جامعة الأمير مقرن بن عبد العزيز
University of Prince Mugrin

Lab 4
Processes

Objectives

جامعة الأمير مقرن بن عبد العزيز
University of Prince Mugrin

- To learn process creation and management in Linux
- To learn system calls fork (), exec (), getpid (), wait () and exit ()

UNIX System Calls for Process Control

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

CREATING MULTIPLE PROCESSES USING `fork()`

- When a program calls **fork**, a duplicate process – a child process - is **created**.
- The **parent process** continues executing the program from the point that **fork** was called.
- **The child process, too, executes the same program. However, execution starts with the next instruction after fork.**

CREATING MULTIPLE PROCESSES USING FORK

- The **fork function** is being called from the parent; however, it returns two values, one in the child and the other in the parent.

1. `fork()` returns **zero** in the **child** process
2. `fork()` returns the **process identifier** of the child in **parent**
3. On failure, **-1** is returned to the **parent** process

- **Function:** `pid_t pid = fork (void)`
- **Header file:** `#include <unistd.h>`

The exec () system call

- `exec()` is used to **execute a file** from **within a program**.
- When `exec()` is called it replaces the **calling process** image with a **new process** image. The calling process will **no** longer exist. The new process assumes the **process ID** of the **calling process**.
- `exec()` usually used **after** `fork()` system call.
- `execlp()` is a variation of `exec()`

- **Syntax:** `int execlp (char const *file, char const *arg0, ...);`
- **Header file:** `#include <unistd.h>`

The getpid() system call

- The process ID of a process can be obtained by calling `getpid()`.
- The function `getppid()` returns the **process ID of the parent** of the current process.
- Both functions return process ID of type **pid_t**, which is basically a signed integer type.

- **Syntax:**

```
pid_t getpid (void);
pid_t getppid (void);
```

- **Header file:** `#include <unistd.h>`

wait () system call

- This function **blocks the calling process until the child process terminates.**

- **Syntax :** `int wait ( int * status);`

- **Header files :**

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

DEMO

Process CREATION: fork() system call (1)

How can we modify `code1.c` of Lab 3 such that it displays "`Welcome to Madinah!`" twice, but with only one `printf ()` statement?

Code 1

```c
#include <stdio.h>

int main()
{
  printf ("Welcome to Madinah! \n");
  return 0;
}
```

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

int main ()
{
   fork();

   printf ("Welcome to Madinah! \n");

   return 0;
}

```

```
~$ gcc Code1.c
~$ ./a.out
Welcome to Madinah!
Welcome to Madinah!
~$
```

Process CREATION: fork() system call (2)

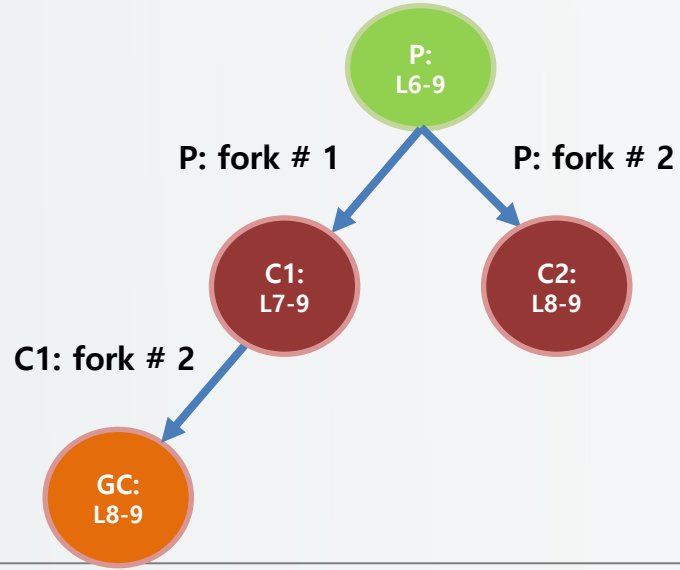
How many "Welcome to Madinah!" will be printed if we add a second `fork()` ?

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

int main ()
{
   fork();      /* fork # 1 */
   fork();      /* fork # 2 */
   printf ("Welcome to Madinah! \n");
   return 0;
}
```

```
~$ gcc Code1-2fork.c
~$ ./a.out
Welcome to Madinah!
Welcome to Madinah!
Welcome to Madinah!
Welcome to Madinah!
~$
```

The total number of **"Welcome to Madinah!"** is equal to the number of processes created which is 2^n where **n** is the number of **fork();**

Tree of processes with two `fork()`



Parent (P) – executes Line6-9, calls 2 fork and **printf**
Child1 (C1) – executes L7-9, calls 1 fork and **printf**
Child2 (C2) – executes L8-9, i.e. **printf**
GrandChild (GC) – executes L8-9, i.e. **printf**
Total = **4 printfs** are being executed

Process EXECUTION : exec() system call

`execlp()` allows a program to execute another executable from within the program that makes the call.

```
#include <unistd.h>

int execlp(const char *path, const char *arg0, ..., NULL);
```

```
1 #include <unistd.h>
2
3 int main(void) {
4   char *programName = "ls";
5   char *pathName = "/bin/ls";
6
7   execlp(pathName, programName, NULL);
8   return 0;
9 }
```

```
~$ ls
Code1          Code1.c  Q1a.c  Q2a.c  Q3.c  a.out
Code1-2fork.c  Q1a      Q1b.c  Q2b.c  Q4.c  demo-exec.c
~$ gcc demo-exec.c
~$ ./a.out
Code1          Code1.c  Q1a.c  Q2a.c  Q3.c  a.out
Code1-2fork.c  Q1a      Q1b.c  Q2b.c  Q4.c  demo-exec.c
~$ _
```

Note: Any statement after `execlp()` will **not** be executed if the call is successful.

