

Operációs Rendszerek BSc

5.gyak.

2021.03.10

Készítette:

Orosz Dániel Bsc

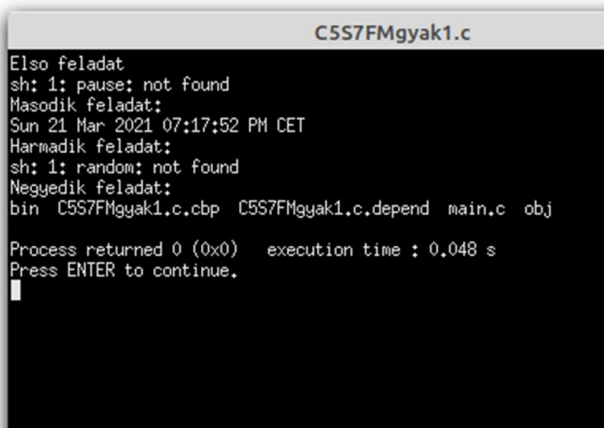
Üzemmmérnök-informatikus

C5S7FM

Miskolc, 2021

1. A system() rendszerhívással hajtson végre létező és nem létező parancsot, és vizsgálja a visszatérési értéket! Mentés: neptunkodgyak1.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      printf("Első feladat\n");
7      system("pause");
8      printf("Második feladat:\n");
9      system("date");
10     printf("Harmadik feladat:\n");
11     system("random");
12     printf("Negyedik feladat:\n");
13     system("ls");
14     return 0;
15 }
16
```

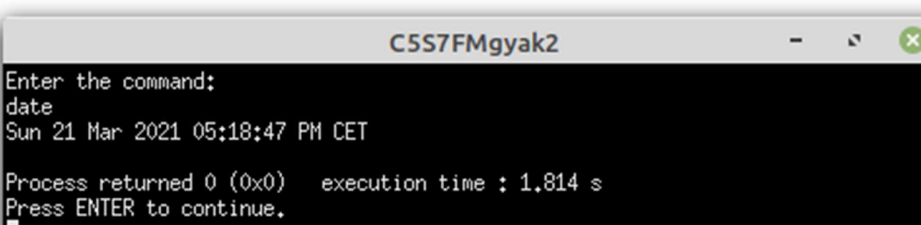


```
C5S7FMgyak1.c
Első feladat
sh: 1: pause: not found
Második feladat:
Sun 21 Mar 2021 07:17:52 PM CET
Harmadik feladat:
sh: 1: random: not found
Negyedik feladat:
bin C5S7FMgyak1.c.cbp C5S7FMgyak1.c.depend main.c obj
Process returned 0 (0x0)   execution time : 0.048 s
Press ENTER to continue.
```

A forráskód C5S7FMgyak1.c néven megtalálható.

2. Írjon programot, amely billentyűzetről bekér Unix parancsokat és végrehajtja őket, majd kiírja a szabványos kimenetre. (pl.: amit bekér: date, pwd, who etc.; kilépés: CTRL-\) Mentés: neptunkodgyak2.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      char input[100];
7      printf("Enter the command:\n");
8      scanf("%s", input);
9      system(input);
10
11     return 0;
12 }
13
```



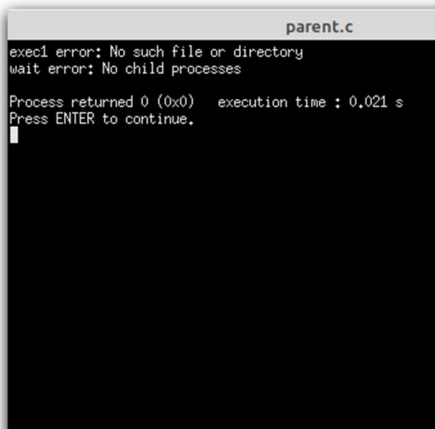
```
C5S7FMgyak2
Enter the command:
date
Sun 21 Mar 2021 05:18:47 PM CET
Process returned 0 (0x0)   execution time : 1.814 s
Press ENTER to continue.
```

A forráskód C5S7FMgyak2.c néven megtalálható.

3. Készítsen egy parent.c és a child.c programokat. A parent.c elindít egy gyermek processzt, ami különbözik a szülőtől. A szülő megvárja a gyermek lefutását. A gyermek szöveget ír a szabványos kimenetre (5-ször) (pl. a hallgató neve és a neptunkód)! Mentés: parent.c, ill. child.c

