Chapter 2: Process Management

SYSTEM CALLS

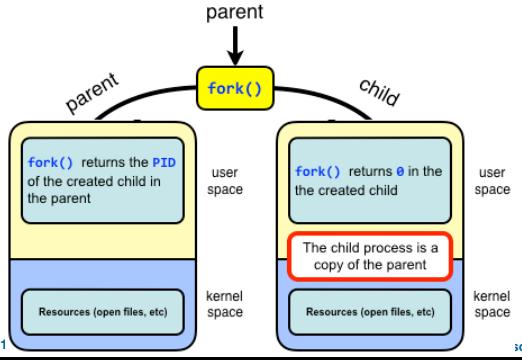
GV: Nguyễn Thị Thanh Vân

System calls

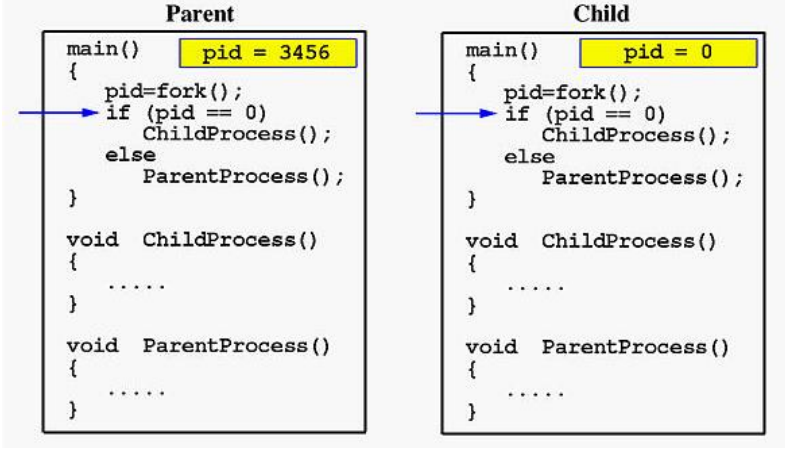
- fork()
- exec family
- wait()
- exit()
- getpid()
- getppid()

fork()

- fork( ): Used to create new processes. It consists of a copy of the address space of the original process. Both run concurrently. Returned values :
  - Negative Value: creation of a child process was unsuccessful.
  - Zero: Returned to the newly created child process.
  - Positive value: Returned to parent or caller
- Syntax : fork( )
- Ex: // make two process which run same prog after this instruction fork();



fork()



fork(), Ex

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
void forkexample()
{
// child process because return value zero
if (fork() == 0)
printf("Hello from Child!\n");
// parent process because return value non-zero.
else
printf("Hello from Parent!\n");
}
int main()
{
forkexample();
return 0;
}
```

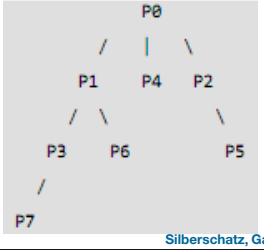
fork(), Ex

- Calculate number of times hello is printed

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
    fork();
    fork();
    fork();
    printf("hello\n");
    return 0;
}
```

```
fork ();   // Line 1
fork ();   // Line 2
fork ();   // Line 3


      L1       // There will be 1 child process
   /     \     // created by line 1.
  L2      L2    // There will be 2 child processes
 /  \    /  \   //  created by line 2
L3  L3  L3  L3  // There will be 4 child processes
                // created by line 3
```



- n is number of fork system calls.

So n = 3, 2^3 = 8 processes

fork(), Ex

- A process executes the following code

```
for (i = 0; i < n; i++)
fork();
```

- The total number of child processes created is
  (A) n
  (B) $2^n - 1$
  (C) 2^n
  (D) $2^{(n+1)} - 1$

fork(), Ex

- Consider the following code fragment:

```
if (fork() == 0)
  { a = a + 5; printf("%d,%d\n", a, &a); }
else { a = a -5; printf("%d, %d\n", a, &a); }
```

- Let u, v be the values printed by the parent process, and x, y be the values printed by the child process. Which one of the following is TRUE?

