

UNIVERSITETI POLITEKNIK I TIRANËS  
FAKULTETI I TEKNOLOGJISË DHE INFORMACIONIT  
DEPARTAMENTI I INXHINIERISË INFORMATIKE

## **Punë Laboratori nr. 3**

**Lënda:** Sisteme Operative

**Grupi:** III-B

**Tema:** Krijimi dhe menaxhimi i proceseve dhe thread-eve me ane te thirrjeve te sistemit **fork( )** , **wait( )**, **pthread\_create( )**.

**Punoi:**  
Piro Gjikdhima

**Pranoi:**  
MSc.Megi Tartari

## Ushtrimi 1

Nderto nje program ne gjuhen C qe perdor thirrjen e sistemit `fork()` per te krijuar dy procese femije nga i njejti proces prind. Prindi P te kete dy procese femije P1 dhe P2.

```
C usht1.c x
C usht1.c
1 #include<stdio.h>
2 #include<unistd.h>
3 #include<sys/types.h>
4
5 int main(){
6
7     pid_t p1=fork();
8
9     if(p1==0){
10         printf("I am child having id %d\n",getpid());
11         printf("My parent's id is %d\n",getppid());
12     }
13     else{
14         pid_t p2 = fork();
15
16         if (p2 == 0){
17             printf("I am the child having id %d\n",getpid());
18             printf("My parent's id is %d\n",getppid());
19         }
20         else{
21             printf("My childrens' ids are: %d,%d\n",p1,p2);
22             printf("I am parent having id %d\n",getpid());
23         }
24     }
25     return 0;
26 }
27
28
```

Ky kod demonstroi krijimin e **dy proceseve fëmijë** nga një **proces prind** duke përdorur funksionin `fork()`. Kur `fork()` ekzekutohet për herë të parë, **procesi ndahet** në dy: një prind dhe një fëmijë. Nëse `fork()` kthen 0, kodi ekzekutohet nga **procesi fëmijë**, i cili shtyp **ID-në** e tij dhe të prindit. Në bllokun tjetër, **prindi krijon një proces tjetër fëmijë** me një thirrje të dytë `fork()`. Në fund, **procesi prind** raporton **ID-të** e të dy proceseve fëmijë dhe **ID-në** e tij. **Proceset fëmijë** dhe **prind** ekzekutojnë blloqe të ndryshme të kodit në bazë të vlerës së kthyer nga `fork()`. Në secilin ekzekutim, **output-i mund të jetë i ndryshëm** për shkak të mënyrës se si **skeduleri i sistemit operativ** cakton proceset.

### Outpute:

```
[Running] cd "/home/kali/Documents/" && gcc usht1.c -o usht1 && "/home/kali/Documents/"usht1
I am child having id 9881
My parent's id is 9875
My childrens' ids are: 9881,9882
I am parent having id 9875
I am the child having id 9882
My parent's id is 1

[Done] exited with code=0 in 0.066 seconds

[Running] cd "/home/kali/Documents/" && gcc usht1.c -o usht1 && "/home/kali/Documents/"usht1
My childrens' ids are: 9903,9904
I am parent having id 9902
I am child having id 9903
My parent's id is 1
I am the child having id 9904
My parent's id is 1

[Done] exited with code=0 in 0.047 seconds
```

## Ushtrimi 2

Nderto nje program ne gjuhen C qe perdor thirrjen e sistemit fork() per te krijuar nje hierarki prej 3 procesesh te tille qe P2 eshte femija i P1 dhe P1 femija i P.

```
1 #include<stdio.h>
2 #include<unistd.h>
3 #include<sys/types.h>
4 #include<sys/wait.h>
5
6 int main()
7 {
8     pid_t p= fork();
9
10    if(p==0){
11
12        pid_t p1=fork();
13
14        if (p1 == 0){
15
16            pid_t p2;
17            p2 = fork();
18
19            if (p2 == 0){
20                printf("I am the child having id %d\n",getpid());
21                printf("My parent's id is %d\n",getppid());
22            }
23            else{
24                printf("I am child having id %d\n",getpid());
25                printf("My parent's id is %d\n",getppid());
26                printf("My child's id is: %d\n",p2);
27            }
28        }
29        else{
30            printf("I am child having id %d\n",getpid());
31            printf("My parent's id is %d\n",getppid());
32            printf("My child's id is: %d\n",p1);
33        }
34    }
35    else{
36        printf("My child's id is: %d\n",p);
37        printf("I am parent having id %d\n",getpid());
38    }
39    return 0;
40 }
```

