Lucrarea de laborator Nr. 03

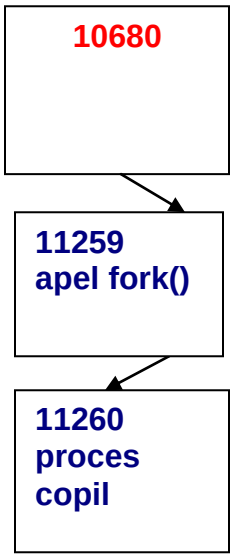
Procese. Mecanismul fork-exec. Executare de cod nou. Comunicarea inter-proces prin variabile de mediu

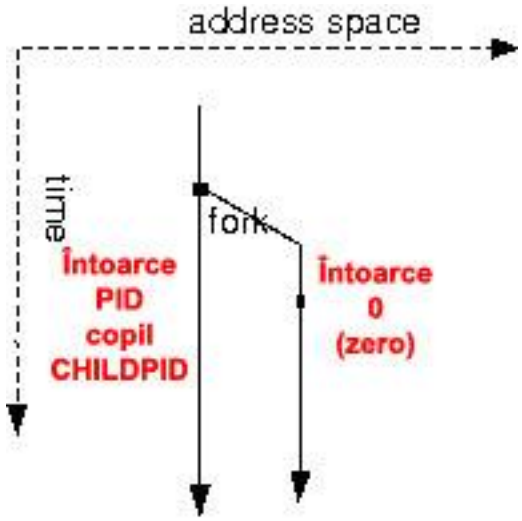
Aplicații demonstrative de executare cod nou prin mecanismul fork-exec:

1. **Utilizarea funcţiei execl():** Copiază un cod nou reprezentând comanda shell ps –f peste imaginea procesului care a făcut apelul și se execută noul cod. Se vizualizează PIP și PPID atât a procesului originar cât și a noii imagini create.
2. **Utilizarea funcţiei execvp():** Copiază un cod nou reprezentând comanda shell ps –f peste imaginea procesului care a făcut apelul și execută noul cod. Se vizualizează PIP și PPID atât a procesului originar cât și a noii imagini create.
3. **Utilizarea funcţiei execvp()** Copiază un cod nou reprezentând comanda shell compusă ps –f;ls –l;ps -f peste imaginea procesului care a făcut apelul și se execută noul cod. Se vizualizează PIP și PPID atât a procesului originar cât și a noii imagini, respectiv a noilor procese create.
4. **Utilizarea funcţiei execve():** Copiază un cod nou, creat anterior în urma unei complilări/linkeditări, peste imaginea procesului care a făcut apelul și execută noul cod cu trecerea argumentelor pe linia de comandă a noului cod și transmiterea unor variabile de mediu pentru noua imagine a procesului. Se vizualizează PIP și PPID atât a procesului originar cît și a noii imagini create, precum și două variabile de mediu care intervin.

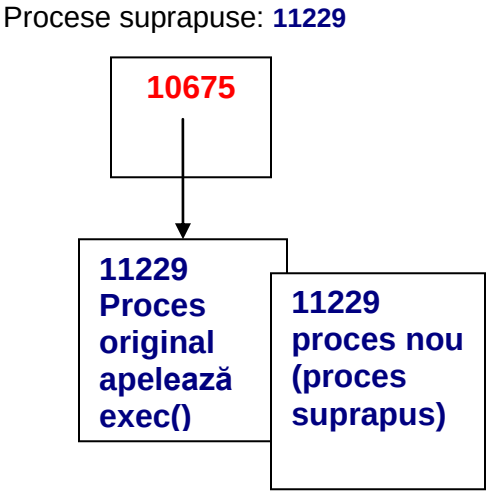
Executare de cod nou

Prin apelul funcţei *fork()* se crează un proces *copil* care execută o altă copie a procesului *părinte*. În fapt *părintele* şi *copilul* execută acelaşi cod dar în locaţii diferite de memorie.