(A) u = x + 10 and v = y
(B) u = x + 10 and v != y
(C) u + 10 = x and v = y
(D) u + 10 = x and v != y

fork(), Ex

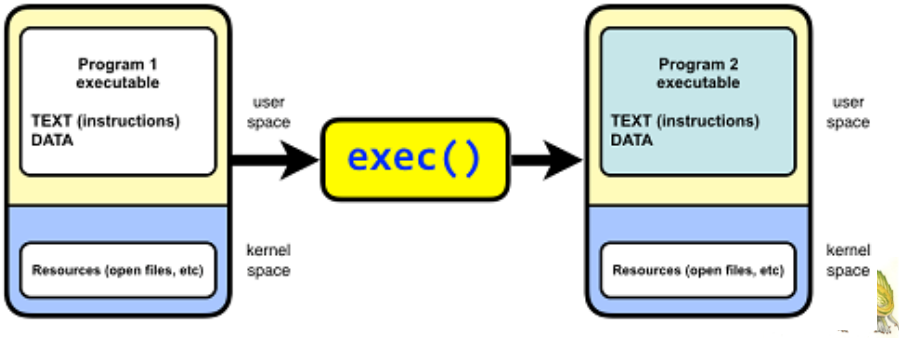
- How many processes will be spawned after executing the program?

```
#include <stdio.h>
#include <unistd.h>
int main()
{
    fork();
    fork() && fork() || fork();
    fork();

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

The exec family

- The exec family: replaces the program executed by a process.
- When a process calls **exec**, all code (text) and data in the process is **lost** and **replaced** with the executable of the **new program**. Although all data is replaced, all open file descriptors remains open after calling **exec** unless explicitly set to *close-on-exec*.
- In the below diagram a process is executing Program 1. The program calls **exec** to replace the program executed by the process to Program 2.



The exec family

- execvp(): allow a process to run any program files, which include a binary executable or a shell script
- execv(): similar to execvp()
- execlp() & execl(): system call duplicates the actions of the shell in searching for an executable file if the specified file name does not contain a slash (/)
- execvpe() & execle()

execvp()

- 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.

```
int main()  //EXEC.c
{
    int i;
    printf("I am EXEC.c called by execvp() ");
    printf("\n");
    return 0;
}
```

```
int main()  //execDemo.c
{
    //A null terminated array of character pointers
    char *args[]={"./EXEC",NULL};
    execvp(args[0],args);
    printf("Ending-----");
    return 0;
}
```

execv()

- This is very similar to execvp() function in terms of syntax as well.
- Syntax: `int execv(const char *path, char *const argv[]);`

- path: should point to the path of the file being executed.

- argv[]: is a null terminated array of character pointers.

```
int main()  //EXEC.c
{
    int i;
    printf("I am EXEC.c called by execv() ");
    printf("\n");
    return 0;
}
```

```
int main()  //execDemo.c
{
    //A null terminated array of character pointers
    char *args[]={"./EXEC",NULL};
    execv(args[0],args);
     printf("Ending-----");
    return 0;
}
```

execlp()& execl()

- Syntax:

`int execlp(const char *file, const char *arg,.../* (char  *) NULL */);`

`int execl(const char *path, const char *arg,.../* (char  *) NULL */);`

- file: file name associated with the file being executed

- const char *arg and ellipses : describe a list of one or more pointers to null-terminated strings that represent the argument list available to the executed program.

- - The execlp system call duplicates the actions of the shell in searching for an executable file if the specified file name does not contain a slash (/) character. The search path is the path specified in the environment by the PATH variable. If this variable isn't specified, the default path ":/bin:/usr/bin" is used.
- - The execlp system call can be used when the number of arguments to the new program is known at compile time. If the number of arguments is not known at compile time, use execvp.

execvpe() and execle()

- Syntax:

```
int execvpe(const char *file, char *const argv[],char
*const envp[]);
```

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

- char * const envp[]: allow the caller to specify the environment of the executed program via the argument envp.

- envp:This argument is an array of pointers to null-terminated strings and must be terminated by a null pointer. The other functions take the environment for the new process image from the external variable environ in the calling process.

Difference between fork() and exec()

- fork starts a new process which is a copy of the one that calls it, while exec replaces the current process image with another (different) one.
- Both parent and child processes are executed simultaneously in case of fork() while Control never returns to the original program unless there is an exec() error.

wait( ) and exit( )

- wait( ): The parent waits for the child process to complete using the wait system call. The wait system call returns the process identifier of a terminated child, so that the parent can tell which of its possibly many children has terminated.

  Syntax : `wait( NULL)`

- exit( ): A process terminates when it finishes executing its final statement and asks the operating system to delete it by using the exit system call. At that point, the process may return data (output) to its parent process (via the wait system call)

  Syntax: `exit(0)`

getpid() and getppid( )

- Getpid(): Each process is identified by its id value. This function is used to get the id value of a particular process.
- getppid( ) : Used to get particular process parent‟s id value.
- perror( ) : Indicate the process error.

End of Chapter 2.1