HOW LINUX CREATES PROCESSES

Linux creates every process using the `fork(2)` or `clone(2)` syscalls. The only way to create a process is to fork your current process, and replace the executable image using the `exec(2)` syscall with the executable image of the process you want to run. Here's how it works:

1. A process (e.g. bash) wants to run another process, such as "ls" (to list files and directories)
2. The process (bash) forks itself using the `fork(2)` or `clone(2)` syscall
3. Forked processes are basically exact copies of their parent process, and resume execution at the exact same spot. Therefore you normally check if you're the forked process by checking the return code of the `fork(2)` or `clone(2)` syscall. It'll return different values for the parent and the child process. The parent will then go on to do something like call `wait(2)` to wait for the child process to complete execution
4. The child process inherits pipes, file descriptors, state, etc from the parent process, but a lot of that isn't needed now so the child process may clean up by closing open file descriptors and sockets, etc

5. The child process calls `execve(2)` or another `exec(2)` syscall to replace itself with the target process. A few things about the `exec(2)` syscalls. They replace the current process with another process. The text, data and stack of the calling process are overwritten by the new process. However, most process attributes are preserved, such as the stdin/stdout/stderr file descriptors (so a child process could set these before running `exec(2)`. File descriptors are kept open in general, unless they are set to close-on-exec.

Here's a code example that does the same thing:

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

int main(void)
{
  int pid = fork();

  if (pid == -1) {
    fprintf(stderr, "Could not fork process\n");
    return -1;
  } else if (pid == 0) {
    fprintf(stdout, "Child will now replace itself with ls\n");

    // Setup the arguments/environment to call
```

```
    char *argv[] = { "/bin/ls", "-la", 0 };
    char *envp[] = { "HOME=/", "PATH=/bin:/usr/bin", "USER=brandon", 0 };

    // Call execve(2) which will replace the executable image of this
    // process
    execve(argv[0], &argv[0], envp);

    // Execution will never continue in this process unless execve returns
    // because of an error
    fprintf(stderr, "Oops!\n");
    return -1;
  } else if (pid > 0) {
    int status;

    fprintf(stdout, "Parent will now wait for child to finish execution\n");
    wait(&status);
    fprintf(stdout, "Child has finished execution (returned %i), parent is done\n", s
tatus);
  }

  return 0;
}
```

The above code will fork the process, and run ls, with some helpful output to see whats going on. Output should look like this:

```
Parent will now wait for child to finish execution

Child will now replace itself with ls

total 24

drwxrwxr-x 2 brandon brandon 4096 Aug 7 11:09 .

drwxr-xr-x 53 brandon brandon 4096 Aug 7 11:09 ..

-rwxrwxr-x 1 brandon brandon 8805 Aug 7 11:08 test

-rw-rw-rw- 1 brandon brandon 1038 Aug 7 11:08 test.c

Child has finished execution (returned 0), parent is done
```

Source: http://brandonwamboldt.ca/how-linux-creates-processes-1528/