Ky kod krijon një **hierarki proceseve** duke përdorur thirrje të shumta **fork()**. Fillimisht, **procesi prind** krijon një **proces fëmijë (p)**. Brenda këtij fëmijë, krijohet një tjetër **proces fëmijë (p1)**. Më tej, brenda fëmijës së dytë (**p1**), krijohet një **proces tjetër (p2)**. Çdo **proces fëmijë** shtyp **ID-në** e tij, **ID-në** e prindit dhe, nëse ka, **ID-në** e fëmijës së tij. **Procesi prind** shtyp vetëm **ID-në** e fëmijës së parë që krijoi (**p**). **Hierarkia** ilustron raportet ndërmjet proceseve prind dhe fëmijë.

**Outpute:**

```
[Running] cd "/home/kali/Documents/" && gcc usht2.c -o usht2 && "/home/kali/Documents/"usht2
My child's id is: 12318
I am parent having id 12317
I am child having id 12318
My parent's id is 1
My child's id is: 12319
I am child having id 12319
My parent's id is 1
My child's id is: 12320
I am the child having id 12320
My parent's id is 1

[Done] exited with code=0 in 0.048 seconds

[Running] cd "/home/kali/Documents/" && gcc usht2.c -o usht2 && "/home/kali/Documents/"usht2
My child's id is: 12519
I am parent having id 12518
I am child having id 12519
My parent's id is 12518
My child's id is: 12520
I am child having id 12520
My parent's id is 1
My child's id is: 12521
I am the child having id 12521
My parent's id is 1

[Done] exited with code=0 in 0.042 seconds
```

### Ushtrimi 3

Nderto nje program ne gjuhen C me dy procese prind, femije. Prindi duhet te printoje dy fjali:

- a) Prindi ka PID <pid>
- b) PID i femijes eshte <PID>.

Procesi femije duhet te printoje dy fjali:

- c) Femija ka PID <PID>
- d) PID i prindit tim eshte <PID>.

Perdor thirrjen e sistemi wait() ne menyre te tille qe te prinoen fillimisht fjalite e procesit child, me pas te prindit.

```
C usht3.c X
C usht3.c
1  #include<stdio.h>
2  #include<unistd.h>
3  #include<sys/types.h>
4  #include<sys/wait.h>
5
6  int main(){
7
8      pid_t p=fork();
9
10     if(p==0){
11         printf("Femija ka PID %d\n",getpid());
12         printf("PID i prindit tim eshte %d\n",getppid());
13     }
14     else{
15         wait(NULL);
16         printf("Prindi ka PID: %d\n",getpid());
17         printf("PID i femijes eshte: %d\n",p);
18     }
19     return 0;
20 }
21
22
```

Ky kod krijon një **proces fëmijë** nga **procesi prind** duke përdorur **fork()**. **Procesi fëmijë** ekzekuton bllokun e kodit brenda **if (p == 0)** dhe shtyp **ID-në** e tij dhe të prindit të tij. Pas kësaj, **procesi prind** përdor **wait(NULL)** për të pritur përfundimin e **procesit fëmijë** dhe më pas shtyp **ID-në** e tij dhe **ID-në** e procesit fëmijë. Kjo siguron që **procesi prind** pritët të përfundojë pas **procesit fëmijë** dhe të dy proceset ekzekutohen në mënyrë të **sinkronizuar**.

