3. Procese

Linux

Proces

Unitatea primitivă prin care sistemul de operare alocă resurse utilizatorilor.

Operaţii 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

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

`pid` pid-ul procesului ce se doreşte aşteptat, 0 sau negativ pentru comportamente speciale
`status` informaţia de terminare
`options` diverse opţiuni, 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 siruri 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 şi se doreşte 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` vectorul descriptorilor celor 2 capete al pipe-ului: 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 fişiere
`mode` permisiunile
întoarce 0 în caz de succes sau -1 în caz de eroare

Windows

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 doreşte moştenirea descriptorilor în procesele create
- `dwCreationFlags`, stabileste clasa de prioritate a procesului
- `lpEnvironment`, environment block-ul
- `lpCurrentDirectory`, directorul curent
- `lpStartupInfo`, atribute auxiliare, ex: redirectări
- `lpProcessInformation`, parametru out, populat de funcţie cu diferite informaţii
- *întoarce* nonzero pentru succes, zero în caz de eroare

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

- `hHandle`, handle-ul procesului ce se doreşte aşteptat
- `dwMilliseconds`, numărul maxim de miliscunde de aşteptare, 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 doreşte 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 doreşte duplicat
- `hSourceHandle`, descriptorul ce se doreşte duplicat
- `hTargetProcessHandle`, handle-ul procesului doritor al handle-ului duplicat
- `lpTargetHandle`, handle-ul duplicat
- `dwDesiredAccess`, drepturile de acces
- `bInheritHandle`, `TRUE` daca se doreşte ca noul handle să poată fi moştenit mai departe
- `dwOptions`, opţiuni suplimentare
- *întoarce* nonzero pentru succes, zero în caz de eroare

Variabile de mediu

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

- *întoarce* un şir cu perechi `VARIABILĂ = VALOARE`

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

- `lpszEnvironmentBlock`, pointer obţinut 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 funcţia 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 încăpătoarecaz, zero în caz de eroare

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

- `lpName`, numele variabilei
- `lpBuffer`, noua valoare sau NULL dacă se doreşte 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 moştenirii
- `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`, şir ce desemnează numele pipe-ului
- `dwOpenMode`, stabileşte o serie de caracteristici, precum sensul simplu sau dublu de circulaţie a informaţiei
- `dwPipeMode`, flux de octeţi sau mesaj
- `nMaxInstances`, numărul maxim de instanţe
- `nOutBufferSize`, dimensiunea buffer-ului de ieşire
- `nInBufferSize`, dimensiunea buffer-ului de intrare
- `nDefaultTimeOut`, durata implicită de aşteptare până ce o instanţă a pipe-ului devine disponibilă
- `lpSecurityAttributes`, controlează posibilitatea de moştenire 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 apariţia unei noi cereri, când se lucrează în mod asincron
- *întoarce* în mod sincronnonzero pentru succes, zero în caz de eroare; în mod asincron, funcţia întoarce zero, iar starea operaţiei este descrisă de `GetLastError`