

## 3. Procese

### Linux

#### Operații cu procese

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

**command** comanda de executat  
**întoarce** 0 în caz de succes sau -1 în caz de eroare

```
pid_t fork(void);
```

**întoarce** 0 în copil, pid în părinte, -1 în caz de eroare

```
pid_t getpid(void);
```

**întoarce** PID-ul procesului apelant

```
pid_t getppid(void);
```

**întoarce** PID-ul părintelui procesului apelant

```
int execl(const char *path, const char *arg, ...);
```

**path** calea către program  
**arg** lista variabilă de parametri; numele primului parametru coincide cu cel al programului; ultimul parametru este NULL

**întoarce** doar în caz de eroare

```
int execvp(const char *file, char *const argv[], ...);
```

**file** numele executabilului; cautat în calea rețiuță de PATH  
**argv** vector parametri; numele primului parametru coincide cu cel al programului; ultimul parametru este NULL  
**întoarce** doar în caz de eroare (altfel nu se întoarce)

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

**pid** pid-ul procesului ce se dorește așteptat, 0 sau negativ pentru comportamente speciale  
**status** informația de terminare  
**options** diverse opțiuni, în general 0  
**întoarce** 0 în caz de succes sau -1 în caz de eroare

```
void exit(int status);
```

**status** codul de terminare a procesului

#### Variabile de mediu

```
int main(int argc, char **argv, char **environ);
```

**environ** vector de șiruri de forma VARIABILĂ = VALOARE

```
char* getenv(const char *name);
```

**name** numele variabilei  
**întoarce** valoarea variabilei sau NULL dacă nu există

```
int setenv(const char *name, const char *value, int replace)
```

**name** numele variabilei  
**value** valoarea variabilei  
**replace** 1 dacă variabila este deja definită și se dorește înlocuirea vechii valori

**întoarce** 0 în caz de succes sau -1 în caz de eroare

```
char* unsetenv(const char *name);
```

**name** numele variabilei  
**întoarce** 0 în caz de succes sau -1 în caz de eroare

#### Pipe-uri

```
int pipe(int filedes[2]);
```

**filedes** vector de descriptori ai capetelor pipe-ului cu 0 - citire, 1 - scriere

**întoarce** 0 în caz de succes sau -1 în caz de eroare

```
int mkfifo(const char *pathname, mode_t mode);
```

**pathname** calea din sistemul de fișiere  
**mode** permisiunile  
**întoarce** 0 în caz de succes sau -1 în caz de eroare

### Windows

#### Operații cu procese

```
BOOL CreateProcess( LPCTSTR lpApplicationName, LPTSTR  
lpCommandLine, LPSECURITY_ATTRIBUTES lpProcessAttributes,  
LPSECURITY_ATTRIBUTES lpThreadAttributes, BOOL  
bInheritHandles, DWORD dwCreationFlags, LPVOID  
lpEnvironment, LPCTSTR lpCurrentDirectory, LPSTARTUPINFO  
lpStartupInfo, LPPROCESS_INFORMATION lpProcessInformation );
```

- **lpApplicationName** - numele modulului de executat
- **lpCommandLine** - linia de comandă de executat
- **lpProcessAttributes** - atributele procesului
- **lpThreadAttributes** - atributele firului principal
- **bInheritHandles** - TRUE dacă se dorește moștenirea descriptorilor în procesele create
- **dwCreationFlags** - stabilește clasa de prioritate a procesului
- **lpEnvironment** - environment block-ul
- **lpCurrentDirectory** - directorul curent
- **lpStartupInfo** - atribute auxiliare, ex: redirectări
- **lpProcessInformation** - parametru out, populat de funcție cu diverse informații
- **întoarce** - nonzero pentru succes, zero în caz de eroare

```
DWORD WaitForSingleObject( HANDLE hHandle, DWORD  
dwMilliseconds );
```

- **hHandle** - handle-ul procesului ce se dorește așteptat
- **dwMilliseconds** - numărul maxim de milisecunde de așteptare, de obicei INFINITE
- **întoarce** - WAIT\_FAILED în caz de eroare

```
void ExitProcess( UINT uExitCode );
```

- **uExitCode** - codul de ieșire al procesului

```
BOOL TerminateProcess( HANDLE hProcess, UINT uExitCode );
```

