

# **Operációs rendszerek**

## **6. Gyakorlat**

2025. 03. 26.

**Készítette:**

Rác László

Szak: 1.PTI

CI880V

**Sárospatak, 2025**

Adott a következő ütemezési feladat, amit a FCFS, SJF és RR ütemezési algoritmus használatával készítsen el (külön-külön táblázatba):

## I. Határozza meg FCFS, SJF és RR esetén

- A befejezési időt?
- A várakozási/átlagos várakozási időt?
- Ábrázolja Gantt diagram segítségével *az aktív/várakozó processzek* futásának menetét!

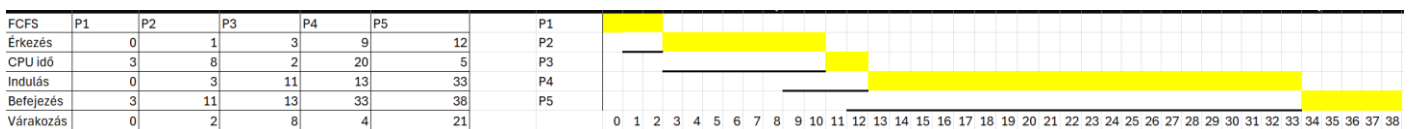
Megj.: a Gantt diagram ábrázolása szerkesztő program segítségével vagy Excel programmal.

- d.) Határozza meg a **processzek végrehajtási sorrendjét!**

**a.) FCFS**

FCFS	P1	P2	P3	P4	P5
Érkezés	0	1	3	9	12
CPU idő	3	8	2	20	5
Indulás	0	3	11	13	33
Befejezés					
Várakozás					

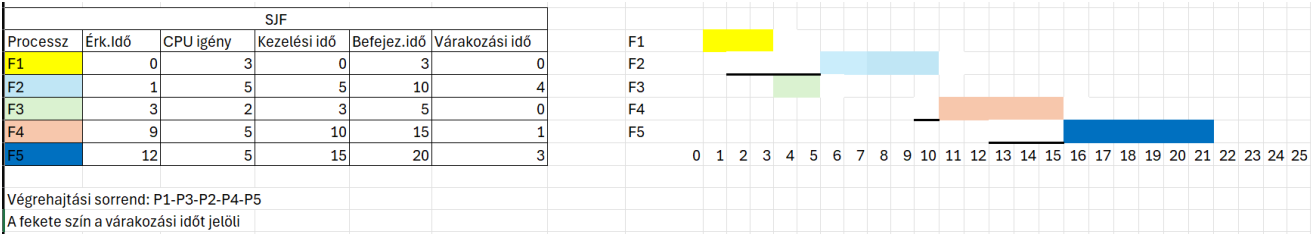
## FCFS megoldás



b.) SJF

SJF					
Processz	Érk. idő	CPU igény	Kezdési idő	Befejez idő	Várakozási idő
F1	0	3	0		
F2	1	5	5		
F3	3	2	3		
F4	9	5	10		
F5	12	5	15		

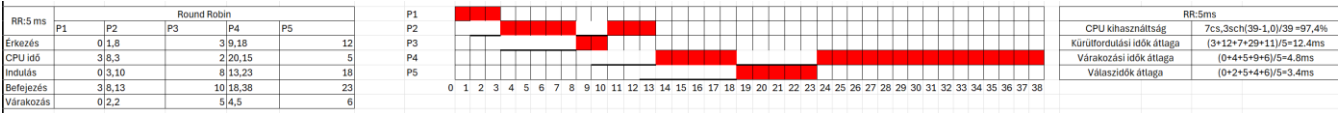
SJF megoldás



c.) RR: 5 ms

RR: 5 ms	Round Robin				
	P1	P2	P3	P4	P5
Érkezés	0	1	3	9	12
CPU idő	3	8	2	20	5
Indulás	0	3	8	13	18
Befejezés					
Várakozás					

RR megoldás



## II. Feladat

„1. Készítsen egy neptunkod\_parent.c és a neptunkod\_child.c programokat. A neptunkod\_parent.c elindít egy gyermek processzt, ami különbözik a szülőtől. A szülő megvárja a gyermek lefutását. A gyermek szöveget ír a szabványos kimenetre (10-szer) (pl. a hallgató nevét és a neptunkód)! - magyarázza egy-egy mondattal.”

A fordítás/futtatás után készítsen egy képernyőképet (minden parancs esetén) és illessze be a dokumentumba.

