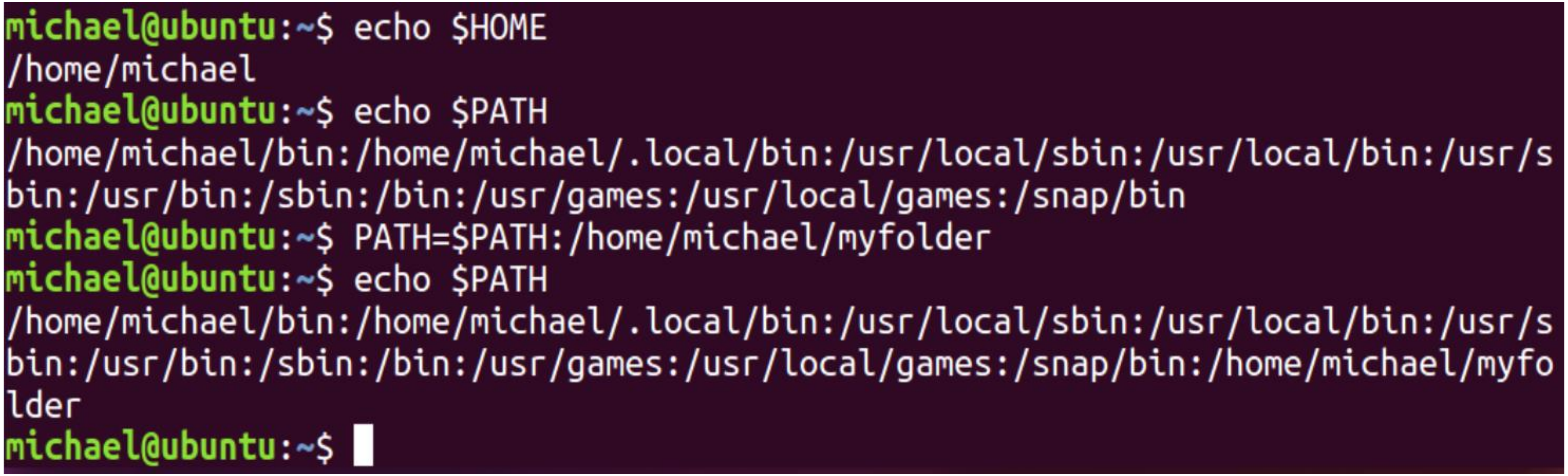
Vulnerabilitati specifice sistemelor de operare UNIX / Linux

CE LUNG TITLU DE PREZENTARE . . .

Variabile de mediu

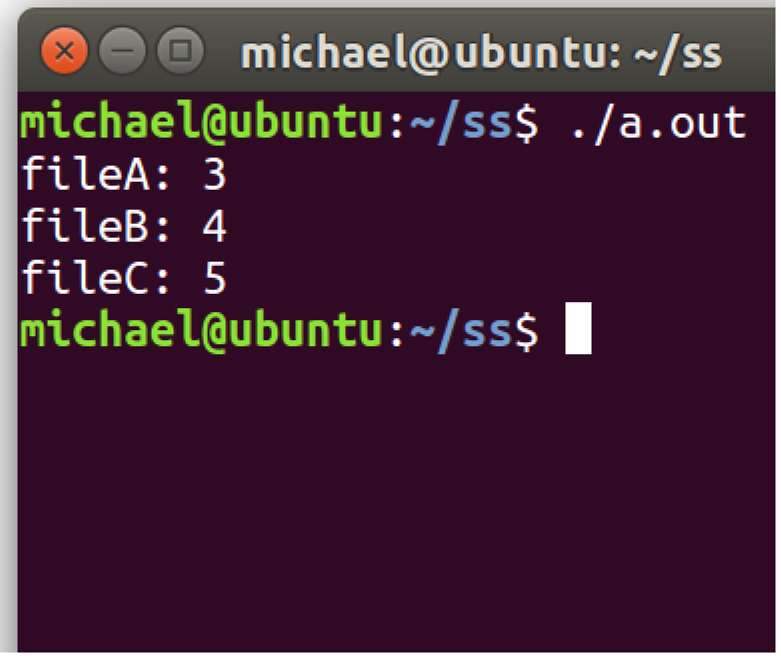


```
michael@ubuntu:~$ echo $HOME
/home/michael
michael@ubuntu:~$ echo $PATH
/home/michael/bin:/home/michael/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/s
bin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
michael@ubuntu:~$ PATH=$PATH:/home/michael/myfolder
michael@ubuntu:~$ echo $PATH
/home/michael/bin:/home/michael/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/s
bin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/home/michael/myfo
lder
michael@ubuntu:~$ 
```

