

JEGYZŐKÖNYV

Operációs rendszerek BSc

2022. tavasz féléves feladat

Készítette: **Szendrei Gábor**

Neptunkód: V9ZK10

A feladat leírása:

1. IPC mechanizmus

Írjon egy olyan C programot, mely egy fájlból számpárokat kiolvassa meghatározza a legnagyobb közös osztóját. A feladat megoldása során használjon nevesített csővezetékot, valamint a kimenet kerüljön egy másik fájlba. A kimeneti fájl struktúrája kötött!

Példa a bemeneti és kimeneti fájl struktúrájára:

Bemeneti fájl:

i (Ez jelzi a számpárok darabszámár)

x y

Kimeneti fájl:

x y z (Az x,y jelzi a bemeneti adatokat a z pedig a kimenet eredményét)

2. OS algoritmus

Adott négy processz(A,B,C,D) a rendszerbe, induláskor a p_cpu értéke A=0, B=0, C=0, D=0. A rendszerben a P_USER = 60 Az óráütés 1indul, a befejezés 301-ig.

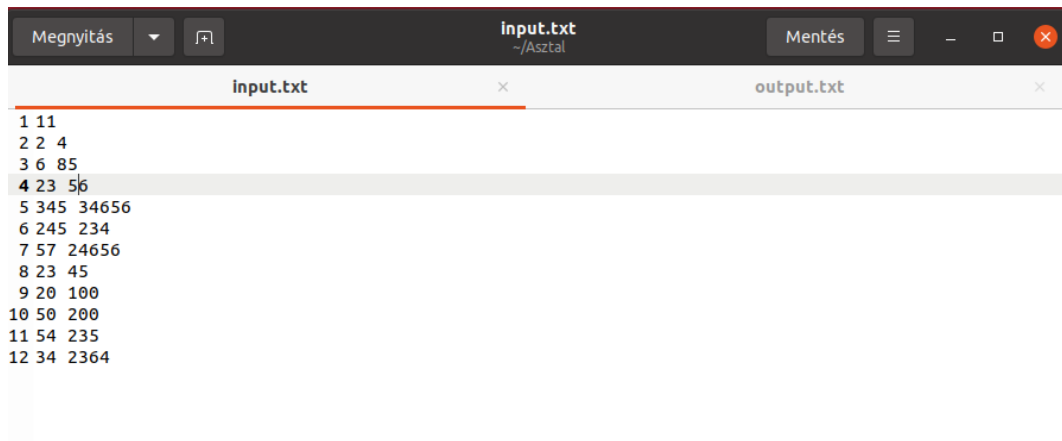
Induláskor a p_usrpri A=60, B=60, C =65 és D=60. Induláskor a p_nice értéke A=0, B=0, C=5 és D=0.

- Határozza meg az ütemezést RR nélkül 301 óráütésig – táblázatba!
- Minden óráütem esetén határozza meg a processzek sorrendjét óráütés előtt/után.
- Igazolja a számítással a tanultak alapján.

A feladat elkészítésének lépései:

1. IPC mechanizmus

Először létrehoztam a feladat főbb részeit. Gondolok ezalatt az LNKO módszerre, a fájl olvasás és írás részegységeit. Ezek után végrehajtottam pár próbát az input file segítségével. Miután ez működött következett a csővezeték „beépítése” a kódba. Végrehajtottam pár tesztet, mindegyik sikeres lett. Ezután kommentárt írtam a kódba, hogy segítse az átláthatóságot, majd mentettem és feltöltöttem GitHubra.



Először is definiáltam a fifo-t

```
#define FIFO_NAME "Fifo"
if(mkfifo(FIFO_NAME, 0666) < 0){
    perror("mkfifo");
}
```

Aztán fork rendszerhívás segítségével 2 szekcióra bontottam a programot. Először is `close(fd[0])` segítségével a csővezeték végét lezártam, hogy csak beleírni lehessen. Aztán az adatokat betöltöm `write` parancs segítségével a csővezetékbe.

```
int id = fork();

close(fd[0]);
write(fd[1], &x, sizeof(int));
```

Miután ezek megtörténtek, a fork 2. szekciójára került a futás, itt szintén `close(fd[1])` parancsal bezártam, csak a csővezeték elejét. Így a kiolvasásra használt `read` parancs megtudta oldani a feladatát.

```
close(fd[1]);
read(fd[0], &y, sizeof(int));
```

