Sisteme de operare
- Laborator 9 -

PROCESE UNIX

1. PROCESE UNIX

- **proces** = *program aflat în execuţie*
- fiecare proces are un identificator de proces unic, un număr întreg pozitiv. Acesta e denumit pe scurt `pid` (`Process IDentifier`).
- nucleul (*kernel*) sistemului de operare întreţine permanent o tabelă cu procese
- examinarea proceselor:

```
ps -e
ps -f -p 1
```

Procese UNIX speciale:

- procesul cu `pid 0` este planificatorul de procese, numit ***swapper***. Acest proces face parte din nucleu și este considerat un proces sistem.
- procesul cu `pid 1` este procesul ***init***, invocat de nucleu la sfârșitul procedurii de încărcare a sistemului. Fișierul program pentru acest proces era */etc/init* în versiunile vechi și */sbin/init* în versiunile noi. Procesul *init* citește fișierele de inițializare dependente de sistem (fișiserele de resurse */etc/rc**) și aduce sistemul într-o stare stabilă (de exemplu multiuser). Procesul *init* este un proces utilizator care se execută cu drepturi de superuser.

2. FUNCŢIA fork() -> Crearea unui proces

- crearea unui proces se realizează prin apelul sistem *fork*, apel care are sintaxa:

```
#include <unistd.h>
pid_t fork(void);
```

- crează un nou proces prin duplicarea procesului apelant (**procesul părinte**)
- noul proces este denumit **proces copil/fiu** şi este o copie aproape exactă a procesului părinte
- = procesul care apelează funcția *fork* se numește ***proces părinte***, procesul nou creat fiind ***procesul fiu***.
- cele două procese îşi continuă execuţia cu instrucţiunea care urmează apelului **fork()**

- returnează:
  - valoarea `0` - în procesul copil
  - identificatorul procesului copil (`child PID`) - în procesul părinte
  - valoarea `-1` - dacă apelul a eşuat
- apelul funcţiei **`fork()`** eşuează în următoarele situaţii:
  - nu există spaţiu de memorie suficient pentru duplicarea procesului părinte
  - numărul total de procese depăşeşte limita maximă admisă
- **exemple**: `fork_1.c, fork_2.c, fork_3.c`

3. FUNCȚIA exit() -> terminarea unui proces

- procesul fiu semnalează terminarea sa tatălui aflat în așteptare prin intermediul apelului de sistem *`exit`* sau *`_exit`*. Apelul *`exit`* transmite prin parametrul său un număr care semnalează tatălui o terminare normală sau cu eroare. Prin convenție, un cod de stare `0` semnifică terminarea normală a procesului, iar un cod diferit de zero indică apariția unei erori.
- Sintaxa apelurilor este:

```
#include <stdlib.h>

void exit(int status);
```

- Apelul *`exit`* termină procesul care-l execută cu un cod de stare egal cu octetul mai puțin semnificativ al cuvântului de stare, *`status`*, și închide toate fișierele deschise de acesta. După aceea, procesului tată este transmis semnalul *`SIGCHLD`*.
- Pentru procesele aflate într-o relație părinte-fiu la un apel *`exit`* sunt esențiale trei cazuri:
  - procesul părinte se termină înaintea procesului fiu;
  - procesul fiu se termină înaintea procesului părinte;
  - procesul fiu, moștenit de procesul *`init`* se termină.
- Procesul *`init`* devine părintele oricărui proces pentru care procesul părinte s-a terminat. Când un proces se termină, nucleul parcurge toate procesele active pentru a vedea dacă printre ele există un proces care are ca părinte procesul terminat. Dacă există un astfel de proces, pid-ul procesului părinte devine 1 (pid-ul procesului *`init`*). Nucleul garantează astfel că fiecare proces are un părinte.
- Dacă procesul fiu se termină înaintea procesului părinte, nucleul trebuie să păstreze anumite informații (pid, starea de terminare, timp de utilizare CPU) asupra modului în care fiul s-a terminat.

3. FUNCŢIILE wait(), waitpid()

- sintaxa:

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


pid_t wait(int *status);

pid_t waitpid(pid_t pid, int *status, int options);
```

- **wait()** suspendă execuţia procesului apelant până la terminarea unui proces copil
- apelul **wait(&status)** este echivalent cu **waitpid(-1, &status, 0)**
- **waitpid()** suspendă execuţia procesului apelant până la apariţia unuia dintre următoarele evenimente:
  - procesul fiu specificat prin argumentul **pid** şi-a terminat execuţia
  - procesul fiu specificat prin argumentul **pid** a fost oprit printr-un semnal
  - procesul fiu specificat prin argumentul **pid** a fost repornit printr-un semnal
- semnificaţia valorilor argumentului **pid**:

pid	**Semnificaţie**
< -1	Se aşteaptă terminarea tuturor proceselor copil al căror identificator de grup (GID) este egal cu valoarea absolută a parametrului pid
-1	Se aşteaptă terminarea tuturor proceselor copil
0	Se aşteaptă terminarea tuturor proceselor copil al căror identificator de grup (GID) este egal cu GID-ul procesului părinte
> 0	Se aşteaptă terminarea procesului cu PID-ul specificat prin parametrul pid

- **exemple:** fork_4.c, fork_5.c

4. FUNCŢIA signal()

- sintaxa:

```
#include <signal.h>


sighandler_t signal(int signum, sighandler_t handler);
```

- stabileşte modul de acţiune la apariţia unui semnal
- semnale:

```
man 7 signal
```

- dacă semnalul *signum* poate fi livrat unui proces, atunci acesta poate să aleagă:
  - să ignore semnalul - SIG_IGN
  - să-l trateze în mod implicit - SIG_DFL
  - să specifice o funcţie care defineşte acţiunile care se execută la apariţia semnalului
- În terminologia Unix un proces care s-a terminat și pentru care procesul părinte nu a executat *wait* se numește ***zombie***. În această stare, procesul nu are resurse alocate, ci doar intrarea sa în tabela proceselor. Nucleul poate descărca toată memoria folosită de proces și închide fișierele deschise. Un proces zombie se poate observa prin comanda Unix *ps* care afișează la starea procesului litera 'Z'.
- prevenirea apariţiei proceselor de tip „***zombie***”:

```
signal(SIGCHLD, SIG_IGN)
```

- **exemplu:** fork_7.c

5. FUNCŢIA kill()

- sintaxa:

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


int kill(pid_t pid, int sig);
```

- permite livrarea unui semnal unui proces sau grup de procese
- semnificaţia valorilor argumentului pid:

pid	Semnificaţie
> 0	Semnalul este livrat procesului cu PID-ul specificat prin parametrul pid
0	Semnalul este livrat tuturor proceselor al căror identificator de grup (GID) este egal cu GID-ul procesului apelant
-1	Semnalul este livrat tuturor proceselor pentru care procesul apelant are dreptul de a livra semnale (cu excepţia procesului init)
< -1	Semnalul este livrat tuturor al căror identificator de grup (GID) este egal cu valoarea absolută a parametrului pid

- semnale:

```
man 7 signal
```

REFERINȚE:

- Tutorial IPC:

```
https://beej.us/guide/bgipc/html/single/bgipc.html
```