**Outpute:**

```
[Running] cd "/home/kali/Documents/" && gcc usht3.c -o usht3 && "/home/kali/Documents/
"usht3
Femija ka PID 14641
PID i prindit tim eshte 14639
Prindi ka PID: 14639
PID i femijes eshte: 14641

[Done] exited with code=0 in 0.051 seconds

[Running] cd "/home/kali/Documents/" && gcc usht3.c -o usht3 && "/home/kali/Documents/
"usht3
Femija ka PID 14666
PID i prindit tim eshte 14665
Prindi ka PID: 14665
PID i femijes eshte: 14666

[Done] exited with code=0 in 0.047 seconds
```

## Ushtrimi 4

Nderto nje program qe krijon nje thread T1 me ane te `pthread_create()`. Procesi kryesor i kalon dy numra thread-it T1. T1 llogarit shumen e ketyre dy numrave dhe ia kthen rezultatin procesit kryesor i cili e printon ne ekran.

```
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<unistd.h>
4 #include<pthread.h>
5
6 // Struktura per te ruajtur te dhënat qe do te kalohen ne thread
7 typedef struct {
8     int num1;
9     int num2;
10    int result;
11 } ThreadData;
12
13 // Funksioni qe do te ekzekutohet nga thread-1
14 void* calculate_sum(void* arg) {
15     ThreadData* data = (ThreadData*) arg;
16     data->result = data->num1 + data->num2;
17     return NULL;
18 }
19
20 int main() {
21     pthread_t thread; // Identifikuesi i thread-it
22     ThreadData data; // Struktura qe do te perdoret per te kaluar te dhënat
23
24     // Input per dy numrat
25     printf("Jepni numrin e pare: ");
26     scanf("%d", &data.num1);
27     printf("Jepni numrin e dyte: ");
28     scanf("%d", &data.num2);
29
30     // Krijimi i thread-it T1
31     if (pthread_create(&thread, NULL, calculate_sum, (void*)&data) != 0) {
32         perror("pthread_create");
33         return 1;
34     }
35
36     // Pritja qe thread-1 te perfundoje
37     if (pthread_join(thread, NULL) != 0) {
38         perror("pthread_join");
39         return 1;
40     }
41
42     // Printimi i rezultatit
43     printf("Shuma e %d dhe %d eshte: %d\n", data.num1, data.num2, data.result);
44
45     return 0;
46 }
47
```

Ky kod perdor multithreading per te llogaritur shumën e dy numrave duke perdorur nje thread te veçantë.

Në funksionin **main**, perdoruesi jep dy numra, te cilët ruhen në një strukturë të quajtur **ThreadData**. Pastaj krijohet një thread duke perdorur funksionin **pthread\_create**, i cili thërret funksionin **calculate\_sum**. Ky funksion merr strukturën si argument dhe llogarit shumën e dy numrave, duke ruajtur rezultatin në fushën **result** të strukturës. Pasi thread-i perfundon, **pthread\_join** siguron qe procesi prind te presë perfundimin e thread-it. Më në fund, rezultati i shumës shtypet në ekran.

Output:

```
(kali@kali)-[~/Documents]
$ ./usht4
Jepni numrin e pare: 10
Jepni numrin e dyte: 20
Shuma e 10 dhe 20 eshte: 30

(kali@kali)-[~/Documents]
$
```

## Ushtrimi nga seminari 2

Ndertoni një program ne gjuhën C qe përdor pipe, ku një proces dërgon një string në një proces të dytë, dhe procesi i dytë konverton stringen ku karakteret kthehen nga shkronja te medha ne te vogla ose anasjelltas me pas dërgohet përsëri tek procesi i parë. Për shembull, nëse procesi i parë dërgon mesazhin "Hi There", procesi i dytë do të kthejë "hI tHERE". Përdorni dy pipe, një për dërgimin e mesazhit origjinal nga procesi i parë në procesin e dytë dhe tjetrin për dërgimin e mesazhit të konvertuar nga procesi i dytë në procesin e parë.

Per te ndertuar programet me lart referojuni programeve shembull me poshte:

### **FORK( )**

```
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
int main()
{
    pid_t p;
    printf("before fork\n");
    p=fork();
    if(p==0)
    {
        printf("I am child having id %d\n",getpid());
        printf("My parent's id is %d\n",getppid());
    }
    else{
        printf("My child's id is %d\n",p);
        printf("I am parent having id %d\n",getpid());
    }
    printf("Common\n");
}
```

