Exemplu de program care utilizeaza apelul de sistem fork()

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
        fork();
        printf("Hello world!\n);
        return 0;
}
```

Exemplu de program pentru evidentierea alocarii de memorie

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

int globalVar; /*  A global variable*/

int main(void)
{
    int localVar = 0;
    int* p = (int*) malloc(2);
    pid_t childPID = fork();

    // Putting value at allocated address
    *p = 0;

    if (childPID >= 0) // fork was successful
    {
        if (childPID == 0) // child process
        {
            printf("\n Child Process Initial Value :: localVar"
                  " = %d, globalVar = %d", localVar,
                  globalVar);
            localVar++;
            globalVar++;

            int c = 500;
            printf("\n Child Process :: localVar = %d, "
                  "globalVar = %d", localVar, globalVar);
```

```
            printf("\n Address of malloced mem child = %p "
                  "and value is %d", p, *p);
            printf("\n lets change the value pointed my malloc");

            *p = 50;
            printf("\n Address of malloced mem child = %p "
                  "and value is %d", p, *p);
            printf("\n lets change the value pointed my "
                  "malloc in child");

            *p = 200;
            printf("\n Address of malloced mem child = %p "
                  "and value is %d\n\n\n", p, *p);
         }
         else // Parent process
         {
            printf("\n Parent process Initial Value :: "
                  "localVar = %d, globalVar = %d",
                  localVar, globalVar);

            localVar = 10;
            globalVar = 20;
            printf("\n Parent process :: localVar = %d,"
                 " globalVar = %d", localVar, globalVar);
            printf("\n Address of malloced mem parent= %p "
                  "and value is %d", p, *p);

            *p = 100;
            printf("\n Address of malloced mem parent= %p "
                  "and value is %d", p, *p);
            printf("\n lets change the value pointed my"
                   " malloc in child");
            *p = 400;
            printf("\n Address of malloced mem child = %p"
                  " and value is %d \n", p, *p);
         }
      }
      else // fork failed
      {
         printf("\n Fork failed, quitting!!!!!!\n");
         return 1;
      }

      return 0;
}
```

Exemplu de utilizare a functiilor getpid() & getppid()

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main(void)
{
	//variabila care stocheaza id-ul procesului copil
	pid_t process_id;
	//variabila care stocheaza id-ul procesului parinte
	pid_t p_process_id;

	process_id = getpid();
	p_process_id = getppid();

	//afisarea id-urilor proceselor
	printf("ID-ul procesului copil: %d\n",process_id);
	printf("ID-ul procesului parinte: %d\n",p_process_id);

	return 0;
}
```

Exemplu de creare de procese Zombie

```
 #include<stdio.h>
#include<unistd.h>
#include<sys/wait.h>
#include<sys/types.h>

int main()
{
	int i;
	int pid = fork();

	if (pid == 0)
	{
		for (i=0; i<20; i++)
			printf("I am Child\n");
	}
	else
```

```
	{
		printf("I am Parent\n");
		while(1);
	}
}
```

//pentru a afisa tabelul de procese, rulati comanda **ps -eaf**

Exemplu de prevenire a crearii de procese Zombie

```
#include<stdio.h>
#include<unistd.h>
#include<sys/wait.h>
#include<sys/types.h>

int main()
{
    int i;
    int pid = fork();
    if (pid==0)
    {
        for (i=0; i<20; i++)
            printf("I am Child\n");
    }
    else
    {
        wait(NULL);
        printf("I am Parent\n");
        while(1);
    }
}
```

Exemplu de creare de procese orfan

```
#include<stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
	// Create a child process
	int pid = fork();

	if (pid > 0)
		printf("in parent process");

	// Note that pid is 0 in child process
	// and negative if fork() fails
```

```
        else if (pid == 0)
        {
                sleep(30);
                printf("in child process");
        }

        return 0;
}
```

Familia de functii exec

1. Creati fisierul **exec.c**

```
#include<stdio.h>
#include<unistd.h>

int main()
{
   int i;

   printf("I am EXEC.c called by execvp() ");
   printf("\n");

   return 0;
}
```

2. Creati un fisier executabil al fisierului **exec.c** (gcc EXEC.c -o EXEC)
3. Creati fisierul **execDemo.c**

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main()
{
       //array de pointers
       char *args[]={"./EXEC",NULL};
       execvp(args[0],args);

       /*Toate statementurile sunt ignorate dupa apelul execvp() intrucat procesu
execDemo.c este inlocui de procesul EXEC.c
       */
       printf("Ending-----");

   return 0;
}
```

4. Creati un fisier executabil al fisierului **execDemo.c** (gcc execDemo.c -o execDemo)

5. Rulati fisierul executabil **execDemo** (./execDemo)

Exemplu de prindere a unui semnal

```
#include<stdio.h>
#include<signal.h>
#include<unistd.h>

void sig_handler(int signo)
{
  if (signo == SIGINT)
    printf("Am primit SIGINT\n");
}

int main(void)
{
  if (signal(SIGINT, sig_handler) == SIG_ERR)
  printf("\nNu pot prinde SIGINT\n");
  while(1)
    sleep(1);
  return 0;
}
```

Exercitii

1. Scrieti un program in care tatal testeaza daca procesul fiu are un PID par. Daca da, atunci ii scrie fiului mesajul „*fortune*”, mesaj care va fi afisat de fiu. Daca PID-ul este impar, tatal va scrie mesajul „*lost*” si moare inaintea fiului.
2. Scrieti un program care executa comanda **ls -a -l**. Trebuiesc respectate:
    a. - procesul fiu este cel care va apela primitiva **execlp**
    b. - procesul tata asteapta ca fiul sa se termine si va afisa un mesaj
3. Sa se scrie un program care care afiseaza din 3 in 3 secunde PID-ul procesului curent si a cata afisare este.
    a. La primirea semnalului SIGUSR1 se scrie intr-un fisier mesajul "*Am primit semnal*".
    b. Semnalul SIGINT se va ignora in primele 60 de secunde de la inceperea rularii programului si apoi va avea actiunea *default*.