File Descriptor Tables

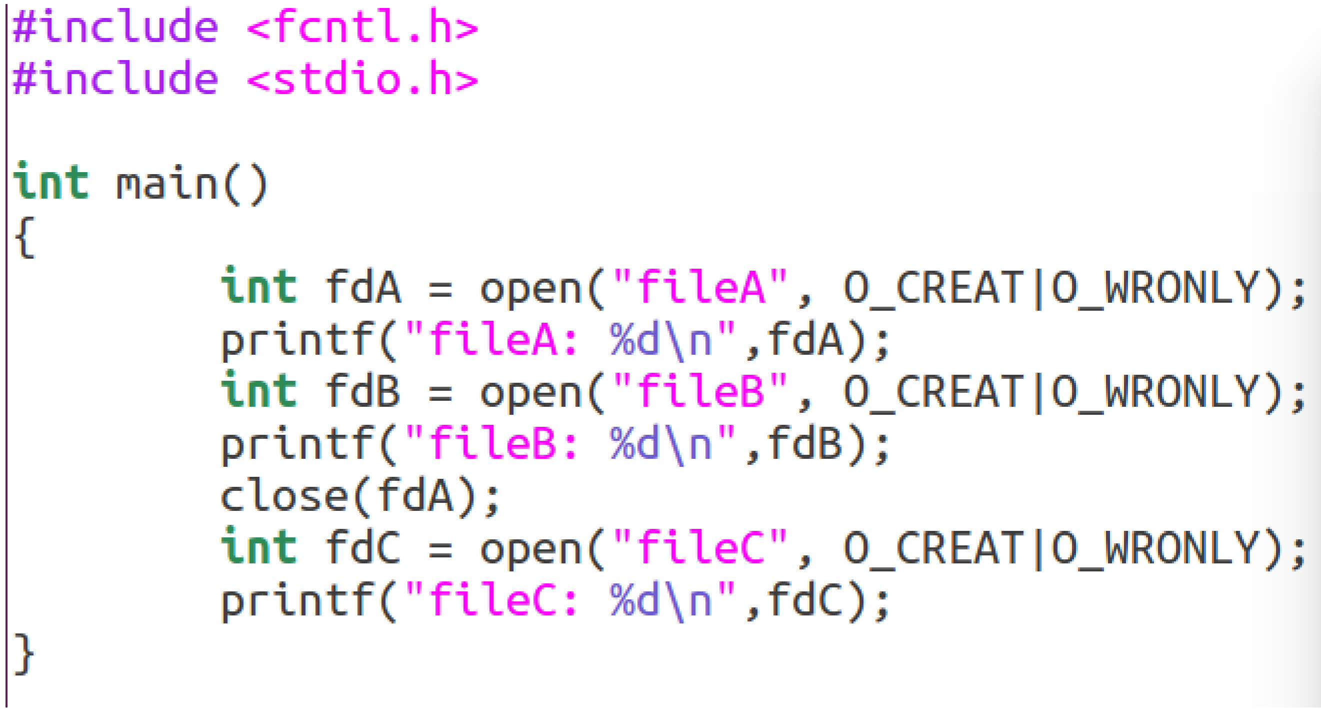
```c
#include <fcntl.h>
#include <stdio.h>

int main()
{
        int fdA = open("fileA", O_CREAT|O_WRONLY);
        printf("fileA: %d\n",fdA);
        int fdB = open("fileB", O_CREAT|O_WRONLY);
        printf("fileB: %d\n",fdB);
//      close(fdA);
        int fdC = open("fileC", O_CREAT|O_WRONLY);
        printf("fileC: %d\n",fdC);
}
```



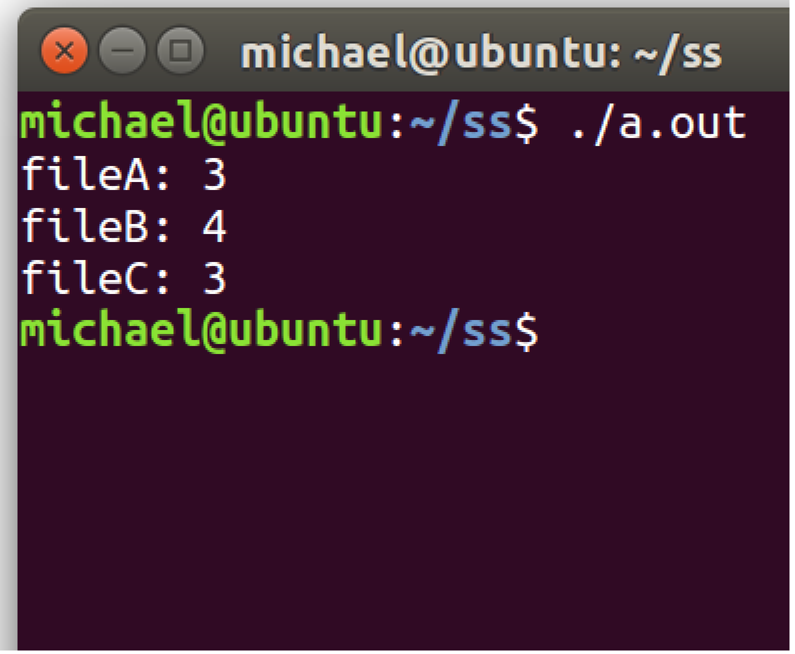
```
michael@ubuntu:~/ss$ ./a.out
fileA: 3
fileB: 4
fileC: 5
michael@ubuntu:~/ss$
```