Parent.c:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <sys/types.h>
5  #include <sys/time.h>
6
7  int main()
8  {
9      pid_t pid;
10
11     if ((pid = fork()) < 0){
12         perror("Process error");
13     }
14     else if (pid == 0){
15         if (execl("./child", "child", (char *) NULL) < 0){
16             perror("execl error");
17         }
18     }
19
20     if (waitpid(pid, NULL, 0) < 0){
21         perror("wait error");
22     }
23
24     return 0;
25 }
26
```

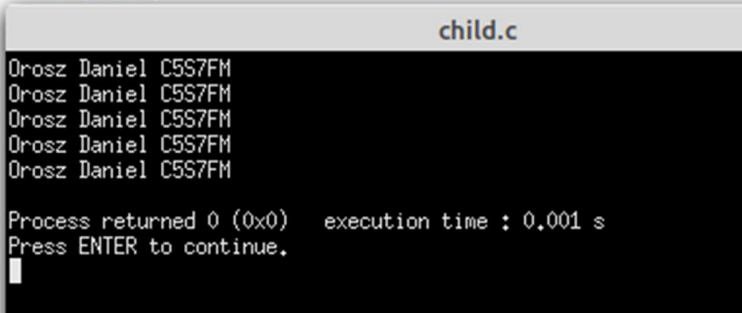


```
parent.c
execl error: No such file or directory
wait error: No child processes
Process returned 0 (0x0)   execution time : 0.021 s
Press ENTER to continue.
```

Forráskód parent.c néven megtalálható.

Child.c:

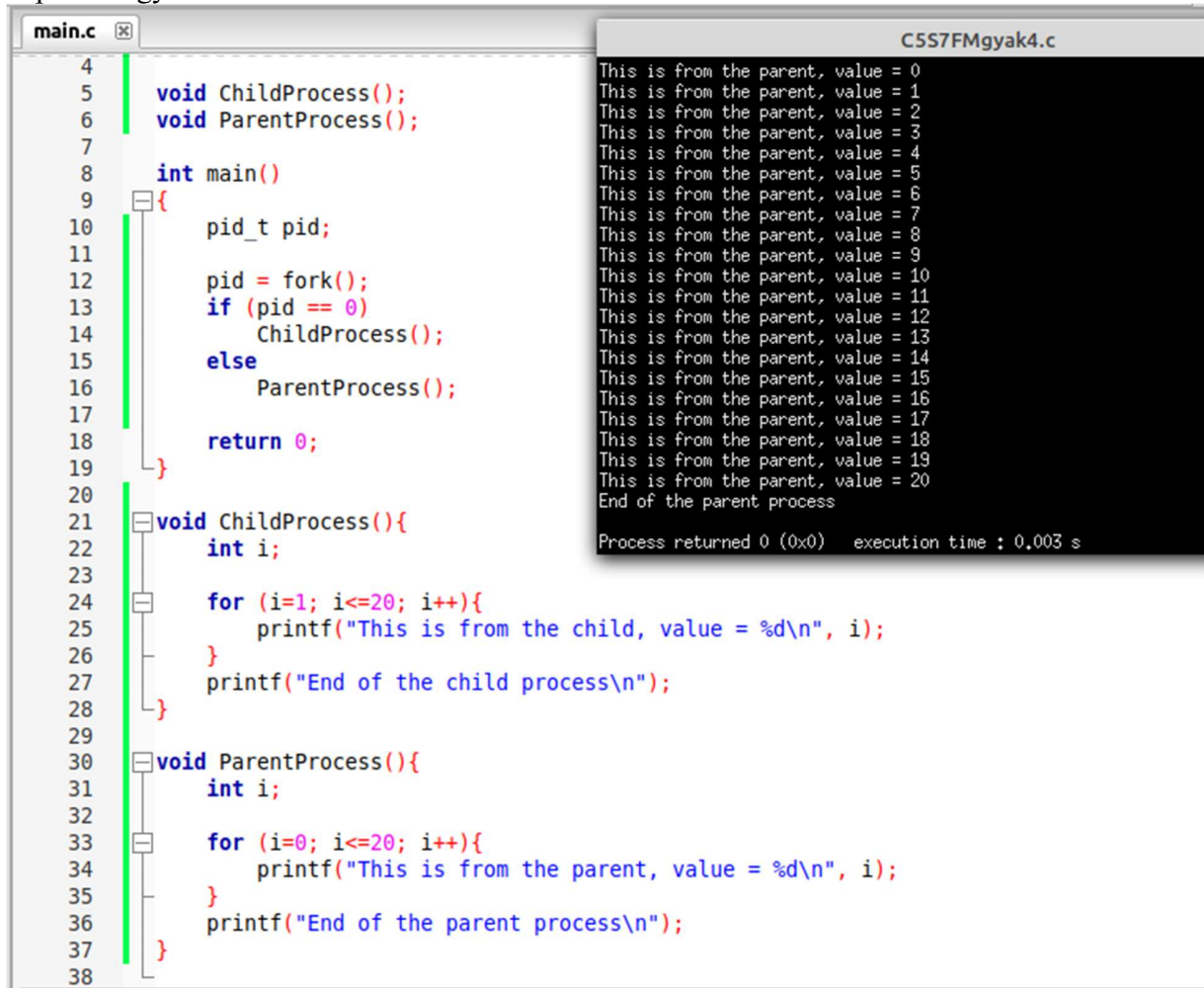
```
main.c
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      for (int i=0; i<5; i++){
7          printf("Orosz Daniel C5S7FM\n");
8      }
9      return 0;
10 }
11
```