Adeseori, însă, avem nevoie de executarea unui cod diferit în același spațiu de memorie. Acest lucru este posibil prin utilizarea unui apel sistem numit generic *exec* prin care codul *procesului apelant* (proces original) este înlocuit cu un cod executabil nou. Noul proces reacoperă procesul ce a executat apelul *exec* carel moşteneşte şi unele dintre caracteristicile acestuia: PID, descriptorii fișierelor care au fost deschise în procesul original (permite procesului original crearea de *pipe anonim* sau un *pip cu nume* – vezi L05 şi L06 pentru comunicare cu procesul nou startat prin exec()), etc.



Există 6 variante de apel sistem *exec* care se diferenţiază prin modul cum sunt pasate argumentele de pe linia de comandă sau din context (context: o mulţime de string-uri de forma variabila=valoare) precum şi cum a fost transmis *pathname* către executabil.

Observaţie: în caz de succes, funcţia *exec* nu returnează, fiind singurul exemplu de funcţie al carei apel nu returnează valoare spre programul apelant.

```
//----------- numărul argumentelor este cunoscut la compilare şi este fix
```

Apelurile *execl* (*execl(), execle()* şi *execlp()*)

Sunt utilizate când numărul argumentelor funcției precum și valoarea lor sunt cunoscute la compilarea programului.

```
//----------- numărul arumentelor este cunoscut în execuţie şi poate fi variabil
```

Apelurile *execv* (*execv(), execve()* şi *execvp()*)

Pasează argumentele funcției ca un argument şir (argument array), numărul argumentelor și valoarea lor nefiind cunoscute în momentul compilării programului.

Un apel sistem *exec* copiază un cod executabil nou peste imaginea procesului care a făcut apelul și execută noul cod. Ceea ce se păstrează de la *procesul* original nu este tocmai evident. Astfel codul programului, variabilele, stiva şi heap-ul sunt suprascrise. Noul *proces* moşteneşte variabilele de mediu (contextul), mai puţin în cazul când *procesu*l original apelează *execle()* sau *execve()*. Fişierele care sunt deschise înaintea unui apel sistem *exec* în mod normal rămân deschise şi după apel, lucru exploatat în comunicarea inter-proces de tip *pipe anonim sau cu nume* (vezi Lab. 05 şi Lab. 06). Pot să apară efecte variate în cazul unor *blocări* sau *alarme.* Programarea trebuie făcută cu mare grijă.

Cele 6 variante de apel funcții ***exec*** sunt detaliate mai jos (cu roşu sunt cele exemplificate în programele din lucrare):

//----------

NOTĂ: Semnificație parametri formali din sintaxa funcțiilor *exec*

file *executabilul* este căutat în directorele definite în variabila de mediu PATH

```
ex. "ls"          utilizare validă pentru comada ls (caută în $PATH)
    "/bin/ls"     utilizare validă
    "./fisexec"   utilizare validă pentru fisexec în directorul curent
    "fisexec"     utilizare invalidă (fişier negăsit) pentru fisexec în directorul
                                                                    curent
```

path *executabilul* este căutat numai în locaţia indicată

```
ex. "/bin/ls"     utilizare validă pentru comanda ls
    "ls"          utilizare invalidă (fișer negăsit, nu caută în $PATH)
    "fisexec"     utilizare validă, pentru fisexec în directorul curent
    "./fisexec"   utilizare validă, pentru fisexec în directorul curent
```

//---------- numărul argumentelor este cunoscut la compilare şi este fix

int execl(const char *path, const char *arg0, ..., const char *argn, char * /*NULL*/);

Exemplu de utilizare execl()

Execută comanda *ls* specificând calea absolută a executabilului (**/bin/ls**) cu utilizarea unui argument (*-1*) al comenzii care produce în ieșire o singură coloană. Comanda *which -a ls* afișează locația comenzii *ls* utilizată.

```
#include <unistd.h>

int ret;
...
ret = execl ("/bin/ls", "ls", "-1", (char *)0);
```

SAU

```
ret = execl ("/bin/ls", "ls", "-1", NULL);
```

int execle (const char *path, const char *arg0, ... , const char *argn, char * /*NULL*/, char *const envp[]);

