C Programming Basics

Lab Session: System Calls in C Programming

Course: Operating Systems
Date: 23.05.2025 Practical01

Introduction to System Calls in C

System calls are functions provided by the OS kernel. In C, they enable interaction between a user-level application and the OS, such as process control using fork(), getpid(), and getppid().

---

1. Basic fork() and getpid()

```
#include <stdio.h>

#include <unistd.h>


int main(){

   printf("\nHello world!");

   int f = fork();

   int p = getpid();

   printf("\nMy PID is %d", p);

   printf("\nValue of f (PID returned by fork) is %d", f);

   return 0;

}
```

Fedora Output:

```
[2021ict108@fedora ~]$ gcc basic_fork.c -o basic_fork

[2021ict108@fedora ~]$ ./basic_fork

Hello world!

My PID is 2321
```

Value of f is 2322

Hello world!

My PID is 2322

Value of f is 0

Explanation:

- fork() returns 0 for the child and the PID of the child to the parent.

---

2. Identifying Parent and Child

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

int main(){
   printf("\nHello world!");
   int f = fork();
   int p = getpid();

   if (f > 0) {
      printf("\nI am the parent, my PID is %d and my child's PID is %d", p, f);
   } else if (f == 0) {
      printf("\nI am the child, my PID is %d", p);
   } else {
      printf("\nFork failed");
   }
   return 0;
}
```

Fedora Output:

[2021ict108@fedora ~]$ gcc parent_child.c -o parent_child

[2021ict108@fedora ~]$ ./parent_child

Hello world!

I am the parent, my PID is 2456 and my child's PID is 2457

Hello world!

I am the child, my PID is 2457

Explanation:

- Child and parent identify themselves using PID.

---

3. Two Child Processes

c

CopyEdit

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

int main(){
   int f1, f2;
   int p = getpid();
   printf("Process A (Parent) PID: %d\n", p);

   f1 = fork();

   if (f1 == 0) {
      printf("Process B (Child 1) PID: %d, Parent PID: %d\n", getpid(), getppid());
```

```
    } else {
        f2 = fork();
        if (f2 == 0) {
            printf("Process C (Child 2) PID: %d, Parent PID: %d\n", getpid(), getppid());
        }
    }
    return 0;
}
```

Fedora Output:

```
[2021ict108@fedora ~]$ gcc two_children.c -o two_children
[2021ict108@fedora ~]$ ./two_children
Process A (Parent) PID: 2521
Process B (Child 1) PID: 2522, Parent PID: 2521
Process C (Child 2) PID: 2523, Parent PID: 2521
```

Explanation:

- Two children created from one parent.

---

4. Simple fork() Example

```
#include <stdio.h>
#include <unistd.h>
int main(){
    fork();
    printf("Hello World!\n");
    return 0;
}
```

Fedora Output:

```
[2021ict108@fedora ~]$ gcc simple_fork.c -o simple_fork
[2021ict108@fedora ~]$ ./simple_fork
Hello World!
Hello World!
```

Explanation:

- Message printed twice due to two processes.

---

5. Parent and Child Message

```
#include <stdio.h>
#include <unistd.h>
int main(){
   int id = fork();
   if(id == 0)
      printf("I am the child!\n");
   else
      printf("I am the parent!\n");
   return 0;
}
```

Fedora Output:

```
[2021ict108@fedora ~]$ gcc parent_child_msg.c -o parent_child_msg
[2021ict108@fedora ~]$ ./parent_child_msg
I am the parent!
I am the child!
```

---

6. Number Printing by Parent and Child

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

```
#include <unistd.h>
int main(){
   int sum = 0;
   int id = fork();

   if(id == 0){
      printf("Child process printing 1 to 5\n");
      for(int i = 1; i <= 5; i++) {
         printf("%d\n", i);
         sum += i;
      }
      printf("Child Sum: %d\n", sum);
   } else if(id > 0){
      printf("Parent process printing 6 to 10\n");
      for(int i = 6; i <= 10; i++) {
         printf("%d\n", i);
         sum += i;
      }
      printf("Parent Sum: %d\n", sum);
   }
   return 0;
}
```

Fedora Output:

```
[2021ict108@fedora ~]$ gcc split_sum.c -o split_sum
```

[2021ict108@fedora ~]$ ./split_sum

Child process printing 1 to 5

1

2

3

4

5

Child Sum: 15

Parent process printing 6 to 10

6

7

8

9

10

Parent Sum: 40

Explanation:

- Child calculates sum from 1–5.
- Parent calculates sum from 6–10.

---

7. Summing in Parent and Child and Total Sum

```
#include <stdio.h>
#include <unistd.h>
int main(){
   int sum1 = 0, sum2 = 0;
   int id = fork();
```

```
    if (id == 0) {
        for (int i = 1; i <= 5; i++) sum1 += i;
        printf("Child Sum: %d\n", sum1);
    }
    for (int i = 6; i <= 10; i++) sum2 += i;
    printf("Parent Sum: %d\n", sum2);
    printf("Total Sum: %d\n", sum1 + sum2);
    return 0;
}
```

Fedora Output:

```
[2021ict108@fedora ~]$ gcc total_sum.c -o total_sum
[2021ict108@fedora ~]$ ./total_sum
Child Sum: 15
Parent Sum: 40
Total Sum: 15
Total Sum: 55
```

Explanation:

- Due to parallelism, sum1 and sum2 are not shared.
- Total printed separately in both processes.