### **WAIT( )**

```
#include<unistd.h>
#include<sys/types.h>
#include<stdio.h>
#include<sys/wait.h>
int main()
{
    pid_t p;
    printf("before fork\n");
    p=fork();
    if(p==0)//child
    {
        printf("I am child having id %d\n",getpid());
        printf("My parent's id is %d\n",getppid());
    }
```

```

}
else//parent
{
wait(NULL);
printf("My child's id is %d\n",p);
printf("I am parent having id %d\n",getpid());
}
printf("Common\n");
}

```

## **PTHREAD\_CREATE()**

```

#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<pthread.h>
void *thread_function(void *arg);
int i,j;
int main() {
pthread_t a_thread;  //thread declaration

pthread_create(&a_thread, NULL, thread_function, NULL);
//thread is created
pthread_join(a_thread, NULL); //process waits for thread to
finish . //Comment this line to see the difference
printf("Inside Main Program\n");
for(j=20;j<25;j++)
{
printf("%d\n",j);
sleep(1);
}
}

void *thread_function(void *arg) {
// the work to be done by the thread is defined in this function
printf("Inside Thread\n");
for(i=0;i<5;i++)
{
printf("%d\n",i);
sleep(1);
}
}

```

```

C Ushtrimi2.c x
C Ushtrimi2.c
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <string.h>
4 #include <ctype.h>
5 #include <stdlib.h>
6
7 #define BUFFER_SIZE 100
8
9 // Funkcioni për të kthyer shkronjat nga uppercase në lowercase dhe anasjelltas
10 void toggle_case(char* str) {
11     for (int i = 0; str[i] != '\0'; i++) {
12         if (islower(str[i])) {
13             str[i] = toupper(str[i]);
14         } else if (isupper(str[i])) {
15             str[i] = tolower(str[i]);
16         }
17     }
18 }
19
20 int main() {
21     int pipe1[2], pipe2[2]; // Dy pipe: një për dërgimin dhe një për kthimin
22     pid_t pid;
23     char input[BUFFER_SIZE];
24     char buffer[BUFFER_SIZE];
25
26     // Krijimi i pipe-ve
27     if (pipe(pipe1) == -1 || pipe(pipe2) == -1) {
28         perror("pipe");
29         exit(1);
30     }
31
32     pid = fork(); // Krijimi i procesit fëmijë
33
34     if (pid < 0) {
35         perror("fork");
36         exit(1);
37     }
38
39     if (pid == 0) { // Procesi fëmijë
40         close(pipe1[1]); // Mbyllet fundin e shkrimit të pipel
41         close(pipe2[0]); // Mbyllet fundin e leximit të pipel
42     }

```

```

43     // Lexojmë string-un nga procesi prind përmes pipe1
44     read(pipe1[0], buffer, BUFFER_SIZE);
45     printf("Procesi fëmijë mori: %s\n", buffer);
46
47     // Konvertojmë rastin e shkronjave
48     toggle_case(buffer);
49
50     // Dërgojmë string-un e konvertuar te procesi prind përmes pipe2
51     write(pipe2[1], buffer, strlen(buffer) + 1);
52
53     // Mbyllet pipe-at që kemi përdorur
54     close(pipe1[0]);
55     close(pipe2[1]);
56 } else { // Procesi prind
57     close(pipe1[0]); // Mbyllet fundin e leximit të pipel
58     close(pipe2[1]); // Mbyllet fundin e shkrimit të pipel
59
60     // Marrim input nga përdoruesi
61     printf("Shkruani një string: ");
62     fgets(input, BUFFER_SIZE, stdin);
63     input[strcspn(input, "\n")] = '\0'; // Hiq newline nga fgets
64
65     // Dërgojmë string-un te procesi fëmijë përmes pipe1
66     write(pipe1[1], input, strlen(input) + 1);
67
68     // Presim përgjigjen nga procesi fëmijë përmes pipe2
69     read(pipe2[0], buffer, BUFFER_SIZE);
70     printf("Procesi prind mori: %s\n", buffer);
71
72     // Mbyllet pipe-at që kemi përdorur
73     close(pipe1[1]);
74     close(pipe2[0]);
75 }
76
77 return 0;
78 }
79

```

Ky kod përdor **dy pipe** dhe një **proces fëmijë** për të konvertuar rastin e shkronjave në një string të dhënë nga përdoruesi.

