SET);
if (seekInfo == -1){
    perror("A pozicionalas nem volt sikeres:");
    exit(seekInfo);
}
printf("A kurzor pozicioja: %d\n", seekInfo);
```

Fájlból olvasás read() paranccsal

A read paranccsal byte szinten tudunk fájlokból olvasni. 3 argumentumot fogad el:

- Egy egész változót, amely mutat a fájlra, jelen esetben fileDescriptor.
- Egy “buffer változó”, amelybe beolvassuk a fájlban található adatokat.
- Egy méret változó, amellyel megadjuk a beolvasni kívánt byte-ok számát.

Visszatérési értéke -1 hiba esetén, helyes működéskor visszaadja a sikeresen beolvasott byte-ok számát. Ez a szám lehet kisebb mint a kért byte-ok száma, mivel lehet csak 10 byte-nyi adat érhető el a fájlban, a kért 50 helyett. Ez természetesen nem hiba. A teszteléshez és a hibakezeléshez most is létrehoztam egy változót, ennek neve readInfo, valamint egy buffer változót, amely egy karakter típusú “buf” változó lesz. A megvalósítás a következő oldalon található.

```
//Olvasás read()-el
readInfo = read(fileDescriptor, buf, 15);
if (readInfo == -1){
    perror("Az olvasás nem volt sikeres:");
    exit(seekInfo);
}
printf("A read() erteke: %d\n", readInfo);
printf("A beolvasott ertek: %s", buf);
```

Fájlba írás write() paranccsal

A write() metódus hasonlóan működik a read() metódushoz, ugyanúgy 3 argumentummal rendelkezik:

- Egy egész, amely mutat a fájlra amelybe írni szeretnénk. Ez a fileDescriptor-ban van benne.
- Egy buffer változó (karakter típus most), amely tartalmát be szeretnénk írni a fájlba
- Egy méretet definiáló egész, amely megmondja hány byte-ot szeretnénk írni a fájlba a buffer változóból.

A visszatérési érték megadja az írás kimenetelét: Vagy a sikeresen beírt byte-ok mennyiségét adja meg, vagy egy -1-el jelzi a hibát.

```
//Írás write()-al
strcpy(buf, "Ez egy teszt");
bufLength = strlen(buf);
writeInfo = write(fileDescriptor, buf, bufLength);
if (writeInfo == -1){ //Hiba ellenőrzés
    perror("Az iras nem volt sikeres:");
    exit(writeInfo);
}
printf("A write()-al beirt byte-ok szama: %d", writeInfo);
```