File Descriptor Tables



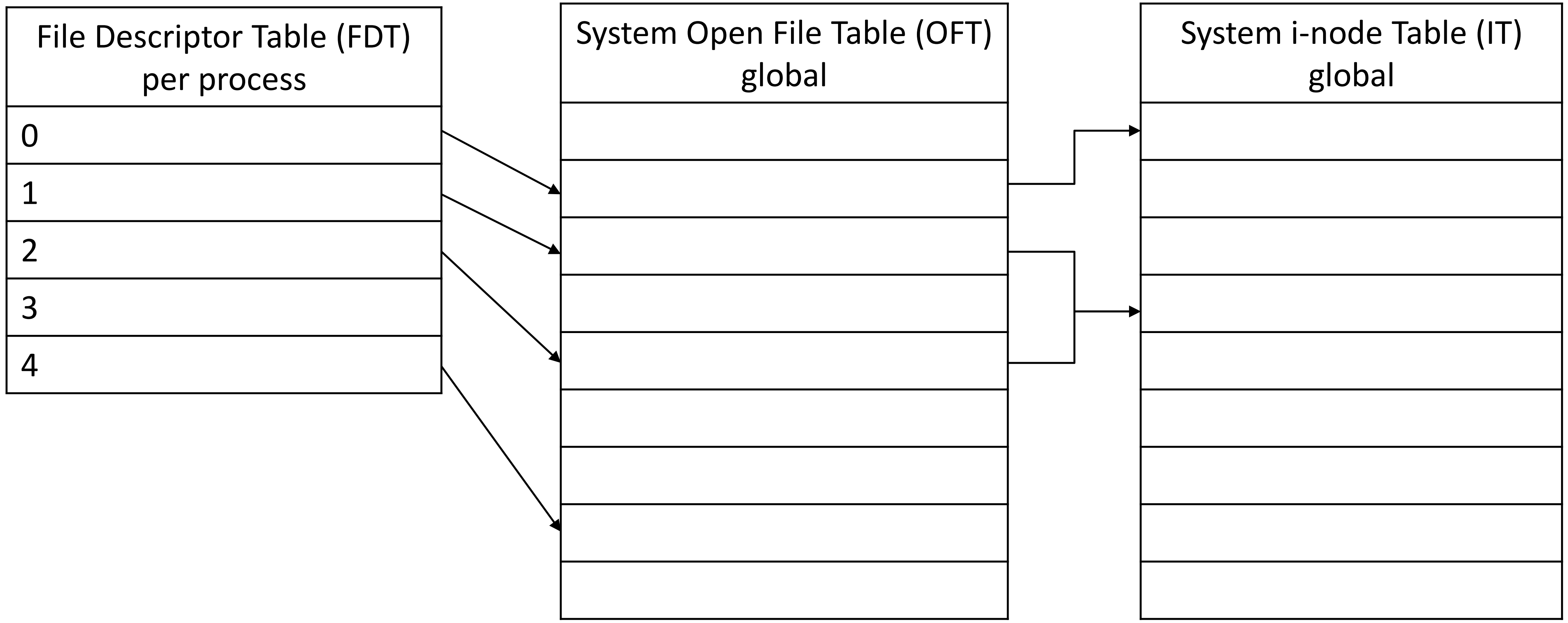
```c
#include <fcntl.h>
#include <stdio.h>

int main()
{
        int fdA = open("fileA", O_CREAT|O_WRONLY);
        printf("fileA: %d\n",fdA);
        int fdB = open("fileB", O_CREAT|O_WRONLY);
        printf("fileB: %d\n",fdB);
        close(fdA);
        int fdC = open("fileC", O_CREAT|O_WRONLY);
        printf("fileC: %d\n",fdC);
}
```

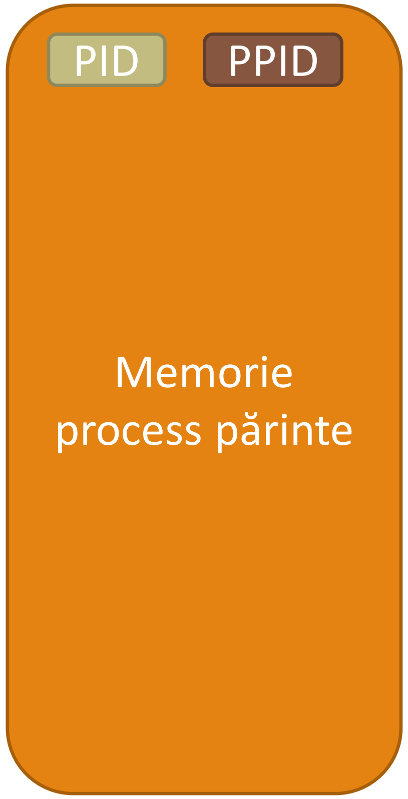


```
michael@ubuntu:~/ss$ ./a.out
fileA: 3
fileB: 4
fileC: 3
michael@ubuntu:~/ss$
```

