Miskolci Egyetem

Gépészmérnöki- és Informatikai Kar (GÉIK)

Gazdaságinformatikus alapképzés (G 2BGI)

2020/2021 II. (tavaszi) félév

Operációs rendszerek BSc (GEIAL302A-B)

tantárgy

2021 tavasz féléves feladat

MISKOLCI EGYETEM

UNIVERSITY OF MISKOLC

Készítette:

Név: Szeli Márk

NEPTUN kód: B8VNQ7

Miskolc
2021.

Feladat leírása

3. Írjon C nyelvű programot, ami:

- létrehoz két gyermekprocesszt,
- ezek a gyermekprocesszek létrehoznak 3-3 további gyermeket,
- ezek az unokák várakoznak néhány másodpercet és szűnjenek meg,
- a szülők várják meg a gyermekek befejeződését és csak utána szűnjenek meg.

A feladat elkészítésének lépései

Első lépésben a szükséges header állományokat deklarálom, ugyanis a *sys/wait.h* állomány nélkül nem működne programom.

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h> // Szükséges, hiányos header állomány deklarálása.
```

Következő lépésként a feladat leírását kiírom megjegyzésként.

```c
/*
Feladat leírása:
--------------------------------------------------------------------------------
    3. Írjon C nyelvű programot, ami:
        - létrehoz két gyermekprocesszt,
        - ezek a gyermekprocesszek létrehoznak 3-3 további gyermeket,
        - ezek az unokák várakoznak néhány másodpercet és szűnjenek meg,
        - a szülők várják meg a gyermekek befejeződését és csak utána szűnjenek meg.
*/
```

Ezután a *main* főprogramban deklarálok két különböző tömböt. Első tömb neve: *gyermek_pid*, típusa int, mérete 2, amely a gyermekek PID (Process ID) értékét tárolja. Második tömb neve: *unoka_pid*, típusa szintén int, mérete 6, amely az unokák PID (Process ID) értékét tárolja. A Process ID értéke az adott processz egyéni azonosítóját jelenti. A *pid_t* a processzek azonosítását jelenti, végül pedig létrehoztam egy *alvas* változót, melynek értéke 10, melynek a későbbiekben lesz jelentősége.

```c
int main()
{
    int gyermek_pid[2]; // Gyermekprocesszek PID értékeineknek tárolására szolgáló tömb.
    int unoka_pid[6]; // Unokaprocesszek PID értékeineknek tárolására szolgáló tömb.
    pid_t pid; // pid_t adattípus a processzek azonosítását jelenti.
    int alvas = 10; // A várakozásra szolgáló változó, melynek értéke 10.
```

Ezek után kiíratom a képernyőre a szülőprocessz létezését.

```c
printf("Szülőprocessz vagyok!\n"); // Szülőprocessz kiíratása.
```

Ezután indítok egy számláló (for) ciklust, amely a szülőprocessz gyermekeit (két gyermekprocessz) hozza létre. Mivel két gyermeket kell létrehozni, az *i* változó 0-tól indul és <2-ig megy, így két gyermekprocessz kerül létrehozásra.

```
for(int i = 0; i < 2; i++) // A 2 gyermekprocessz létrehozását szolgáló számláló (for) ciklus.
{
```

A cikluson belül egy hibaellenőrzést futtatok, amely hiba esetén kiírja a *Fork ERROR!* szöveget.

```
if((pid = fork()) < 0) // Hibaellenőrzés.
{
    perror("Fork ERROR!\n"); // Fork error kiíratása.
}
```

Jelen elágazásban, ha a feltétel teljesül (azaz a *fork()* függvény visszatérési értéke 0), akkor lefut az *if* ág, azaz létrehozásra kerülnek a gyermekprocesszek.

```
if(fork() == 0) // A gyermekprocessz létrehozása. Ha a fork() rendszerhívás visszatérési értéke 0, akkor lefut az elágazás.
{
```

Az elágazáson belül először beállítom, hogy a tömb (*gyermek_pid*) i-edik eleme legyen a létrehozott gyermekprocessz PID-je.

```
gyermek_pid[i] = getpid(); // A tömb i-edik elemének beállítása.
```

A beállítás után kiírtam a gyermekprocessz létezését, majd annak PID értékét (*getpid()*), valamint szülőprocesszének PID értékét (*getppid()*).

```
printf("Gyermekprocessz vagyok! PID értékem: %d, Szülőm PID értéke: %d\n", getpid(), getppid()); // A gyermekprocessz, annak PID-je, valamint a szülőjének PID-jének kiíratása.
```

A kiírás után jöhet a gyermekprocesszek gyermekprocesszének, azaz az unokaprocesszek létrehozása. A létrehozás hasonló, mint a gyermekprocessz létrehozása. Indítok egy újabb számláló (for) ciklust, amely a gyermekprocessz gyermekeit (három unokaprocesszt) hozza létre. Abban különbözik az előzőtől, hogy az i értéke <3-ig megy, ezáltal három-három gyermekprocesszt (összesen hat unokaprocesszt) hoz létre.

