Department of Computer and Information Sciences
Towson University
Laboratory for Operating Systems Course

Process Creation and Management

Learning objective of this lab:

- To better understand process concept
- To become familiar with system calls dealing with process creation and management

Fork ( )

System call **fork()** is used to create processes. It takes no arguments and returns a process ID. The purpose of **fork()** is to create a ***new*** process, which becomes the *child* process of the caller. After a new child process is created, ***both*** processes will execute the next instruction following the ***fork()*** system call. Therefore, we have to distinguish the parent from the child. This can be done by testing the returned value of **fork()**:

- If **fork()** returns a negative value, the creation of a child process was unsuccessful.
- **fork()** returns a zero to the newly created child process.
- **fork()** returns a positive value, the ***process ID*** of the child process, to the parent. The returned process ID is of type **pid_t** defined in **sys/types.h**. Normally, the process ID is an integer. Moreover, a process can use function **getpid()** to retrieve the process ID assigned to this process.

Exec( )

int execlp(const char **file*, const char *arg, ..., (char *) 0) replaces the current process image with a new process image that will be loaded from *file*. The first argument *arg* must be the same as *file*.

Wait( )

int wait (int *status_location) forces a parent process to wait for a child process to stop or terminate. wait() return the pid of the child or -1 for an error. The exit status of the child is returned to status_location.

Waitpid( )

int waitpid (pid_t pid, int *status_location, int options) is more complicated that wait(). It suspends the parent process until a child process in its wait set terminates.

Exit( )

The **exit()** system call can be used to terminate a program at any point, no matter how many levels of function calls have been made. This is called with a return code. This function also calls a number of other functions which perform cleanup duties such as closing open files etc.

Perror()

System Calls provide a return value that signifies success or error. The C library, when a system call returns an error, writes a special error code into the global ***errno*** variable. The perror() translates the error code into human-readable format.

You can reference to the following links:
http://www.yolinux.com/TUTORIALS/ForkExecProcesses.html
https://ece.uwaterloo.ca/~dwharder/icsrts/Tutorials/fork_exec/

1. Write a simple C code, create a child process, and print out process IDs.
   You may use Cgwin to compile and run your code. Here is some sample commands:

```
% gcc exec.c //compile a c file named exec.c and generate a.out file
% ./a.out   //run the a.out file and see the output
```

2. After understanding how to use above functions, look at the following codes and find out what are the outputs?
   a)

```
int main( int argc, char *argv[], char *env[] )
{
          fork();
          printf("hello\n");


   return 0;
}
```

   b)

```
int main( int argc, char *argv[], char *env[] )
{
          int x =1;
          if(fork()==0)
                  printf("printf1: x=%d\n", ++x);
          printf("printf2: x=%d\n", --x);


   return 0;
}
```

3. Compile and run the program below. Complete the missing information in the diagram given after the code. Identify the parent process for P1. Identify the parent process for P2. What are the possible different outputs for this program?

```
int main()
{
 int pid;
 pid = fork();
 if (pid == -1)
 {
    perror("ERROR - fork \n");
    exit(0);
 }
 else if (pid == 0)
{
      printf("Child process with pid  = %d, and Parent pid = %d\n",
      getpid(), getppid());
      sleep(5);
      printf("After sleeping.  Child process with pid  = %d, and Parent
      pid = %d\n", getpid(), getppid());
      exit(0);
 }
 else {
     printf("Parent process with pid  = %d, and Parent pid = %d\n",
     getpid(), getppid());
     printf("Parent exiting now\n");
     exit(0);
     }
  return 0;
 }
```