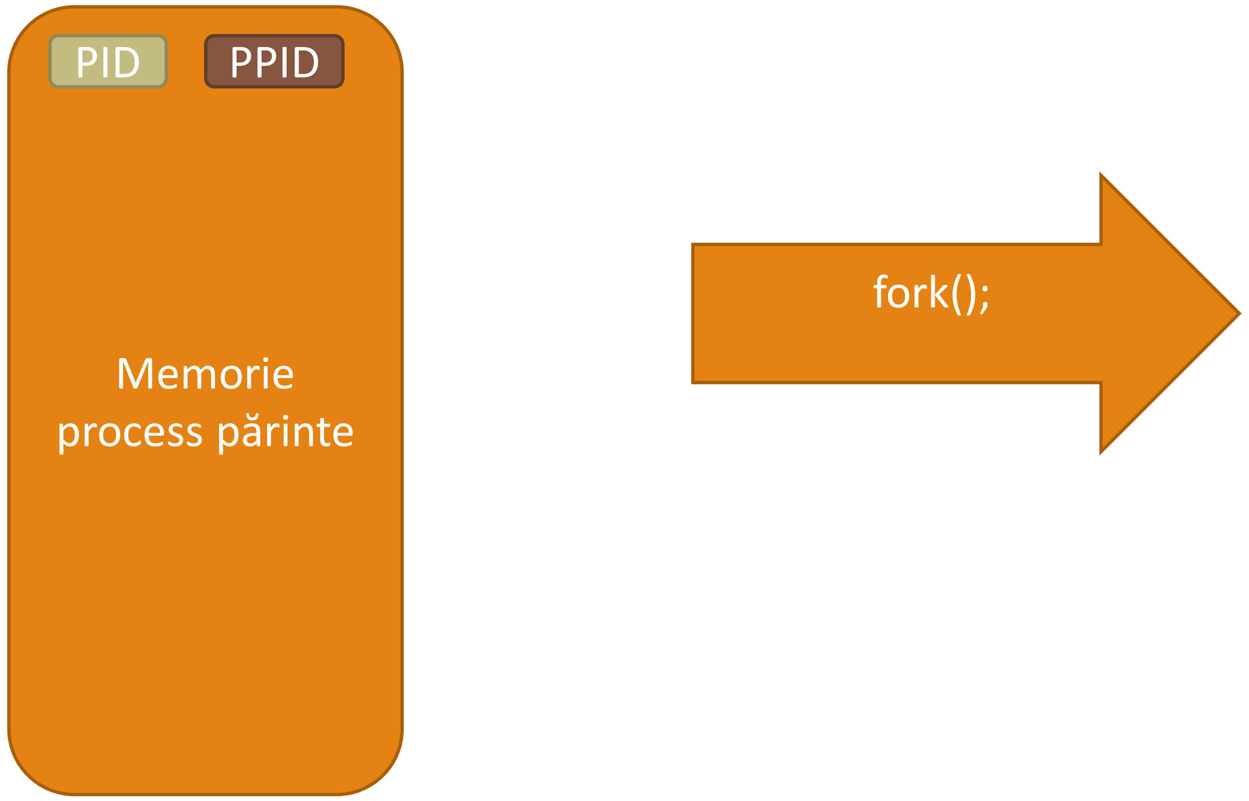
File Descriptor Tables



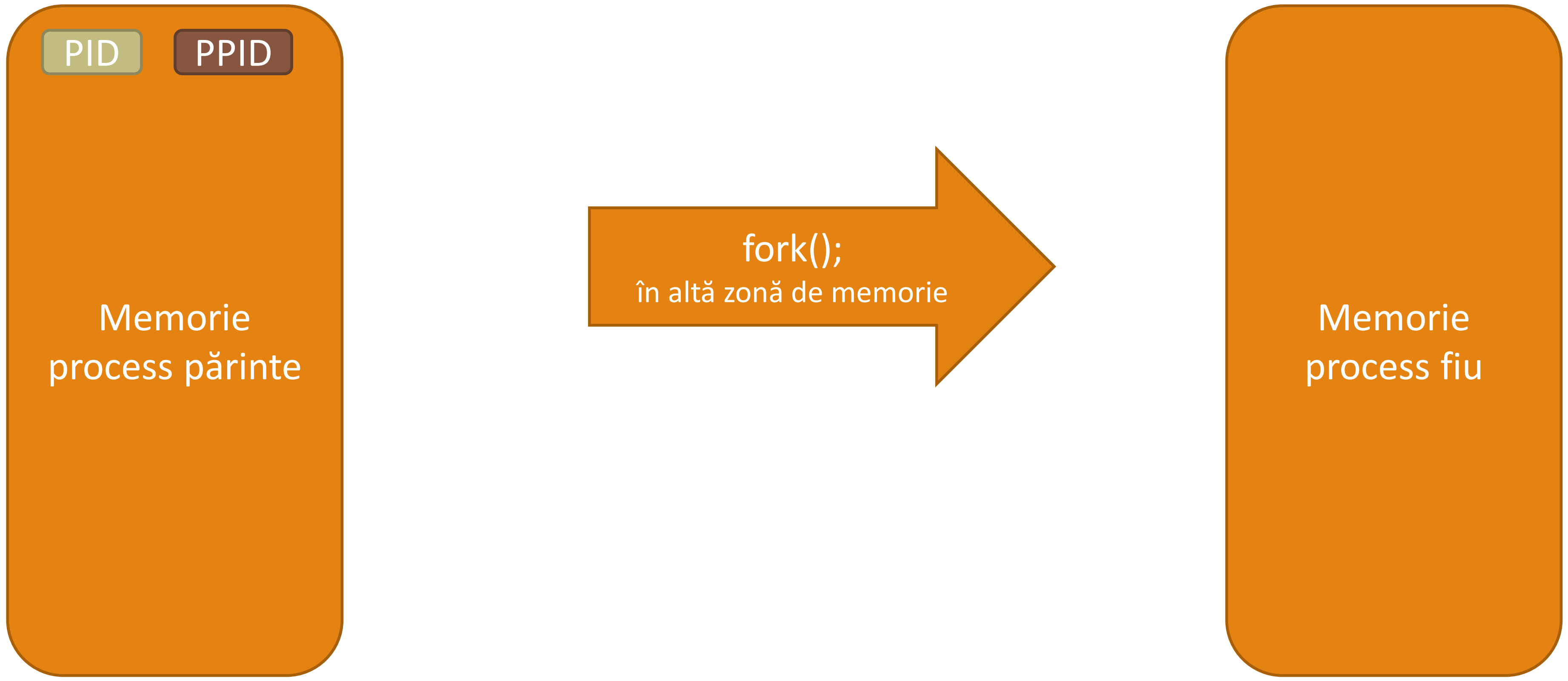
Fork



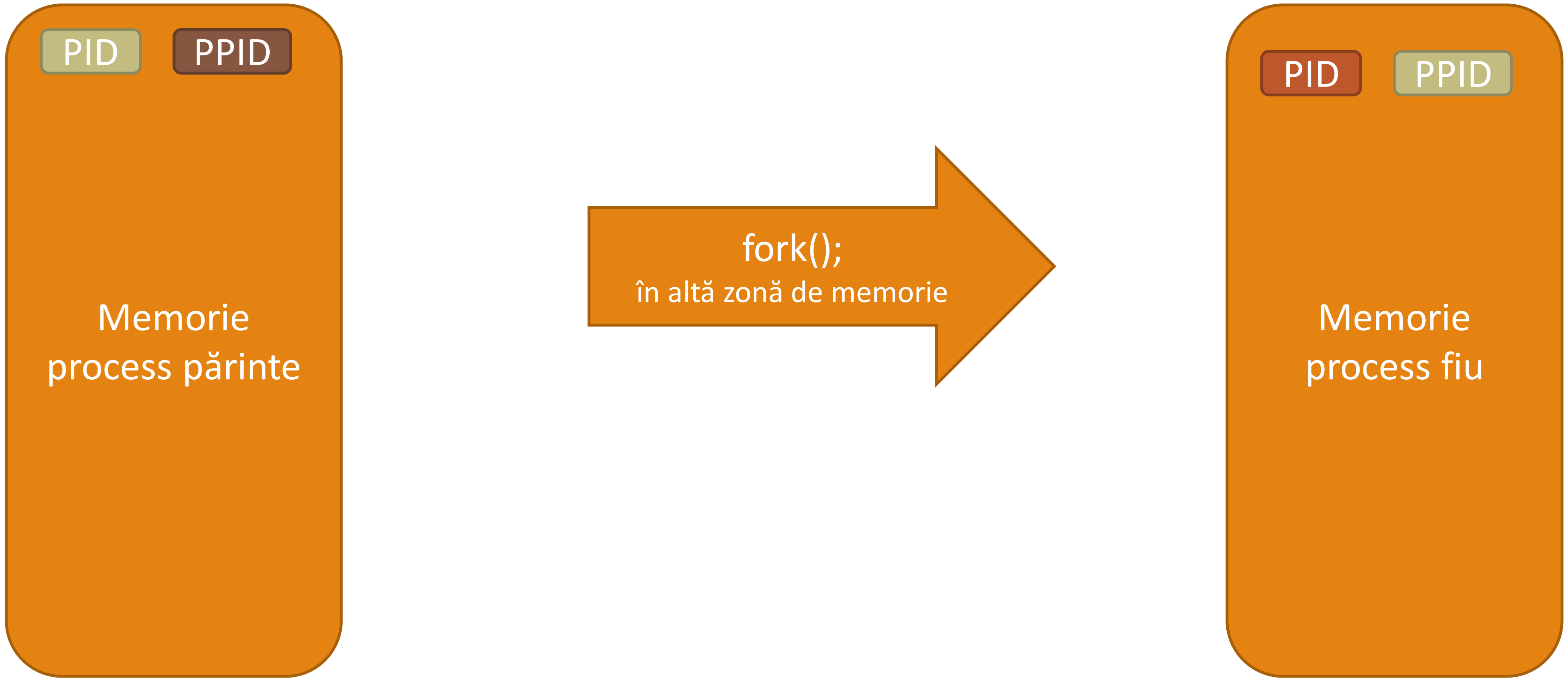
Fork



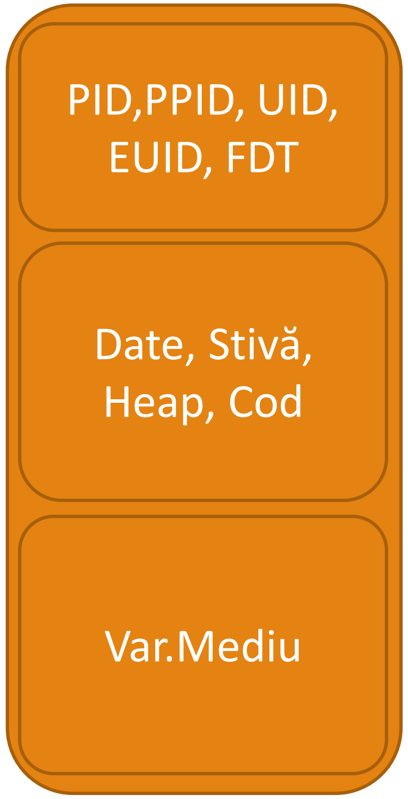
Fork


PID
PPID
Memorie
process părinte
fork();
în altă zonă de memorie
Memorie
process fiu

