SISTEME DE OPERARE
– Laborator 7 –

PROCESE UNIX

1. PROCESE UNIX

- *proces* = *program aflat în execuţie*
- nucleul (*kernel*) sistemului de operare întreţine permanent o tabelă cu procese
- examinarea proceselor:

```
ps -e
ps -f -p 1
ps -F -u dbota
```

2. FUNCŢIA `fork()`

- prototip:

```
#include <unistd.h>

pid_t fork(void);
```

- crează un nou proces prin duplicarea procesului apelant (procesul părinte)
- noul proces este denumit proces copil şi este o copie aproape exactă a procesului părinte
- 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. FUNCŢIILE `wait()`, `waitpid()`

- prototipuri:

```
#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()`

- prototip:

```
#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
- prevenirea apariţiei proceselor de tip „zombie”:

  `signal(SIGCHLD, SIG_IGN)`
- exemplu: `fork_7.c`

5. FUNCŢIA `kill()`

- prototip:

```
#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	Semnificație
> 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 excepția 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
```

6. FAMILIA DE FUNCȚII exec()

- prototipuri:

```
#include <unistd.h>

int execl(const char *path, const char *arg, ...);
int execlp(const char *file, const char *arg, ...);
int execle(const char *path, const char *arg, ..., char *const envp[]);

int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);
int execve(const char *file, char *const argv[], char *const envp[]);
```

- lansează în execuție un nou program (fișier executabil)
- imaginea în memorie a procesului curent este înlocuită cu imaginea celui lansat în execuție
- exemple: exec_1.c, exec_2.c, exec_3.c

REFERINȚE:

- Curs: http://www.cs.ubbcluj.ro/~rares/course/os/
- Laborator: http://www.cs.ubbcluj.ro/~dbota/SO/