Në funksionin **main**, krijohen **dy pipe**: një për dërgimin e të dhënave nga prindi tek fëmija dhe një tjetër për kthimin e të dhënave nga fëmija tek prindi. Pasi krijohet një **proces fëmijë** me **fork()**, procesi prind merr një string nga përdoruesi, e dërgon atë te procesi fëmijë përmes **pipe1**, dhe pastaj pret që fëmija të kthejë rezultatin përmes **pipe2**.

Procesi fëmijë lexon të dhënat nga **pipe1**, konverton rastin e shkronjave të string-ut nga **uppercase në lowercase** dhe anasjelltas, dhe e dërgon rezultatin prapa te prindi përmes **pipe2**. Çdo proces mbyllet fundin e papërdorur të **pipe-ve** për të parandaluar gabimet e mundshme gjatë komunikimit.

**Output:**

```

C Ushtrimi2.c X
C Ushtrimi2.c
1 #include <stdio.h>
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

• (kali@kali) - [~/Documents]
$ ./Ushtrimi2
Shkruani një string: Hello World
Procesi fëmijë mori: Hello World
Procesi prind mori: HELLO WORLD

• (kali@kali) - [~/Documents]
$ ./Ushtrimi2
Shkruani një string: qqqqEEEEEEfjsufhSFDAGIUGYt
Procesi fëmijë mori: qqqqEEEEEEfjsufhSFDAGIUGYt
Procesi prind mori: QQQQEEEEEEfJSUFHsfdagiugYt

• (kali@kali) - [~/Documents]
$

```

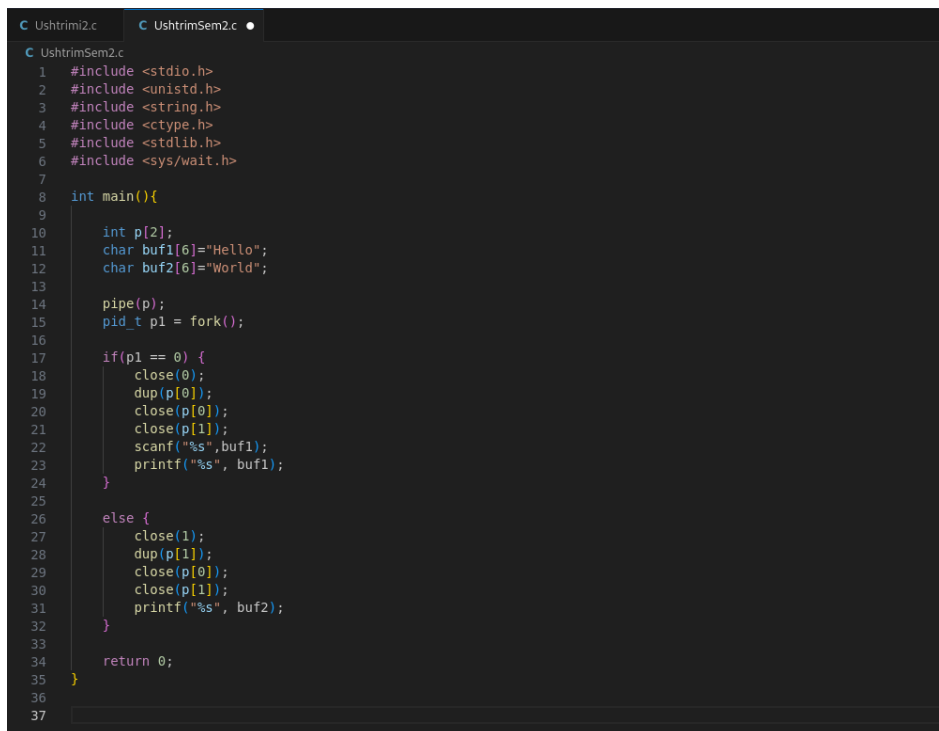


## Ushtrimi nga seminari 2

Plotesoni programin me poshte ne menyre korrekte ne gjihen C dhe nga ekzekutimi shikoni cfare do te printohet ne ekran ?

```
int p[2];
char buf1[6]="Hello"
char buf2[6]="World"
pipe(p);
if(fork() == 0) {
    close(0);
    dup(p[0]);
    close(p[0]);
    close(p[1]);
    scanf("%s",buf1);
    printf("%s, buf1);
} else {
    close(1);
    dup(p[1]);
    close(p[0]);
    close(p[1]);
    printf("%s, buf2);
}
```