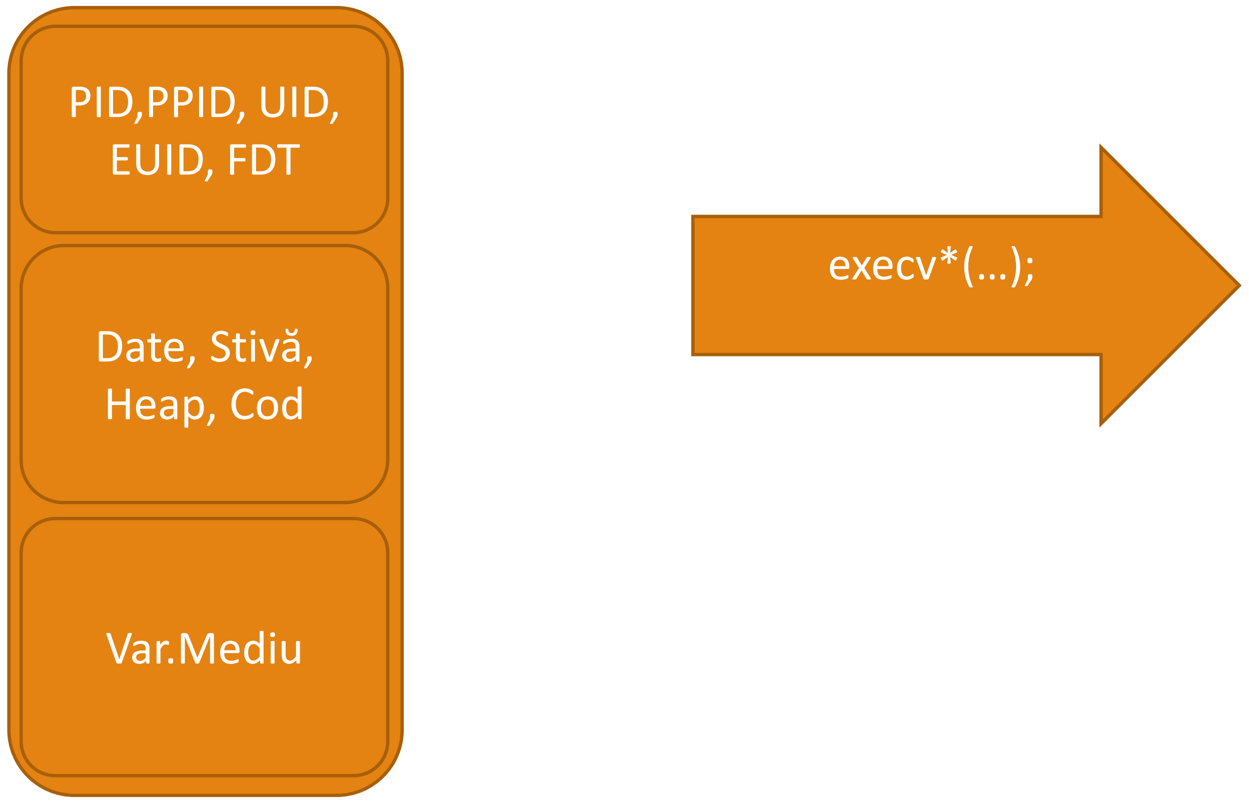
Fork


PID
PPID
Memorie
process părinte
fork();
în altă zonă de memorie
PID
PPID
Memorie
process fiu