Exemplu de utilizare execle()

Este similar cu ***Exemplul de utilizare*** *execl()*. În plus, se specifică variabilele de mediu pentru nouă imagine proces utilizând argumentul *env*. Comanda *which -a ls* afișează locația comenzii *ls* utilizată.

```
#include <unistd.h>

int ret;
char *env[] = { "HOME=/usr/home", "LOGNAME=home", (char *)0 };
```

SAU

```
char *env[] = { "HOME=/usr/home", "LOGNAME=home", NULL };
...
ret = execle ("/bin/ls", "ls", "-l", (char *)0, env);
```

SAU

```
ret = execle ("/bin/ls", "ls", "-l", (NULL, env);
```

int execlp(const char *file, const char *arg0, ... , const char *argn, char * /*NULL*/);

Exemplu de utilizare execlp()

Acest exemplu caută locația comenzii *ls* conform setării directoarelor de căutare specificate în variabila de mediu *PATH*. Comanda *echo $PATH* afișează conținutul variabilei de mediu $*PATH.*

```
#include <unistd.h>

int ret;
...
ret = execlp ("ls", "ls", "-l", (char *)0);
```

SAU

```
ret = execlp ("ls", "ls", "-NULL);
```

//---------- numărul arumentelor e cunoscut în execuţie şi poate fi variabil

int execv(const char *path, char *const argv[]);

Exemplu de utilizare execv()

Argumentele comenzii *ls* sunt pasate prin intermediul șirului *cmd.* Comanda *which -a ls* afișează locația comenzii *ls* utilizată.

```
#include <unistd.h>

int ret;
char *cmd[] = { "ls", "-l", (char *)0 };
```

SAU

```
char *cmd[] = { "ls", "-l", NULL };
...
ret = execv ("/bin/ls", cmd);
```

int execve (const char *path, char *const argv[], char *const envp[]);

Exemplu de utilizare execve()

Argumentele comenzii *ls* sunt pasate prin intermediul șirului *cmd.* În plus, se specifică variabilele de mediu pentru nouă imagine proces utilizând argumentul *env.* Comanda *which -a ls* afișează locația comenzii *ls* utilizată.

```
#include <unistd.h>

int ret;
char *cmd[] = { "ls", "-l", (char *)0 };
```

SAU

```
char *cmd[] = { "ls", "-l", NULL };

char *env[] = { "HOME=/usr/home", "LOGNAME=home", (char *)0 };
```

SAU

```
char *env[] = { "HOME=/usr/home", "LOGNAME=home", NULL };
...
ret = execve ("/bin/ls", cmd, env);
```

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

Exemplu de utilizare execvp()

Acest exemplu caută locația comenzii *ls* conform setării directoarelor de căutare specificate în variabila de mediu *PATH*. Argumentele comenzii *ls* sunt pasate prin intermediul șirului *cmd.* Comanda *echo $PATH* afișează conținutul variabilei de mediu *PATH.*

```
#include <unistd.h>

int ret;
char *cmd[] = { "ls", "-l", (char *)0 };
```

SAU

```
char *cmd[] = { "ls", "-l", NULL };
...
ret = execvp ("ls", cmd);
```

Aplicații demonstrative de executare cod nou prin mecanismul fork-exec:

1. **Utilizarea funcţiei execl():** Copiază un cod nou reprezentând comanda shell *ps –f* peste imaginea procesului care a făcut apelul și se execută noul cod. Se vizualizează PIP și PPID atât a procesului originar cât și a noii imagini create.