Process IDENTIFICATION: getpid()

`getpid()` system call returns the process ID of the process that calls the function

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

void hello_function(void);

int main (void)
{
   hello_function();
   return 0;
}

void hello_function(void)
{
   pid_t pid;

   pid=fork();
   if (pid==0) {
      printf ("\n Hello I am the child with pid %d. \n",getpid());
   }
   else {
      printf ("\n Hello I am the parent with pid %d. \n",getpid());
   }
}
```

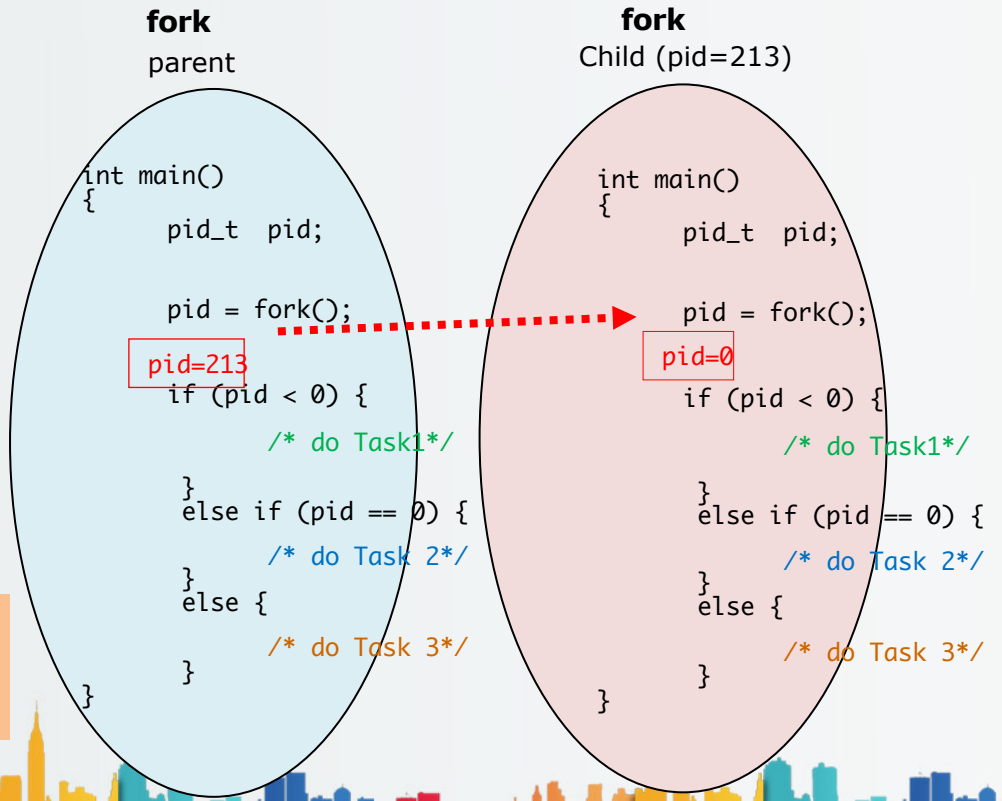
```
~$ gcc demo-getpid.c
~$ ./a.out

 Hello I am the parent with pid 993

 Hello I am the child with pid 994
~$
```

Task Assignments to Parent and Child via pid

Program: fork.c

```
int main()
{
      pid_t  pid;
      pid = fork(); /* fork another
    process */
    if (pid < 0) {/* error occurred *

        fprintf(stderr, "Fork Failed");
        exit(-1);

       }
    else if (pid == 0) { /* child
    process */

        execlp("/bin/ls", "ls", NULL);

    }
    else { /* parent process */

        /* parent will wait for the child to complete */
        wait (NULL);
        printf ("Child Complete");
        exit(0);

    }
}
```

We assign different tasks to parent and child by differentiating the pid return by fork()



Additional resources:

1. System Calls https://www.youtube.com/watch?v=IhToWeuWWfw
2. fork() and exec() System Calls https://www.youtube.com/watch?v=IFEFVXvjiHY
3. The fork() function in C https://www.youtube.com/watch?v=cex9XrZCU14
4. Waiting for processes to finish (using the wait function) in C https://www.youtube.com/watch?v=tcYo6hipaSA
5. Linux System Calls in Details https://www.geeksforgeeks.org/linux-system-call-in-detail/
6. The exec family of system calls http://www.it.uu.se/education/course/homepage/os/vt18/module-2/exec/