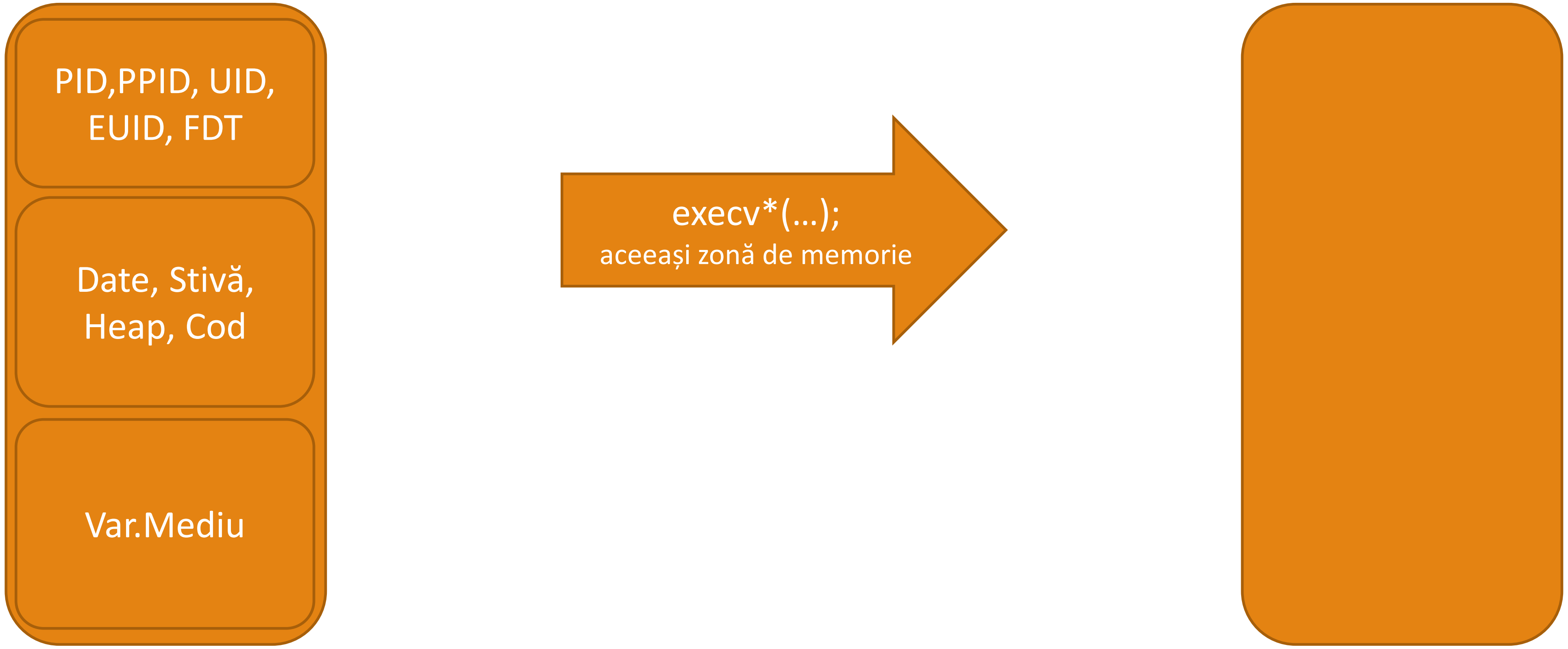
exec*



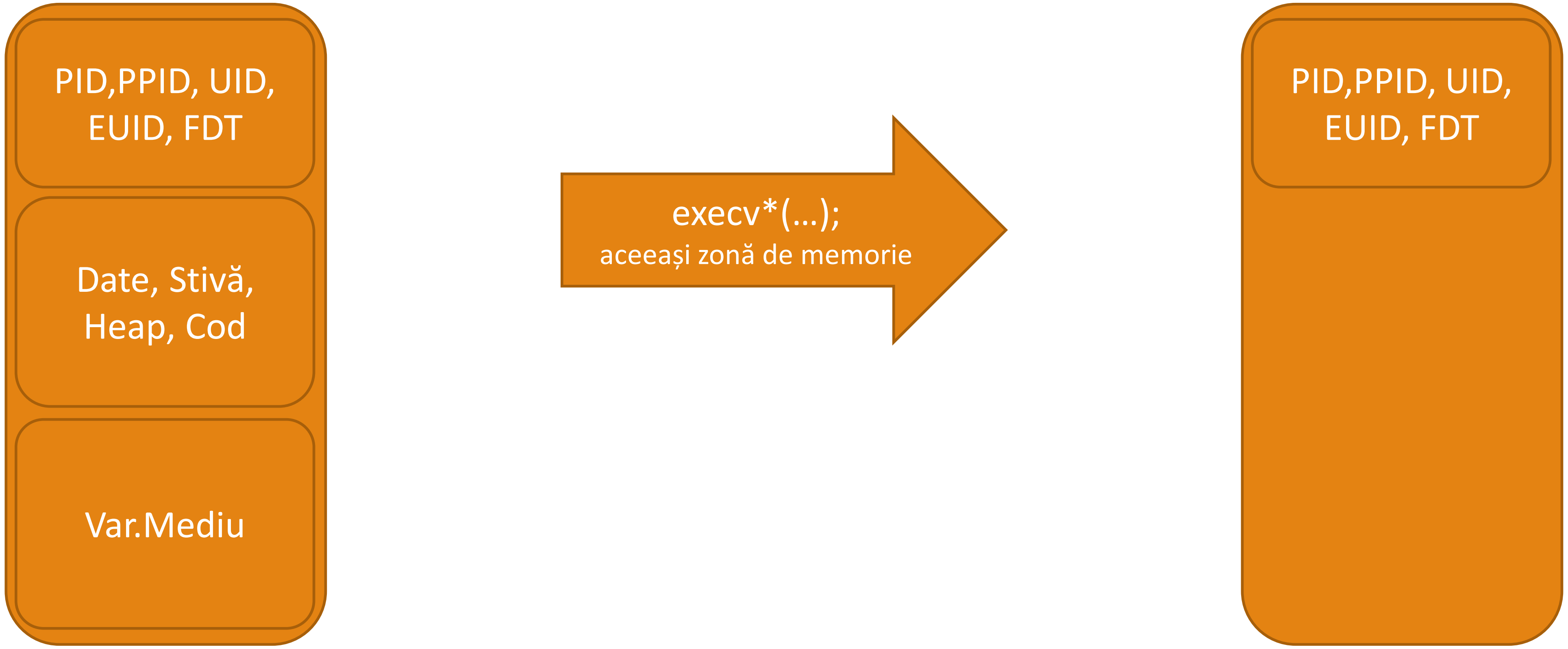
exec*



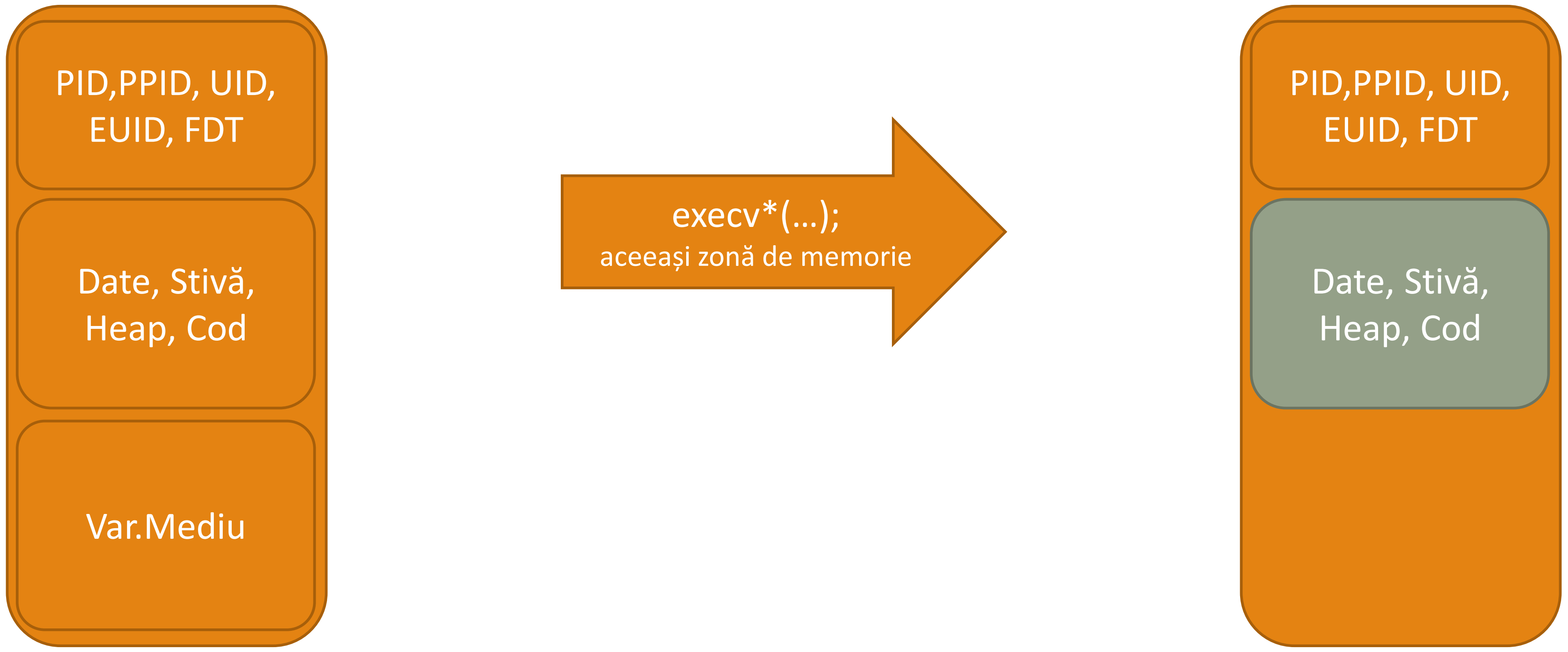
exec*


PID,PPID, UID, EUID, FDT
Date, Stivă, Heap, Cod
Var.Mediu
execv*(…);
aceeași zonă de memorie