- **hProcess** - handle-ul procesului ce se dorește terminat
- **uExitCode** - codul de ieșire al procesului
- **întoarce** - nonzero pentru succes, zero în caz de eroare

```
BOOL DuplicateHandle( HANDLE hSourceProcessHandle, HANDLE  
hSourceHandle, HANDLE hTargetProcessHandle, LPHANDLE  
lpTargetHandle, DWORD dwDesiredAccess, BOOL bInheritHandle,  
DWORD dwOptions );
```

- **hSourceProcessHandle** - handle-ul procesului proprietar al descriptorului ce se dorește duplicat
- **hSourceHandle** - descriptorul ce se dorește duplicat
- **hTargetProcessHandle** - handle-ul procesului doritor al handle-ului duplicat
- **lpTargetHandle** - handle-ul duplicat
- **dwDesiredAccess** - drepturile de acces
- **bInheritHandle** - TRUE dacă se dorește ca noul handle să poată fi moștenit mai departe
- **dwOptions** - opțiuni suplimentare
- **întoarce** - nonzero pentru succes, zero în caz de eroare

#### Variabile de mediu

```
LPTCH GetEnvironmentStrings(void);
```

- **întoarce** - un șir cu perechi VARIABILĂ = VALOARE

```
BOOL FreeEnvironmentStrings( LPTSTR lpszEnvironmentBlock );
```

- **lpszEnvironmentBlock** - pointer obținut prin GetEnvironmentStrings
- **întoarce** - nonzero pentru succes, zero în caz de eroare

```
DWORD GetEnvironmentVariable( LPCTSTR lpName, LPTSTR  
lpBuffer, DWORD nSize );
```

- **lpName** - numele variabilei
- **lpBuffer** - zona în care funcția va depune valoarea
- **nSize** - dimensiunea zonei de mai sus

- **întoarce** - numărul de caractere ale valorii(dacă zona este suficient de încăpătoare), numărul de caractere necesar(dacă zona nu este suficient de încăpătoare), zero în caz de eroare

```
BOOL SetEnvironmentVariable( LPCTSTR lpName, LPCTSTR lpValue);
```

- **lpName** - numele variabilei
- **lpBuffer** - noua valoare sau NULL dacă se dorește înlăturarea variabilei
- **întoarce** - nonzero pentru succes, zero în caz de eroare

## Pipe-uri

```
BOOL CreatePipe( PHANDLE hReadPipe, PHANDLE hWritePipe, LPSECURITY_ATTRIBUTES lpPipeAttributes, DWORD nSize );
```

- **hReadPipe** - descriptorul capătului de citire
- **hWritePipe** - descriptorul capătului de scriere
- **lpPipeAttributes** - determină posibilitatea moștenirii
- **nSize** - dimensiunea, în bytes, a buffer-ului intern
- **întoarce** - nonzero pentru succes, zero în caz de eroare

```
HANDLE CreateNamedPipe( LPCTSTR lpName, DWORD dwOpenMode, DWORD dwPipeMode, DWORD nMaxInstances, DWORD nOutBufferSize, DWORD nInBufferSize, DWORD nDefaultTimeOut, LPSECURITY_ATTRIBUTES lpSecurityAttributes );
```

- **lpName** - șir ce desemnează numele pipe-ului
- **dwOpenMode** - stabilește o serie de caracteristici, precum sensul simplu sau dublu de circulație a informației
- **dwPipeMode** - flux de octeti sau mesaj
- **nMaxInstances** - numărul maxim de instanțe
- **nOutBufferSize** - dimensiunea buffer-ului de ieșire
- **nInBufferSize** - dimensiunea buffer-ului de intrare
- **nDefaultTimeOut** - durata implicită de așteptare până ce o instanță a pipe-ului devine disponibilă
- **lpSecurityAttributes** - controlează posibilitatea de moștenire a handle-ului
- **întoarce** - handle-ul capătului dinspre server al pipe-ului, INVALID\_HANDLE\_VALUE în caz de eroare

```
BOOL ConnectNamedPipe( HANDLE hNamedPipe, LPOVERLAPPED lpOverlapped );
```

- **hNamedPipe** - handle-ul capătului dinspre server al pipe-ului
- **lpOverlapped** - oferă un mecanism de notificare la apariția unei noi cereri, când se lucrează în mod asincron

- **întoarce** - în mod sincron nonzero pentru succes, zero în caz de eroare; în mod asincron, funcția întoarce zero, iar starea operației este descrisă de GetLastError