Operating Systems

CSN-232

Dr. R. Balasubramanian

Associate Professor

Department of Computer Science and Engineering

Indian Institute of Technology Roorkee

Roorkee 247 667

balarfcs@iitr.ac.in

https://sites.google.com/site/balaiitr/

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

    int main()
{
    fork();
    fork() && fork() || fork();
    fork();

    printf("IITR\n");
    return 0;
}
```

http://tpcg.io/7EQsrC

```
$gcc -o main *.c
$main
IITR
IITR
IITR
IITR
IITR
IITR
IITR
IITR
IITR
IITR
IITR
IITR
IITR
IITR
IITR
IITR
IITR
IITR
IITR
IITR
```

Process Identification

- The **pid_t** data type represents process IDs.
- We can get the process ID of a process by calling `getpid` .
- The function `getppid` returns the process ID of the parent of the current process (this is also known as the parent process ID).

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

int main()
{
    pid_t pid;

    pid=fork();

    printf("PID = %d\n", pid);

    return 0;
}
```

```
$gcc -o main *.c
$main
PID = 145735
PID = 0
```

http://tpcg.io/6z3ueX

```cpp
#include <iostream>
#include <unistd.h>
using namespace std;

// Driver Code
int main()
{
    int pid;
    pid = fork();
    if (pid == 0)
    {
        cout << "\nprocess ID of the current process : "
             << getpid() << endl;
        cout << "\nthe process ID of the parent of the current process : "
             << getppid() << endl;
    }

    return 0;
}
```

```
$g++ -o main *.cpp
$main

process ID of the current process : 167821

the process ID of the parent of the current process : 0
```

http://tpcg.io/dYpNcl

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

int main()
{
    pid_t cpid;
    if (fork()== 0)
    {   printf("Test\n");
        exit(0);            /* terminate child */
    }
    else
        cpid = wait(NULL); /* reaping parent */
    printf("Parent pid = %d\n", getpid());
    printf("Child pid = %d\n", cpid);

    return 0;
}
```

http://tpcg.io/qnJa9s

```
$gcc -o main *.c
$main
Test
Parent pid = 40884
Child pid = 40885
```

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

int main()
{
    pid_t cpid;
    if (fork()== 0)
    {   printf("Test\n");
      //  exit(0);            /* terminate child */
    }
    else
        cpid = wait(NULL); /* reaping parent */
    printf("Parent pid = %d\n", getpid());
    printf("Child pid = %d\n", cpid);

    return 0;
}
```

http://tpcg.io/iQE9bL

```
$gcc -o main *.c
$main
Test
Parent pid = 75819
Child pid = 0
Parent pid = 75818
Child pid = 75819
```

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

int main()
{
    if (fork()== 0)
        printf("HC: hello from child\n");
    else
    {
        printf("HP: hello from parent\n");
        wait(NULL);
        printf("CT: child has terminated\n");
    }

    printf("Bye\n");
    return 0;
}
```

http://tpcg.io/ZpVgMn

```
$gcc -o main *.c
$main
HC: hello from child
Bye
HP: hello from parent
CT: child has terminated
Bye
```

exec family of functions in C

- **execvp** : Using this command, the created child process does not have to run the same program as the parent process does. The **exec** type system calls allow a process to run any program files, which include a binary executable or a shell script .

- **Syntax:** `int execvp (const char *file, char *const argv[]);`

- **file:** points to the file name associated with the file being executed.
  argv: is a null terminated array of character pointers.

exec family of functions in C

```c
//EXEC.c

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

int main()
{
    int i;

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

    return 0;
}
```

```
gcc EXEC.c -o EXEC
```

```c
//execDemo.c

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

        /*All statements are ignored after execvp() call as this whole
        process(execDemo.c) is replaced by another process (EXEC.c)
        */
        printf("Ending-----");

    return 0;
}
```

```
gcc execDemo.c -o execDemo
```

I am EXEC.c called by execvp()

Process Termination

- Process executes last statement and then asks the operating system to delete it using the **`exit()`** system call.
  - Returns status data from child to parent (via **`wait()`**)
  - Process' resources are deallocated by operating system

- Parent may terminate the execution of children processes using the **`abort()`** system call. Some reasons for doing so:
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
  - The parent is exiting and the operating systems does not allow a child to continue if its parent terminates

Process Termination

- Some operating systems do not allow child to exists if its parent has terminated. If a process terminates, then all its children must also be terminated.
  - **cascading termination.** All children, grandchildren, etc. are terminated.
  - The termination is initiated by the operating system.
- The parent process may wait for termination of a child process by using the **`wait()`** system call**.** The call returns status information and the pid of the terminated process

```
pid = wait(&status);
```

- If no parent waiting (did not invoke **`wait()`**) process is a **zombie**
- If parent terminated without invoking **`wait`** , process is an **orphan**