- **Program execl_.c – Utilizarea funcţiei *execl()* pentru executarea comenzii shell *ps -f***

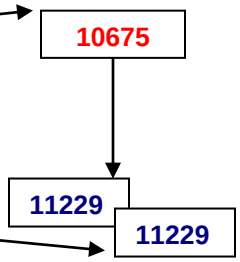
```
//  FISIER: exec_.c
    // Utilizarea functiei execl() in executarea comenzii shell ps -f
    #include <sys/types.h>
    #include <unistd.h>
    #include <stdio.h>
    #include <stdlib.h>

    int main()
    {
        /* secventa proces  apelant  */
        printf("PROCES APELANT:: proces ID=%ld, parinte ID= %ld\n", (long)getpid(), (long)getppid());
        // se starteaza un nou proces, ps –f, peste procesul apelant
        //    se va constata ca PID si PPID sunt pastrate
        execl("/bin/ps","se ignora","-f",NULL);
        perror("Eroare: Aici NU se revine...functia exec pentru comanda shell ps –f a esuat");
        return 0;
    }
```

Compilare linkeditare şi rulare program exec_.c

```
$gcc -o execl_ execl_.c
$./execl_
```

Mesaje afişate în urma rulării

```
PROCES APELANT:: proces ID=11229, parinte ID= 10675
UID        PID  PPID  C STIME TTY          TIME CMD
streian  10675 10584  0 16:48 pts/0    00:00:00 -bash
streian  11229 10675  0 16:55 pts/0    00:00:00 se ignora -f
__
```



2. **Utilizarea funcţiei execvp():** Copiază un cod nou reprezentând comanda shell *ps –f* peste imaginea procesului care a făcut apelul și execută noul cod. Se vizualizează PIP și PPID atât a procesului originar cât și a noii imagini create.

- **Program execvp_1.c – Utilizarea funcţiei *execvp()* pentru executarea comenzii shell *ps -f***

```
//  FISIER: execvp_1.c
    // Utilizarea functiei execvp() in executarea comenzii shell ps -f
    #include <sys/types.h>
    #include <unistd.h>
    #include <stdio.h>
    #include <stdlib.h>

    int main()
    {
        /* secventa proces  apelant  */
        char *const parmList[] = {"se ignora", "-f", NULL};
        printf("PROCES APELANT:: proces ID=%ld, parinte ID= %ld\n", (long)getpid(),(long)getppid());
        // se starteaza un nou proces, ps –f, peste procesul apelant
        //    se va constata ca PID si PPID sunt pastrate
        execvp("ps",&parmList[0]);  // paramList poate fi completata si dinamic,
                                    // in executie
        perror("Eroare: Aici NU se revine...functia exec pentru comanda shell ps –f a esuat");
        return 0;
    }
```

Compilare linkeditare şi rulare program execvp_1.c

```
$gcc -o execvp_1.c

$./ execvp_1
```

Mesaje afişate în urma rulării

```
 PROCES APELANT:: proces ID=3757, parinte ID= 3743
UID        PID  PPID  C STIME TTY          TIME CMD
streian   3743 3739  0 14:28 pts/1    00:00:00 -bash
streian   3757 3743  0 14:28 pts/1    00:00:00 se ignora -f
—
```

3. Utilizarea funcţiei execvp() Se invoca ca parametru o comandă shell compusă: ***ps -f;ps -F;ps -C ps -f*** . execvp() copiază un cod nou reprezentând numai comanda shell *ps –f* peste imaginea procesului care a făcut apelul și se execută noul cod, precum și separat, fără suprapunere de cod, comenzile shell *ps -F;ps -C ps -f.* Se vizualizează PIP și PPID atât a procesului originar cât și a noii imagini, respectiv a noilor procese create.

- **Program execvp_2.c – Utilizarea funcţiei *execvp()* pentru executarea comenzii shell compuse *ps -f;ps -F;ps -C ps -f***

Utilizarea functiei *execvp()* în executarea comenzii compuse shell *ps –f;ls –l;ps* introdusă pe linia de comandă.
Procesul originar cheama *execvp()* având ca argument un *pointer array* format de argumentele liniei de comandă a programului originar.