```
child.c
Orosz Daniel C5S7FM
Orosz Daniel C5S7FM
Orosz Daniel C5S7FM
Orosz Daniel C5S7FM
Orosz Daniel C5S7FM
Process returned 0 (0x0)   execution time : 0.001 s
Press ENTER to continue.
```

A forráskód child.c néven megtalálható.

4. A fork() rendszerhívással hozzon létre egy gyerek processzt-t és abban hívjon meg egy exec családbeli rendszerhívást (pl. execlp). A szülő várja meg a gyerek futását! Mentés: neptunkodgyak4.c



The image shows a code editor window titled 'main.c' and a terminal window titled 'C5S7FMgyak4.c'. The code in 'main.c' defines a parent process that forks a child process. The child process prints values from 1 to 20, and the parent process prints values from 0 to 20. The terminal output shows the execution of the program, displaying the parent's output first, followed by the child's output, and finally the parent's completion message.

```
main.c
4
5 void ChildProcess();
6 void ParentProcess();
7
8 int main()
9 {
10     pid_t pid;
11
12     pid = fork();
13     if (pid == 0)
14         ChildProcess();
15     else
16         ParentProcess();
17
18     return 0;
19 }
20
21 void ChildProcess(){
22     int i;
23
24     for (i=1; i<=20; i++){
25         printf("This is from the child, value = %d\n", i);
26     }
27     printf("End of the child process\n");
28 }
29
30 void ParentProcess(){
31     int i;
32
33     for (i=0; i<=20; i++){
34         printf("This is from the parent, value = %d\n", i);
35     }
36     printf("End of the parent process\n");
37 }
38
```

```
C5S7FMgyak4.c
This is from the parent, value = 0
This is from the parent, value = 1
This is from the parent, value = 2
This is from the parent, value = 3
This is from the parent, value = 4
This is from the parent, value = 5
This is from the parent, value = 6
This is from the parent, value = 7
This is from the parent, value = 8
This is from the parent, value = 9
This is from the parent, value = 10
This is from the parent, value = 11
This is from the parent, value = 12
This is from the parent, value = 13
This is from the parent, value = 14
This is from the parent, value = 15
This is from the parent, value = 16
This is from the parent, value = 17
This is from the parent, value = 18
This is from the parent, value = 19
This is from the parent, value = 20
End of the parent process
Process returned 0 (0x0)   execution time : 0.003 s
```

A forráskód C5S7FMgyak4.c néven megtalálható.

5. A fork() rendszerhívással hozzon létre gyerekeket, várja meg és vizsgálja a befejeződési állapotokat (gyerekekben: exit, abort, nullával való osztás)! Mentés: neptunkodgyak5.c

```

4  int main()
5  {
6      int pid, pid1, pid2;
7
8      pid = fork();
9
10     if (pid == 0){
11         sleep(5);
12         printf("child1 -> pid = %d and ppid = %d\n", getpid(), getppid());
13     }
14     else{
15         pid1 = fork();
16         if (pid1 == 0){
17             sleep(3);
18             printf("child2 -> pid = %d and ppid = %d\n", getpid(), getppid());
19         }
20         else{
21             pid2 = fork();
22             if (pid2 == 0){
23                 printf("child3 -> pid = %d and ppid = %d\n", getpid(), getppid());
24             }
25             else{
26                 sleep(5);
27                 printf("parent -> pid = %d\n", getpid());
28             }
29         }
30     }
31
32     return 0;
33 }

```

```

C5S7FMgyak5
child3 -> pid = 2866 and ppid = 2863
child2 -> pid = 2865 and ppid = 2863
parent -> pid = 2863
child1 -> pid = 2864 and ppid = 2863

Process returned 0 (0x0)   execution time : 5.006 s
Press ENTER to continue.

```

A forráskód C5S7FMgyak5.c néven megtalálható.

6. Adott a következő ütemezési feladat, ahol a RR ütemezési algoritmus használatával készítse el: Határozza meg a

- Ütemezze az adott időszeket alapján az egyes processzek paramétereit (ms)!
- A rendszerben lévő processzek végrehajtásának sorrendjét?
- Ábrázolja Gantt diagram segítségével az aktív/várakozó processzek futásának menetét!

A feladat mindennel együtt megtalálható a C5S7FMgyak5.xlsx fájlban.

a.)

RR: 5ms	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	3	13	10	38	23
Várakozás	0	4	5	9	6

b.) P1-P2-P3-P2-P4-P5-P4-P4-P4

c.)

												1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	3	3	3	3	3	3	3		
	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8
P 1																																							
P 2																																							
P 3																																							
P 4																																							
P 5																																							

A b r sz n a processz v rako z s t jel li.