Az lnko számítást az alábbi metódussal oldottam meg.

```
int lnko(int n1, int n2)
{
    int gcd = -1;
    int i;
    for(i=1; i <= n1 && i <= n2; ++i)
    {
        if(n1%i==0 && n2%i==0)
            gcd = i;
    }
    return gcd;
}
```

2. OS algoritmusok

Először is kutatómunkát végeztem, mivel az ismereteim hiányosak voltak. A BME egyik weboldalán találtam hasznos információkat (https://www.mit.bme.hu/eng/system/files/oktatas/targyak/8776/unix_3_process_scheduling.pdf). Miután ezzel a tudással felszerelkeztem, elkezdtem kitölteni a táblázatot a megfelelő képletek alapján.

Ez azt jelenti hogy minden 100.óraütésnél p_cpu-t illetve p_pri-t számoltam az éppen futó processznek.

The screenshot shows an Excel spreadsheet with a table of process scheduling data and a text box explaining the priority calculation algorithm.

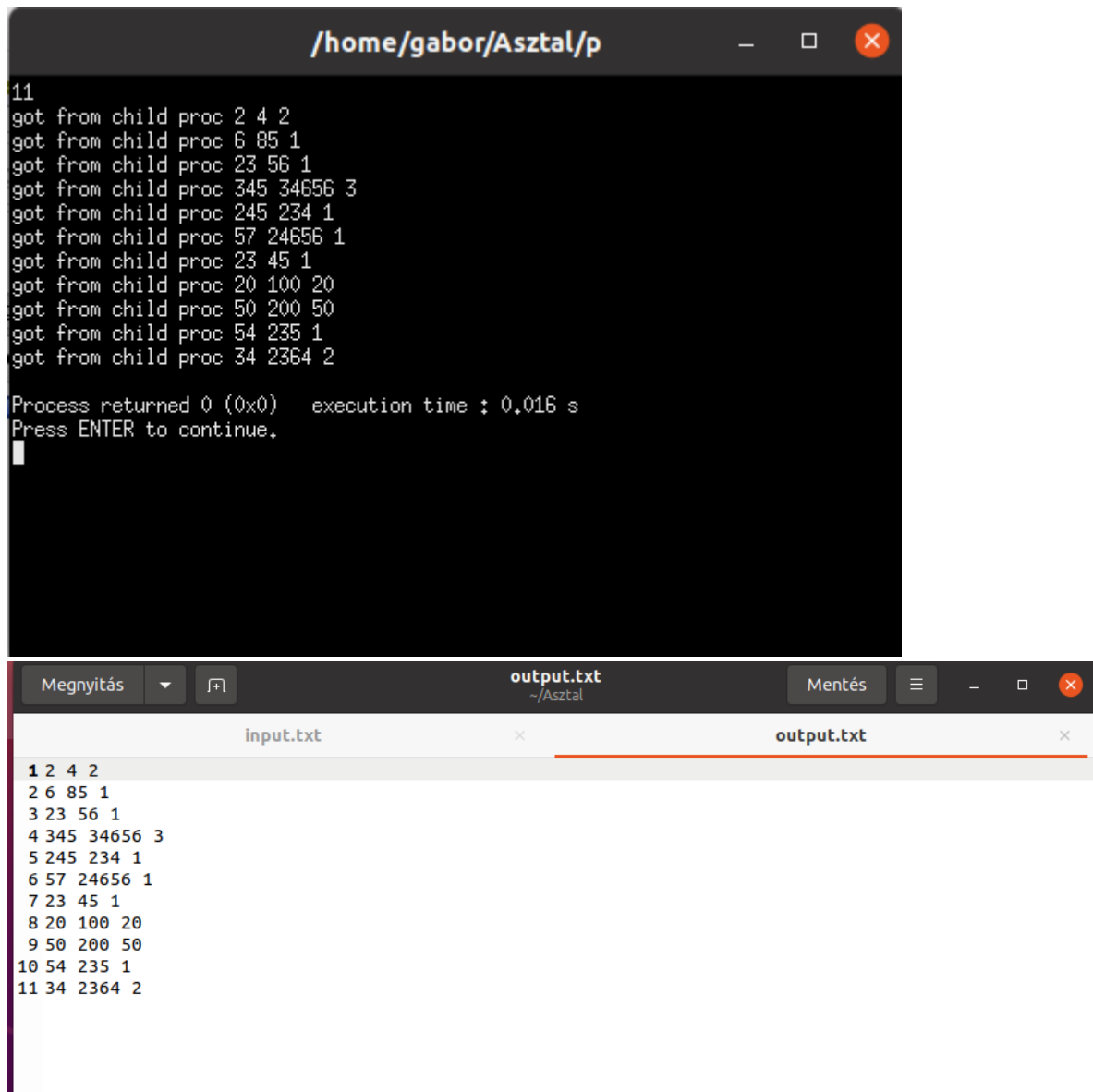
Óraütés	A p_pri	A p_cpu	B p_pri	B p_cpu	C p_pri	C p_cpu	D p_pri	D p_cpu	Utolsó proc.	Köv. proc.
0	60	0	60	0	65	0	60	0	-	A
1	60	1	60	0	65	0	60	0	A	A
2	60	2	60	0	65	0	60	0	A	A
3	60	3	60	0	65	0	60	0	A	A
4	60	4	60	0	65	0	60	0	A	A
5	60	5	60	0	65	0	60	0	A	A
6	60	6	60	0	65	0	60	0	A	A
7	60	7	60	0	65	0	60	0	A	A
8	60	8	60	0	65	0	60	0	A	A
9	60	9	60	0	65	0	60	0	A	A
10	60	10	60	0	65	0	60	0	A	A
11	60	11	60	0	65	0	60	0	A	A
12	60	12	60	0	65	0	60	0	A	A
13	60	13	60	0	65	0	60	0	A	A
14	60	14	60	0	65	0	60	0	A	A
15	60	15	60	0	65	0	60	0	A	A
16	60	16	60	0	65	0	60	0	A	A
17	60	17	60	0	65	0	60	0	A	A
18	60	18	60	0	65	0	60	0	A	A
19	60	19	60	0	65	0	60	0	A	A
20	60	20	60	0	65	0	60	0	A	A
21	60	21	60	0	65	0	60	0	A	A
22	60	22	60	0	65	0	60	0	A	A
23	60	23	60	0	65	0	60	0	A	A
24	60	24	60	0	65	0	60	0	A	A
25	60	25	60	0	65	0	60	0	A	A
26	60	26	60	0	65	0	60	0	A	A
27	60	27	60	0	65	0	60	0	A	A
28	60	28	60	0	65	0	60	0	A	A

