Operációs rendszerek BSc

7.gyak.

2021. 03. 24.

Készítette:
Kovács Krisztián
Programtervező informatikus
WIQPM2

Miskolc, 2021

1. Adott négy processz a rendszerbe, melynek beérkezési sorrendje: A, B, C és D. Minden processz USER módban fut és mindegyik processz futásra kész.

   Kezdetben mindegyik processz p_uspri = 60. Az A, B, C processz p_nice = 0, a D processz p_nice = 5. Mindegyik processz p_cpu = 0, az óraütés 1 indul, a befejezés legyen 201. óraütés-ig.

a) Határozza meg az ütemezést RR nélkül és az ütemezést RR-nal - külön-külön táblázatba.
b) Minden óraütem esetén határozza meg a processzek sorrendjét óraütés előtt/után.
c) Igazolja a számítással a tanultak alapján.

	A process		B process		C process		D process		Reschedule				
Clock tick	p_uspri	p_cpu	p_uspri	p_cpu	p_uspri	p_cpu	p_uspri	p_cpu	running before	running after		D p_nice:	5
Starting point	60	0	60	0	60	0	60	0		A		KF	0,857143
1	60	1	60	0	60	0	60	0	A	A			
...													
99	60	99	60	0	60	0	60	0	A	A			
100	72	86	50	0	50	0	60	0	A	B			
101	72	86	50	1	50	0	60	0	B	B			
...													
199	72	86	50	99	50	0	60	0	B	B			
200	69	74	72	86	50	0	60	0	B	C			
201	69	74	72	86	50	1	60	0	C	C			

	A process		B process		C process		D process		Reschedule				
Clock tick	p_uspri	p_cpu	p_uspri	p_cpu	p_uspri	p_cpu	p_uspri	p_cpu	running before	running after		D p_nice:	5
Starting point	60	0	60	0	60	0	60	0		A		KF	0,857143 (2*3)/(2*3+1)
1	60	1	60	0	60	0	60	0	A	A			
...												p_cpu:	p_cpu*KF
9	60	9	60	0	60	0	60	0	A	A			
10	60	10	60	0	60	0	60	0	A	B		p_uspri:	50+p_cpu/4+2*p_nice ; ahol a p_cpu már az újonnan kiszámolt.
11	60	10	60	1	60	0	60	0	B	B			
...													
19	60	10	60	9	60	0	60	0	B	B			
20	60	10	60	10	60	0	60	0	B	C			
...													
30	60	10	60	10	60	10	60	0	C	D			
...													
40	60	10	60	10	60	10	60	10	D	A			
...													
50	60	20	60	10	60	10	60	10	A	B			
...													
60	60	20	60	20	60	10	60	10	B	C			
...													
70	60	20	60	20	60	20	60	10	C	D			
...													
80	60	20	60	20	60	20	60	20	D	A			

80	60	20	60	20	60	20	60	20	D	A
...										
90	60	30	60	20	60	20	60	20	A	B
...										
99	60	30	60	29	60	20	60	20	B	B
100	57	26	57	26	54	17	67	17	B	C
101	57	26	57	26	54	18	67	17	C	C
...										
110	57	26	57	26	54	27	67	17	C	A
...										
120	57	36	57	26	54	27	67	17	A	B
...										
130	57	36	57	36	54	27	67	17	B	C
...										
140	57	36	57	36	54	37	67	17	C	A

	140	57	36	57	36	54	37	67	17	C	A	
...												
	150	57	46	57	36	54	37	67	17	A	B	
...												
	160	57	46	57	46	54	37	67	17	B	C	
...												
	170	57	46	57	46	54	47	67	17	C	A	
...												
	180	57	56	57	46	54	47	67	17	A	B	
...												
	190	57	56	57	56	54	47	67	17	B	C	
...												
	199	57	56	57	56	54	56	67	17	C	C	
	200	62	48	62	48	62	49	64	15	C	D	
	201	62	48	62	48	62	49	64	16	D	D	

Azért fut a 100 óraütés után az A B és C, mert az 54-57 prioritással rendelkező folyamatok azonos sorban vannak.

2. 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


```
fd=open("WIQPM2.txt", O_RDWR);
```

```
close(fd);
```

Először is létrehoztam a .txt fájlt, aminek a tartalma: név, szak, neptunkód. A fájl megnyitását az open() rendszerhívás végzi, melynek 2 atribútuma van: fájl elérési útja, és mire nyissuk meg: olvasás, írás, vagy olvasás és írás. Jelen esetben az utolsó van.

- hiba ellenőrzést,

```
if (fd == -1){
    perror("open() hiba!");
    exit(-1);
}
```

Ha a fájlt nem sikerült megnyitni valamilyen oknál fogva, akkor az open visszatérési értéke -1. Ilyenkor a program kiírja a hibát és leáll. Például ha egy nem létező fájlt adunk meg (esetleg hibás az elérési út), akkor a “No such file or directory” hibaüzenetet kapjuk.

- read() - kiolvassa a neptunkod.txt tartalmát és mennyit olvasott ki (byte), és kiírja konzolra.

```
ret=read(fd, buf, 64);
printf("read() olvasott %d byteot, ami a kovetkezo: %s\n", ret, buf);
```

A karaktertömbbe eltároljuk a read() által beolvasott szöveget. read() paraméterei: fájl, tárolásra kijelölt változó, hány byte-ot olvasson ki a fájlból. Látható, hogy jelenleg 64 byte-ot olvas.

- lseek() – pozícionálja a fájl kurzor helyét, ez legyen a fájl eleje: SEEK_SET, és kiírja a konzolra.

```
ret=lseek(fd, 0, SEEK_SET);
printf("lseek() mondja: %d\n", ret);
```

Beolvasás után a fájlban a kurzor a 64 byte után található, ami jelenleg a fájl vége. Az lseek() rendszerhívással ezt módosíthatjuk. Paraméterei: fájl, hány byte-al tolja el, a kurzor mozgatását mihez képest számoljuk(0, SEEK_SET-fájl eleje, 1, SEEK_CUR-kurzor jelenlegi pozíciója, 2, SEEK_END-fájl vége),

- write() - mennyit ír ki a konzolra.

```
strcpy(buf, "WIQPM2");
```

```
ret=write(fd, buf, 6);
printf("write() mondja: %d\n", ret);
```

A fájlba ír a write() rendszerhívás. Három paramétere van, melyek a következők: fájl, melyik változót írja a fájlba, annak hány byte-át. Látható, hogy a változó értékét lecseréltük a Neptun kódra, ezért az kerül az első hat byte-ra a fájlban.

```
read() olvasott 51 byteot, ami a kovetkezo: Kovacs Krisztian Programtervezo informatikus WIQPM2
lseek() mondja: 0
write() mondja: 6

Process returned 0 (0x0)   execution time : 0.109 s
Press any key to continue.
```