Operációs rendszerek

5. gyakorlat:
Processzus kontextus

Gyakori programozói feladat, hogy az egyik programnak el kell indítania egy másikat. Ezen a gyakorlaton ennek a megvalósításával fogunk foglalkozni.

Egy processzus által létrehozott processzust az előző gyermekének nevezzük, az előzőt pedig szülőnek.

Ha egy gyermek processzust kell indítani, akkor felmerül két kérdés:

- A gyermek a szülő másolata, vagy teljesen más?
- A gyermek a szülővel párhuzamosan fut, vagy a szülő várakozik, amíg a gyermek befejeződik?

Speciális esetben úgy hozunk létre új processzust, hogy függetlenítjük azt mindentől, amit a szülőjétől örökölt és így egy un. *daemon* processzust hozhatunk létre.

Néha szükség lehet arra, hogy az egyik processzus futása közben kicseréljük a program kódot a processzus kontextusában, vagyis egy másik bináris állományt töltsünk a kontextusba. (Ez nem is olyan ritka, hiszen UNIX rendszerre karakteres terminálon való bejelentkezéskor ugyanabba a processzus kontextusba három különböző program is betöltődik egymás után!)

A system() függvény

A *system()* függvényt használhatjuk arra, ha új processzust akarunk létrehozni úgy, mint egy ahogyan azt a parancssorban tennénk.

```
#include <stdlib.h>

int system(const char* cmd);
```

A függvény meghívása azt eredményezi, hogy elindul egy gyermek processzus, amiben betöltődik egy bourne shell aminek –c kapcsolója után az argumentumban megadott sztring áll. Ennek hatására a shell végrehajtja a sztringben leírt utasítást úgy, mintha azt a parancssorba írtuk volna be. Visszatérési értéke a megadott utasítása kilépési kódja.

A legtöbb helyen intik a programozókat a függvény használatától, mert biztonsági problémákat vet fel és bizonyos esetekben jogosulatlan rendszergazdai hozzáférést adhat a felhasználónak.

Pl.

```
#include <stdlib.h>

int main(void)
{
	printf("Program indulasa\nkilso program meghivasa\n");
	printf("a kulso program befejezodott, exitcode = %d\n", \
		system("date +\"%Y. %m. %d. %H:%M %Z(%z)\""));
	printf("program befejezese\n");
}
```

Az exec() függvények

A legalapvetőbb tevékenységek közé tartozik egy programban egy másik program „visszatérés nélküli” végrehajtása. UNIX rendszereken ennek megoldása a már létező processzus kontextusba, új programkódot betöltése. Ennek tulajdonsága, hogy a processzus azonosítói (PID/GID), jogosultságai (UID/GID/EUID/EGID), memória foglalása[1], megnyitott fájljai nem változnak.

Ilyen művelet megvalósítására szolgálnak az *exec*()* függvények. Ezekből a függvényekből több is rendelkezésre áll, melyek funkciója megegyezik, csak az argumentumaiban van eltérés. Az összes függvénynek viszont egy az alapja, ez az *execve()* függvény.

Az execve() függvény

```
#include <unistd.h>

int execve(const char * file, const char * arg[], const char *envp[]);
```

A függvény első argumentuma annak a fájlnak a neve (teljes elérési útvonallal) amelyiket a processzus kontextusába be szeretnénk tölteni. A második argumentum sztringek tömbje, amit a betöltendő program kap argumentumként. A harmadik argumentum szintén sztringek tömbje, ami a betöltendő program környezete (environment) lesz, és *változó=érték* párokban kell megadni a sztringeket.

A függvény hívás sikere esetén nincs visszatérési értéke, hiszen ha sikerül egy másik kódot betölteni a kontextusba, akkor az fog futni és a függvényt meghívó kódba nem

[1] UNIX processzusok általános tulajdonsága, hogy memóriafoglalásuk nem csökken életük során. A free() hívással ugyan felszabadul memória, de az továbbra is a processzus tulajdonában marad, nem kerül vissza az operációs rendszer tulajdonába.

tér vissza. (Ez tehát egy olyan függvény ami sikeres végrehajtás esetén nincs visszatérés.) Sikertelen végrehajtás estén a visszatérési érték -1 lesz és a hiba kódját pedig az *errno* változó tartalmazza.

Új processzusok létrehozása

Természetesen van lehetőség egy programban arra, hogy gyermek processzusokat hozzon létre. Ilyenkor a processzusok futhatnak egymást megvárva és egymással párhuzamosan is.

A fork() függvény

```
#include <sys/types.h>
#include <unistd.h>

pid_t fork();
```

A függvény hatására létrejön egy új processzus, ami a hívónak a gyermeke lesz. A gyermek örökli a szülőtől a program kódot, az adat szegmenst, a regisztereket (beleértve az IP-t is), minden nyitott fájlt és a jogosultságait (UID/GID/EUID/EGID). Ez gyakorlatilag azt jelenti, hogy az újonnan létrejött processzus másolata a szülőnek és úgy kezdődik a végrehajtása, mintha éppen a *fork()* függvényből térne vissza. A különbség – ami alapján meg lehet a programban különböztetni a szülőt a gyermektől – az, hogy a függvény visszatérési értéke a szülőben a létrejött gyermek PID-je, a gyermekben pedig nulla (0)! Ha sikertelen a hívás, akkor nem jön létre gyermek processzus és a szülőben a visszatérési érték –1 (a hiba kódját pedig az *errno* változó tartalmazza).

Történeti áttekintést és technikai magyarázatot lásd a http://www.faqs.org/faqs/unix-faq/programmer/faq/ címen az 1. fejezetben a vfork() függvénynél.

Processzus befejezése, az exit() függvény

```
#include <stdlib.h>

void exit(int status);
```

Ez egy speciális függvény hívás, mert ebből a függvényből sosincs visszatérés. Hatására a hívó processzus befejeződik és az argumentumban megadott *status* lesz az exit kódja.

A függvény tevékenységei:

1. Minden megnyitott fájlt lezár.
2. A processzus minden gyermekének a szülőjét átállítja 1-re (init).

3. A processzus szülőjének küld egy SIGCHLD szignált.
4. Beállítja a processzus exit státuszát az argumentumban adott értékre
5. A processzus állapotát Zombi-ra állítja.

Várakozás a gyermek processzus befejeződésére, a wait() függvény

```
#include <sys/types.h>
#include <sys/wait.h>

pid_t wait(int *status);
```

Ha egy processzus ezt a függvényt meghívja, akkor azzal blokkolódik és csak valamelyik gyermek processzusának befejeződése hatására folytatódik. Ekkor a függvény visszatérési értéke a befejeződött processzus azonosítója, a *status* argumentumban adott változó pedig, az alsó 8 bitjén tartalmazza az operációs rendszer által adott kilépési kódok (a rendszer elképzelése a processzus befejeződéséről), a felső 8 biten pedig azt kapjuk, amit a gyermekprocesszus main függvényének visszatérési értéke, ill. az exit függvény argumentuma.

Történeti áttekintést és technikai magyarázatot lásd a http://www.faqs.org/faqs/unix-faq/programmer/faq/ címen az 1. fejezetben a_exit() függvénynél.