Calculating the priority in user mode

- In every cycle p_cpu is increased for the running process
 p_cpu++
- After 100 cycles p_pri is calculated in the following way
 - p_cpu is „aged” by a correction factor (CF)
 $p_cpu = p_cpu * CF$
 - then the new priority is calculated according to this equation:
 $p_pri = P_USER + p_cpu / 4 + 2 * p_nice$ $P_USER = 50$ (constant)
- Calculating the correction factor:
 - SVR3: $CF = 1/2$ (What is the problem with this?)
 - 4.3 BSD: CF depends on the average number of processes (load_avg) in runnable (ready to run) state:
 $CF = 2 * load_avg / (2 * load_avg + 1)$
 - See the following commands: `w`, `top` and the graphical system monitor

p_user = 60
kf=3/4

A futtatás eredménye:



The image shows two windows from a Linux desktop environment. The top window is a terminal titled `/home/gabor/Asztal/p` with a dark background. It displays the output of a program, showing 11 lines of data received from child processes. The bottom window is a file editor titled `output.txt` with a light background, showing the same 11 lines of data saved in a file. The file editor has tabs for `input.txt` and `output.txt`, with `output.txt` being the active tab.

```
11
got from child proc 2 4 2
got from child proc 6 85 1
got from child proc 23 56 1
got from child proc 345 34656 3
got from child proc 245 234 1
got from child proc 57 24656 1
got from child proc 23 45 1
got from child proc 20 100 20
got from child proc 50 200 50
got from child proc 54 235 1
got from child proc 34 2364 2

Process returned 0 (0x0)   execution time : 0.016 s
Press ENTER to continue.

```

input.txt

```
1 2 4 2
2 6 85 1
3 23 56 1
4 345 34656 3
5 245 234 1
6 57 24656 1
7 23 45 1
8 20 100 20
9 50 200 50
10 54 235 1
11 34 2364 2

```

output.txt