// Fisier: execvp_2.c

```
// Utilizarea functiei execvp() in executarea comenzii compuse ”ps –f;ps –F;ps -C ps -f” introdusa pe linia de comanda
// Procesul originar cheama execvp() avand ca argument un pointer array format de argumentele liniei de comanda
// a programului originar (ex. ps -f).

#include <unistd.h>
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>    // pt. exit()

int main(int argc, char *argv[])          // sau int main(int argc, char **argv)
{
 int i;
 /* secventa proces  originar  */
 printf("\nNumar total argumente = %d\n", argc);
 printf("\nCitirea  argumentelor\n");
 for (i=0; i<argc; i++)
 {            // afisare argumente
  printf("\targument[%d] = %s\n", i, argv[i]);
 };

 printf("\n\nSunt procesul originar si ma voi inlocui cu procesul %s %s , \n\tam ProcessID = %ld, ParentPID=%ld\n\n",
    argv[1],argv[2], (long) getpid(), (long) getppid());
 // argv este completat dinamic in executie de pe linia de comanda
 // ( execvp v vine de la vector, iar p vine de la pointer)
 if (execvp(argv[1], &argv[1]) < 0) //  se va constata ca PID si PPID proces creat este pastrat
   /* aici se ajunge numai daca eroare execvp */
   perror("functia execvp esuata \n");
  exit(-1);
}
```

Compilare linkeditare şi rulare program *execvp_2.c*

```
$ gcc -o execvp_2 execvp_2.c

$ ./execvp_2 ps -f;ps -F;ps -C ps -f
```

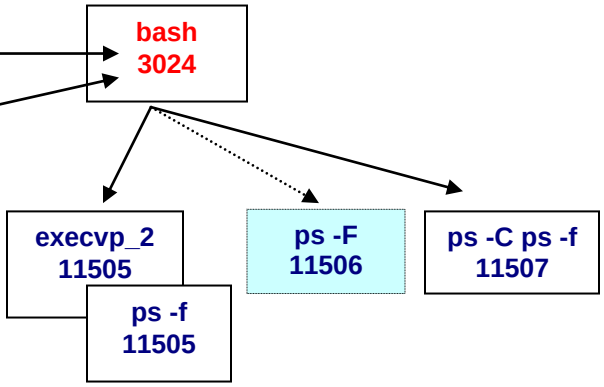
Mesaje afişate în urma rulării

Numar total argumente = 3

Citirea argumentelor
argument[0] = ./execvp_2
argument[1] = ps
argument[2] = -f

Sunt procesul originar si ma voi inlocui cu procesul ps -f ,
am ProcessID = **11505**, ParentPID=**3024**

UID **PID PPID** C STIME TTY TIME CMD
knoppix **3024** 3022 0 08:26 pts/0 00:00:00 bash
knoppix **11505** **3024** 0 11:52 pts/0 00:00:00 ps -f
UID **PID PPID** C SZ RSS PSR STIME TTY TIME CMD
knoppix **3024** 3022 0 2101 4560 0 08:26 pts/0 00:00:00 bash
knoppix **11506** **3024** 0 2293 3120 1 11:52 pts/0 00:00:00 ps -F
UID **PID PPID** C STIME TTY TIME CMD
knoppix **11507** **3024** 0 11:52 pts/0 00:00:00 ps -C ps -f



4. **Utilizarea funcţiei execve():** Copiază un cod nou, creat anterior în urma unei compillări/linkeditări, peste imaginea procesului care a făcut apelul și execută noul cod cu trecerea argumentelor pe linia de comandă a noului cod și transmiterea unor variabile de mediu pentru noua imagine a procesului. Se vizualizează PIP și PPID atât a procesului originar cît și a noii imagini create, precum și două variabile de mediu care intervin. Programele **execve_1.c** și **execve_2.c**