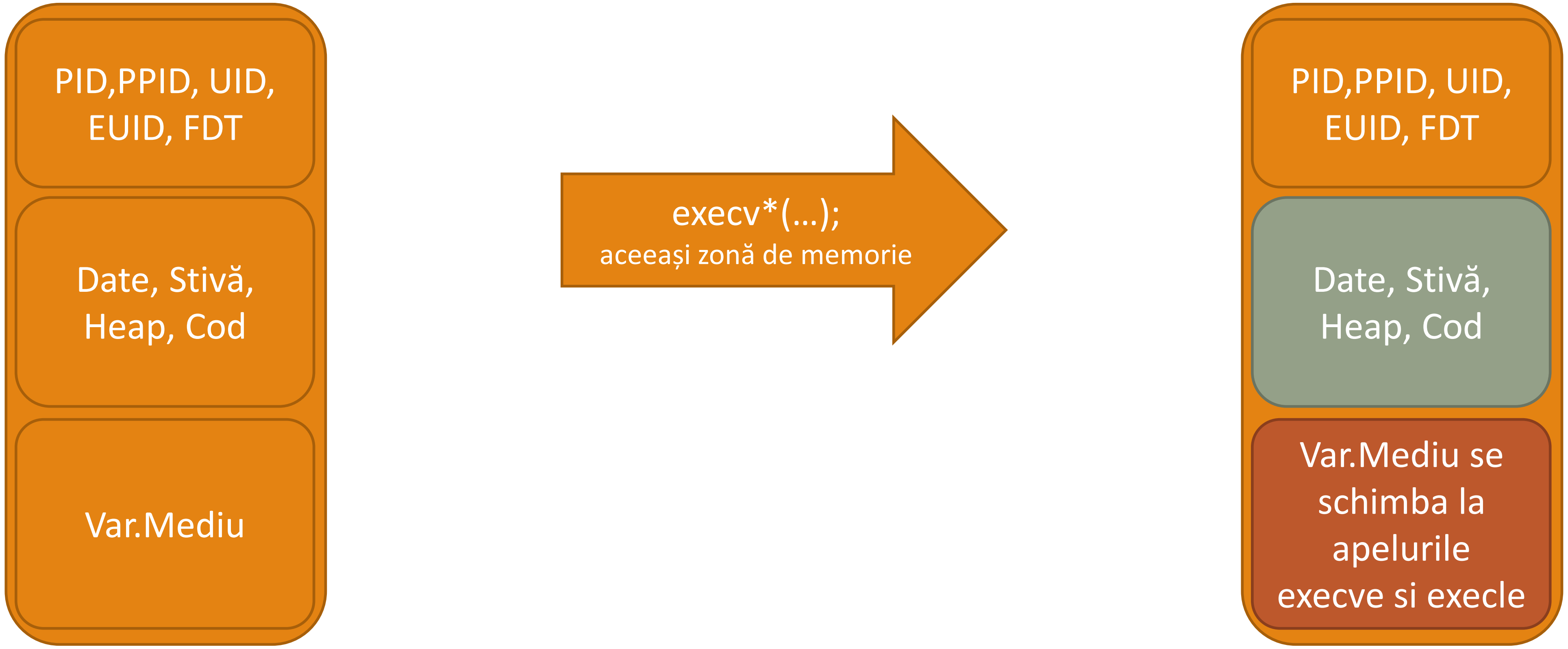
exec*


PID,PPID, UID, EUID, FDT
Date, Stivă, Heap, Cod
Var.Mediu
execv*(…);
aceeași zonă de memorie
PID,PPID, UID, EUID, FDT

exec*


PID,PPID, UID, EUID, FDT
Date, Stivă, Heap, Cod
Var.Mediu
execv*(…);
aceeași zonă de memorie
PID,PPID, UID, EUID, FDT
Date, Stivă, Heap, Cod

exec*


PID,PPID, UID, EUID, FDT
Date, Stivă, Heap, Cod
Var.Mediu
execv*(…);
aceeași zonă de memorie
PID,PPID, UID, EUID, FDT
Date, Stivă, Heap, Cod
Var.Mediu se schimba la apelurile execve si execle

Fork vs exec*

Atribut	Moștenit prin Fork	Rețínute la exec
PID	Nu	Da
UID	Da	Da
EUID	Da	Depinde de bitul “setuid”
Date	Copiate	Nu
Stivă	Copiată	Nu
Heap	Copiat	Nu
Text(cod)	Partajat	Nu
FDT (file descriptors)	Copiate	De regulă, da
Variabilele de mediu	Da	Depinde de tipul exec
Directorul current	Da	Da

Example

```
int print_directory_listing(char *path)
{
    char *argv[] = {"ls", "-l", path, NULL};
    int rc;

    rc = fork();

    if (rc < 0)
        return -1;

    if (rc == 0)
        execvp("ls", argv);
        return 0;
}
```

Resource limit (rlimits)

- getrlimit() and setrlimit()
- RLIMIT_CORE: maximum size for a core file
- RLIMIT_CPU: maximum CPU time (sec)
- RLIMIT_DATA: maximum size (bytes) for the data segment
- RLIMIT_FSIZE: maximum size of a written file
- RLIMIT_MEMLOCK: maximum no of bytes locked in memory
- RLIMIT_NOFILE: maximum number of open files
- RLIMIT_NPROC: maximum no of processes a user can run
- RLIMIT_STACK - maximum size (bytes) for process’ stack
- attack method: force a called privileged process to fail in a predetermined location

Example

```
struct entry {
    char name[32];
    char password[256];
    struct entry *next;
};

int write_entries(FILE *fp, struct entry *list)
{
    struct entry *ent;

    for (ent = list; ent; ent=ent->next)
      fprintf(fp, "%s:%s\n", ent->name, ent->password),
    return 1;
}
```

Attack vector

Attack vector

- Step 1: set a low RLIMIT_FSIZE

Attack vector

- Step 1: set a low RLIMIT_FSIZE
- Step 2: mask signal SIGXFSZ (to be ignored) before calling the privileged program

Attack vector

- Step 1: set a low RLIMIT_FSIZE
- Step 2: mask signal SIGXFSZ (to be ignored) before calling the privileged program
- Step 3: could cause partial writing, e.g. truncating a password