### Kodi I përmirësuar



```
C Ushtrimi2.c C UshtrimSem2.c
C UshtrimSem2.c
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <string.h>
4 #include <ctype.h>
5 #include <stdlib.h>
6 #include <sys/wait.h>
7
8 int main(){
9
10     int p[2];
11     char buf1[6]="Hello";
12     char buf2[6]="World";
13
14     pipe(p);
15     pid_t p1 = fork();
16
17     if(p1 == 0) {
18         close(0);
19         dup(p[0]);
20         close(p[0]);
21         close(p[1]);
22         scanf("%s",buf1);
23         printf("%s", buf1);
24     }
25
26     else {
27         close(1);
28         dup(p[1]);
29         close(p[0]);
30         close(p[1]);
31         printf("%s", buf2);
32     }
33
34     return 0;
35 }
36
37
```

Diferencat midis dy kodeve janë si më poshtë:

1. **Deklarimi i variablave:**

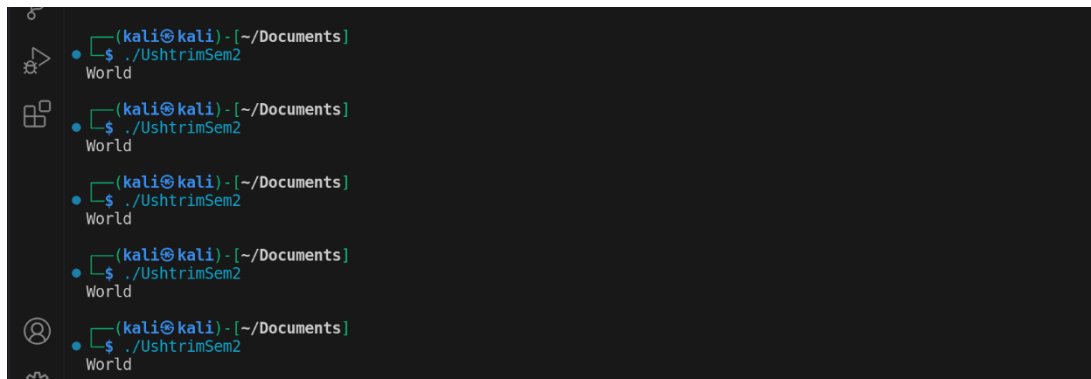
- Në kodin e parë, variablat **int p[2];** dhe **char buf1[6]="Hello";** janë të deklaruara brenda funksionit **main()**.
- Në kodin e dytë, variablat **int p[2];** dhe **char buf1[6]="Hello";** janë të deklaruara jashtë funksionit **main()**, gjë që mund të shkaktojë gabim, pasi variablat nuk janë brenda ndonjë funksioni dhe nuk mund të përdoren ashtu si janë.

2. **Shtypja e printf():**

- Në kodin e dytë, ka një gabim në sintaksën e **printf("%s, buf1);** dhe **printf("%s, buf2);**, ku do të duhej të ishin **printf("%s", buf1);** dhe **printf("%s", buf2);** për të shtypur variablat në mënyrë të saktë.

Kodi krijon një **pipe** që përdoret për të dërguar dhe marrë të dhëna mes procesit prind dhe fëmijës. Procesi prind shkruan "World" në pipe dhe procesi fëmijë e lexon këtë të dhënë dhe e paraqet në ekran si "World". Ky është një shembull i përdorimit të **pipe()**, **fork()**, dhe **dup()** për komunikim ndërmjet proceseve.

**Output:**



```
(kali@kali) - [~/Documents]
$ ./UshtrimSem2
World

(kali@kali) - [~/Documents]
$ ./UshtrimSem2
World

(kali@kali) - [~/Documents]
$ ./UshtrimSem2
World

(kali@kali) - [~/Documents]
$ ./UshtrimSem2
World

(kali@kali) - [~/Documents]
$ ./UshtrimSem2
World
```