```
for(int i = 0; i < 3; i++) // 3-3 unokaprocesszek létrehozására szolgáló számláló (for) ciklus.
{
```

A cikluson belül egy hibaellenőrzést futtatok, amely hiba esetén kiírja a *Fork ERROR!* szöveget.

```
if((pid = fork()) < 0) // Hibaellenőrzés.
{
    perror("Fork ERROR!\n"); // Fork error kiíratása.
}
```

Jelen elágazásban, ha a feltétel teljesül (azaz a *fork()* függvény visszatérési értéke 0), akkor lefut az *if* ág, azaz létrehozásra kerülnek az unokaprocesszek.

```
if(fork() == 0) // Az unokaprocesszek létrehozása. Ha a fork() rendszerhívás visszatérési értéke 0, akkor lefut az elágazás.
{
```

Az elágazáson belül először beállítom, hogy a tömb (*unoka_pid*) i-edik eleme legyen a létrehozott unokaprocessz PID-je.

```
unoka_pid[i] = getpid(); // A tömb i-edik elemének beállítása.
```

A beállítás után kiírtam az unokaprocesszek létezését, majd azok PID értékét (*getpid()*), valamint szülőprocesszüknek (azaz a gyermekprocesszek) PID értékét (*getppid()*).

```
printf("Unokaprocessz vagyok! Unokaprocessz PID értékem: %d, Szülőm PID értéke: %d\n", getpid(), getppid()); // Az uokaprocessz, annak PID-je, valamint a szülőjének PID-jének kiíratása.
```

Az unokaprocesszek 10 másodpercet várakoznak, ugyanis az *alvas* változó értéke 10.

```
sleep(alvas); // Az unokaprocesszek 10 másodperc után megszűnnek.
```

A megszűnés eredményét kiíratom, valamint azt is, hogy mely PID értékű unokaprocessz szűnt meg.

```
printf("Megszűnt unokaprocessz! A megszűnt unokaprocessz PID értéke: %d\n", getpid()); // Az unokaprocessz megszűnésének kiíratása, valamint annak PID-je.
```

Majd exitál, befejeződik a processzek élettartama.

```
exit (0);
```

Az elágazáson kívül a *wait(NULL)* paranccsal megvárjuk, hogy az unokaprocesszek megszűnjenek.

```
wait(NULL); // Unokaprocesszekre való várakozás.
```

A megszűnés eredményét kiíratom, valamint azt is, hogy mely PID értékű gyermekprocessz szűnt meg.

```
printf("Megszűnt gyermekprocessz! A megszűnt gyermekprocessz PID értéke: %d\n", getpid()); // A gyermkeporcessz megszűnésének kiíratása, valamint annak PID-je.
```

Majd exitál, befejeződik a processzek élettartama.

```
exit (0);
```

Az elágazáson és a számláló cikluson kívül a *wait(NULL)* paranccsal megvárjuk, hogy a gyermekprocesszek megszűnjenek.

```
wait(NULL); // Gyermekprocesszekre való várakozás.
```

A megszűnés eredményét kiíratom.

```
printf("Megszűnt szülőprocessz!\n"); // Szülőprocessz megszűnésének kiíratása.
```

Végül megszűnik a szülőprocessz is és a program leáll.