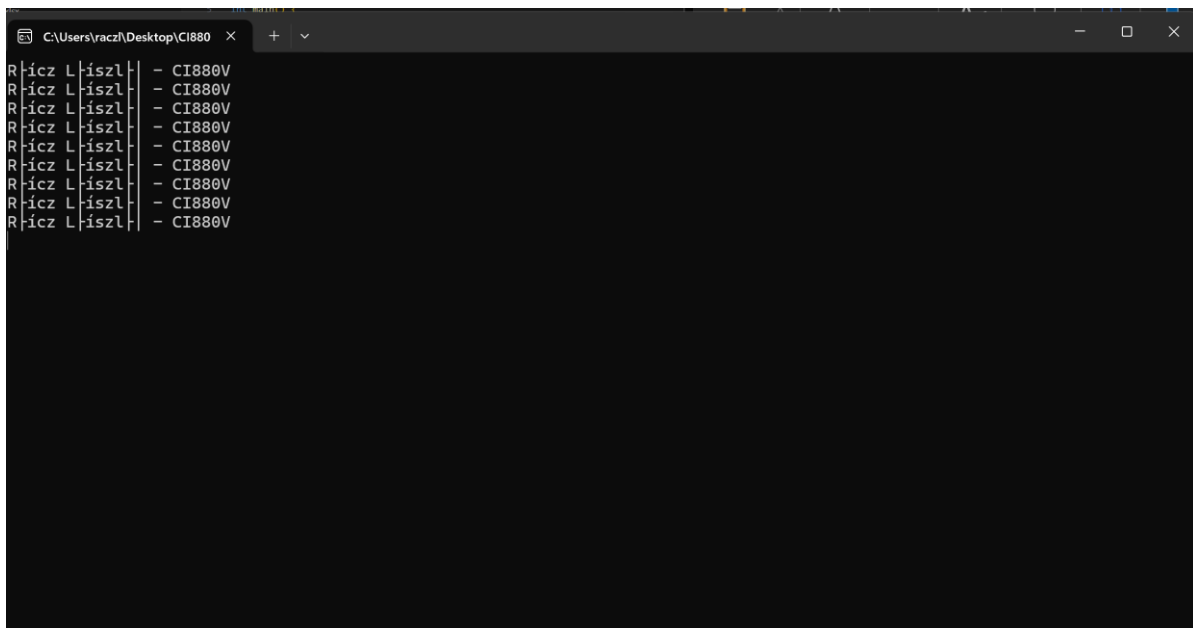
**Mentés:** neptunkod\_parent.c, ill. neptunkod\_child.c

```
C CI880V_parent.c > main()
4
5  int main() {
6      STARTUPINFO si;
7      PROCESS_INFORMATION pi;
8
9      ZeroMemory(&si, sizeof(si));
10     si.cb = sizeof(si);
11     ZeroMemory(&pi, sizeof(pi));
12
13     // Gyermek processz indítása
14     if (!CreateProcess(
15         "CI880V_child.exe",    // A gyermek program neve
16         NULL,                  // Parancssori argumentumok
17         NULL,                  // Processz attribútumok
18         NULL,                  // Szál attribútumok
19         FALSE,                 // Öröklés handle-ekből
20         0,                     // Létrehozási flag-ek
21         NULL,                  // Környezet
22         NULL,                  // Aktuális könyvtár
23         &si,                   // STARTUPINFO
24         &pi)                   // PROCESS_INFORMATION
25     ) {
26         printf("CreateProcess failed (%d).\n", GetLastError());
27         return 1;
28     }
29
30
31     WaitForSingleObject(pi.hProcess, INFINITE);
32
33     CloseHandle(pi.hProcess);
34     CloseHandle(pi.hThread);
35
36     printf("A gyermek processz befejeződött.\n");
37     return 0;
38 }
```

```

C CI880V_child.c > main()
1  #include <stdio.h>
2  #include <windows.h>
3
4  int main() {
5      for (int i = 0; i < 10; i++) {
6          printf("Rácz László - CI880V\n");
7          Sleep(1000);
8      }
9      return 0;
10 }

```



```

C:\Users\raczf\Desktop\CI880
Rácz László - CI880V
Rácz László - CI880V
Rácz László - CI880V
Rácz László - CI880V
Rácz László - CI880V
Rácz László - CI880V
Rácz László - CI880V
Rácz László - CI880V
Rácz László - CI880V
Rácz László - CI880V

```

#### Magyarázat:

- A szülő processz létrehoz egy gyermek processzt a `fork()` hívással, majd az `execl()` segítségével betölti a gyermek programot.
- A gyermek processz 10-szer kiírja a hallgató nevét és neptun kódját a képernyőre.
- A szülő processz a `wait()` hívással megvárja, amíg a gyermek befejezi a futását.