CS 288 2018S Section 006
Process Control System Calls

Processes

In the context of the UNIX operating system and its clones, a new process can come into existence only by use of the *fork* system call:

```
processid = fork()
```

When fork is executed, the process splits into two independently executing processes, both of which continue to run. The two processes have independent copies of the original memory image, and share all open files. The only difference between the two is the value returned by *fork*, the *processid*. In one process, the child, the returned value is always 0. In the other process, the parent, the returned *processid* identifies the child process and is never 0. If there is any error, *fork* returns `-1`.

Because the values returned by *fork* in the parent and child process are distinguishable, each process may determine whether it is the parent or child.

Execution of Programs

Another major system call is invoked by:

```
execl(filepath, arg0, arg1, arg2, ..., argn, NULL)
```

which overlays the calling process with the named file, then transfers control to the file's entry point.

All the code and data in the process invoking execute is replaced from the file, but open files, and current directory are unaltered. There can be no return from the file; the calling program is lost unless there is an error and the call fails, for example if the file could not be found or is not runnable.

The first argument to *execl* is the filename of the command, with the pathname. The second and subsequent arguments are the command name and the arguments of the command; these become the *argv* array of the new program. The end of the list is marked by *NULL*.

Process Synchronization

Another system call:

```
processid = wait(&status)
```

causes its caller to suspend execution until one of its children has completed execution. Then *wait* returns the *processid* of the terminated process. An error return is taken if the calling process has no descendants. Exit status from the child process is also available.

Termination

Lastly,

```
exit(status)
```

terminates a process, closes its open files, and generally obliterates it. The parent is notified through the *wait* system call, and the value *status* is made available to it.

Source: D. M. Ritchie and K. Thompson, "The UNIX Time-Sharing System", Bell System Tech. J. 57 6, pp. 1905–1929, July-August 1978.