```
return 0;
```

A program teljes képe:

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h> // Szükséges, hiányos header állomány deklarálása.

/*
Feladat leírása:
--------------------------------------------------------------------------------
    3. Írjon C nyelvű programot, ami:
        - létrehoz két gyermekprocesszt,
        - ezek a gyermekprocesszek létrehoznak 3-3 további gyermeket,
        - ezek az unokák várakoznak néhány másodpercet és szűnjenek meg,
        - a szülők várják meg a gyermekek befejeződését és csak utána szűnjenek meg.
*/

int main()
{
    int gyermek_pid[2]; // Gyermekprocesszek PID értékeineknek tárolására szolgáló tömb.
    int unoka_pid[6]; // Unokaprocesszek PID értékeineknek tárolására szolgáló tömb.
    pid_t pid; // pid_t adattípus a processzek azonosítását jelenti.
    int alvas = 10; // A várakozásra szolgáló változó, melynek értéke 10.

    printf("Szülőprocessz vagyok!\n"); // Szülőprocessz kiíratása.

        for(int i = 0; i < 2; i++) // A 2 gyermekprocessz létrehozását szolgáló számláló (for) ciklus.
        {
            if((pid = fork()) < 0) // Hibaellenőrzés.
            {
                perror("Fork ERROR!\n"); // Fork error kiíratása.
            }

            else if(pid == 0) // A gyermekprocessz létrehozása. Ha a fork() rendszerhívás visszatérési értéke 0, akkor lefut az elágazás.
            {
                gyermek_pid[i] = getpid(); // A tömb i-edik elemének beállítása.

                printf("Gyermekprocessz vagyok! PID értékem: %d, Szülőm PID értéke: %d\n", getpid(), getppid()); // A gyermekprocessz, annak PID-je, valamint a szülőjének PID-jének kiíratása.

                for(int i = 0; i < 3; i++) // 3-3 unokaprocesszek létrehozására szolgáló számláló (for) ciklus.
                {
                    if((pid = fork()) < 0) // Hibaellenőrzés.
                    {
                        perror("Fork ERROR!\n"); // Fork error kiíratása.
                    }

                    else if(pid == 0) // Az unokaprocesszek létrehozása. Ha a fork() rendszerhívás visszatérési értéke 0, akkor lefut az elágazás.
                    {
                        unoka_pid[i] = getpid(); // A tömb i-edik elemének beállítása.
                        printf("Unokaprocessz vagyok! Unokaprocessz PID értékem: %d, Szülőm PID értéke: %d\n", getpid(), getppid()); // Az uokaprocessz, annak PID-je, valamint a szülőjének PID-
                        sleep(alvas); // Az unokaprocesszek 10 másodperc után megszűnnek.
                        printf("Megszűnt unokaprocessz! A megszűnt unokaprocessz PID értéke: %d\n", getpid()); // Az unokaprocessz megszűnésének kiíratása, valamint annak PID-je.
                        exit (0);
                    }
                }

                wait(NULL); // Unokaprocesszekre való várakozás.
                printf("Megszűnt gyermekprocessz! A megszűnt gyermekprocessz PID értéke: %d\n", getpid()); // A gyermkeporcessz megszűnésének kiíratása, valamint annak PID-je.
                exit (0);
            }
        }

    wait(NULL); // Gyermekprocesszekre való várakozás.
    printf("Megszűnt szülőprocessz!\n"); // Szülőprocessz megszűnésének kiíratása.
    return 0;
}
```

A futtatás eredménye

```
~$ ./beadando
Szülőprocessz vagyok!
Gyermekprocessz vagyok! PID értékem: 1315, Szülőm PID értéke: 1314
Unokaprocessz vagyok! Unokaprocessz PID értékem: 1317, Szülőm PID értéke: 1315
Unokaprocessz vagyok! Unokaprocessz PID értékem: 1318, Szülőm PID értéke: 1315
Unokaprocessz vagyok! Unokaprocessz PID értékem: 1319, Szülőm PID értéke: 1315
Gyermekprocessz vagyok! PID értékem: 1316, Szülőm PID értéke: 1314
Unokaprocessz vagyok! Unokaprocessz PID értékem: 1320, Szülőm PID értéke: 1316
Unokaprocessz vagyok! Unokaprocessz PID értékem: 1321, Szülőm PID értéke: 1316
Unokaprocessz vagyok! Unokaprocessz PID értékem: 1322, Szülőm PID értéke: 1316
Megszűnt unokaprocessz! A megszűnt unokaprocessz PID értéke: 1317
Megszűnt unokaprocessz! A megszűnt unokaprocessz PID értéke: 1318
Megszűnt unokaprocessz! A megszűnt unokaprocessz PID értéke: 1319
Megszűnt unokaprocessz! A megszűnt unokaprocessz PID értéke: 1320
Megszűnt unokaprocessz! A megszűnt unokaprocessz PID értéke: 1321
Megszűnt unokaprocessz! A megszűnt unokaprocessz PID értéke: 1322
Megszűnt gyermekprocessz! A megszűnt gyermekprocessz PID értéke: 1316
Megszűnt gyermekprocessz! A megszűnt gyermekprocessz PID értéke: 1315
Megszűnt szülőprocessz!
~$
```

1. lépés: A szülőprocessz létezésének kiírása.
2. lépés: Az elsőként létrejött gyermekprocessz, PID értékének és szülője PID értékének kiírása. A gyermekprocessz PID értéke 1315, míg szülőjének PID értéke 1314.
3. lépés: Az elsőként létrejött gyermekprocessz unokáinak létrejötte és PID értékük kiírása, valamint szülőjük PID értékének (1315) kiírása. A létrejött unokaprocesszek PID-jük: 1317, 1318, 1319.
4. lépés: A másodikként létrejött gyermekprocessz, PID értékének és szülője PID értékének kiírása. A gyermekprocessz PID értéke 1316, míg szülőjének PID értéke 1314.
5. lépés: A másodikként létrejött gyermekprocessz unokáinak létrejötte és PID értékük kiírása, valamint szülőjük PID értékének (1316) kiírása. A létrejött unokaprocesszek PID-jük: 1320, 1321, 1322.
6. lépés: Az első gyermekprocessz unokáinak megszűnése (PID értékük: 1317, 1318, 1319).
7. lépés: A második gyermekprocessz unokáinak megszűnése (PID értékük: 1320, 1321, 1322).
8. lépés: A második gyermekprocessz megszűnése (PID értéke: 1316).
9. lépés: Az első gyermekprocessz megszűnése (PID értéke: 1315).
10. A szülőprocessz megszűnése (PID értéke: 1314).