- **Program execve_1.c –** Afișează două variabile de mediu, precum și argumentele de pe linia de comandă. Aceste elemente vor fi transmise de către procesul original, dar pot fi afișate și independent în scop de testare.

```
/* Fisier: execve_1.c */
    //  Afiseaza argumentele introduse pe linia de comanda
    //  Acest program va fi startat de catre programul execve_2
    //   fiind indicat ca si parametru pe linia  pe linia de
    //     comanda a acestuia, incluzand si argumentele care se
    //      afiseaza. Se transmite prin execve_2 si varibila de mediu VARM

    #include <stdio.h>
    #include <stdlib.h>
    ///#include <errno.h>
    #include <sys/types.h>
    #include <unistd.h>

    int main(int argc, char *argv[])
    {
            int j;
            printf("\texecve_1 - \t\tProcessID = %ld, ParentPID=%ld\n", (pid_t) getpid(), (long) getppid());
            // afisare variabile de mediu
            printf("\texecve_1 - PATH : %s\n", getenv("PATH"));  // aceasta variabila de mediu NU e mostenita
            printf("\texecve_1 - VARM : %s\n", getenv("VARM")); // aceasta variabila de mediu este transmisa
            // afisare argumente de pe linia de comanda
            for (j = 0; j < argc; j++) printf("\texecve_1 - argv[%d]: %s\n", j, argv[j]);
            exit(EXIT_SUCCESS);
    }
```

Compilare linkeditare şi rulare program *execve_1.c*

```
$gcc -o execve_1 execve_1.c

$ ./execve_1 aaaa bbbb cccc dddd
```

Mesaje afişate în urma rulării

```
execve_1 -              ProcessID = 3505, ParentPID=3306    --> 3306 reprezintă shell
execve_1 - PATH : /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
execve_1 - VARM : (null)
execve_1 - argv[0]: ./execve_1
execve_1 - argv[1]: aaaa
execve_1 - argv[2]: bbbb
execve_1 - argv[3]: cccc
execve_1 - argv[4]: dddd
```

- **Program execve_2.c – Utilizarea funcţiei *execve()* pentru startarea unui program (*execve_1.c*) cu transmiterea acestuia de argumente pe linia de comandă și a două variabile de mediu.**

```
/* Fisier: execve_2.c */
    // Utilizarea functiei execve()  pentru startarea unui program (execve_1)
    // Programul startat este indicat ca argument pe linia de comanda
    // Programuli startat i se furnizeaza argumente pe linia de comanda
    // si i se transmit doua variabile de mediu
    #include <sys/types.h>
    #include <unistd.h>
    #include <stdio.h>
    #include <errno.h>
    #include <stdlib.h>    // pt. exit()
    int main(int argc, char *argv[])
            // sau int main(int argc, char **argv)
    {
            char *argv_1[] =                    // argumente pe linia de comanda a programului startat
            {
                    NULL, "salut", "pe", "toata", "lumea", NULL
            };
            char *vmediu[] = { "PATH=AM INLOCUIT-O", "VARM=VARIABILA DE MEDIU TRANSMISA", NULL };
             if (argc != 2)
            {
                    fprintf(stderr, "Utilizare: %s <file-to-exec>\n", argv[0]);
                    exit(EXIT_FAILURE);
             };
             printf("\nexecve_2 (proces originar), \tProcessID = %ld, ParentPID=%ld\n", (long) getpid(), (pid_t)
    getppid());
            argv_1[0] = argv[1];
            //Noul proces moşteneşte variabilele de mediu (contextul), mai puţin în cazul
             //        când procesul original apelează execle() sau execve().
            execve(argv[1], argv_1, vmediu); //      se va constata ca PID si PPID sunt pastrate
             /* aici se ajunge numai daca eroare execve */
            perror("functia execve esuata \n");
            exit(EXIT_FAILURE);
    };
```

Compilare linkeditare şi rulare program *execve_2.c*

```
$ gcc -o execve_2 execve_2.c
```

$ ./execve_2 ./execve_1

Mesaje afişate în urma rulării

```
execve_2 (proces originar),      ProcessID = 3532, ParentPID=3306
        execve_1 -               ProcessID = 3532, ParentPID=3306
        execve_1 - PATH : AM INLOCUIT-O
        execve_1 - VARM : VARIABILA DE MEDIU TRANSMISA
        execve_1 - argv[0]: ./execve_1
        execve_1 - argv[1]: salut
        execve_1 - argv[2]: pe
        execve_1 - argv[3]: toata
        execve_1 - argv[4]: lumea
```