Why is the default process creation mechanism fork?

The UNIX system call for process creation, fork(), creates a child process by copying the parent process. My understanding is that this is almost always followed by a call to exec() to replace the child process' memory space (including text segment). Copying the parent's memory space in fork() always seemed wasteful to me (although I realize the waste can be minimized by making the memory segments copy-on-write so only pointers are copied). Anyway, does anyone know why this duplication approach is required for process creation?

/ process / process-management / fork

asked Feb 7 at 19:13

espertus

235 4

2 Note that the `fork(2)` man page under Linux says: `Under Linux, fork() is implemented using copy-on-write pages, so the only penalty that it incurs is the time and memory required to duplicate the parent's page tables, and to create a unique task structure for the child.` I imagine (but do not know for certain) that this is the case for other modern Unix flavors. – larsks Feb 7 at 19:17

2 The original, PDP-11 Unix realy, truly did copy all the bytes of a forked process: but it only had 64Kb of executable, and at most 64Kb of data, so it wasn't a huge burden, even in 1975. I would guess that EVERY unix and unix-a-like since about 1990 has had copy-on-write text segments, so I'm not even sure why books and articles propagate "performance problem with fork" any more. – Bruce Ediger Feb 7 at 19:52

Nowadays, fork is implemented in a similar fashion to vfork (openbsd.org/cgi-bin/…). It's efficient, don't worry. – Aki Feb 7 at 20:27

Also note that there are lots of usage where you do not exec after a fork (or at least, does not exec right away): think of pipes and web servers. – jfgagne Feb 13 at 18:34

feedback

2 Answers

It's to simplify the interface. The alternative to `fork` and `exec` would be something like Windows' CreateProcess function. Notice how many parameters `CreateProcess` has, and many of them are structs with even more parameters. This is because *everything* you might want to control about the new process has to be passed to `CreateProcess`. In fact, `CreateProcess` doesn't have enough parameters, so Microsoft had to add CreateProcessAsUser and CreateProcessWithLogonW.

With the `fork/exec` model, you don't need all those parameters. Instead, certain attributes of the process are preserved across `exec`. This allows you to `fork`, then change whatever process attributes you want (using the same functions you'd use normally), and *then* `exec`. In Linux, `fork` has no parameters, and `execve` has only 3: the program to run, the command line to give it, and its environment. (There are other `exec` functions, but they're just wrappers around `execve` provided by the C library to simplify common use cases.)

If you want to start a process with a different current directory: `fork` , `chdir` , `exec` .

If you want to redirect stdin/stdout: `fork` , close/open files, `exec` .

If you want to switch users: `fork` , `setuid` , `exec` .

All these things can be combined as needed. If somebody comes up with a new kind of process attribute, you don't have to change `fork` and `exec` .

As larsks mentioned, most modern Unixes use copy-on-write, so `fork` doesn't involve significant overhead.

edited Feb 7 at 20:01 answered Feb 7 at 19:31

cjm
5,539 1 9 17

In fact, don't honest comparisons of fork vs CreateThread usually show that fork is competitive with CreateThread? – Bruce Ediger Feb 7 at 19:53

8 Excellent explanation. "Those who don't understand UNIX are condemned to reinvent it, poorly." -- Henry Spencer – Kyle Jones Feb 7 at 19:57

Thanks! Do you have a reference, by any chance? – espertus Feb 7 at 20:07

1 @Aki, nope, CreateProcess() literally makes a new process and builds it up from scratch, no forking. – psusi Feb 7 at 23:19

1 But must there not be some equivalent of CreateProcess() somewhere in Unix? Otherwise how does the very first process get created? Unlike a mythological creator god, the first process cannot fork() itself from nothingness. ;-) – Steven Monday Feb 8 at 3:54

feedback

In adition to the cjm's answer, the Single Unix Specification defines a function named `vfork()` . That function works like fork, except that the forked process has undefined behavior if it does anything other than try calling an exec familly function, or calling `_exit()` .

Thus pretty much the only use with defined behavior is:

```
pid_t ret = vfork();
if(ret == 0)
{
    exec(...);
    _exit(EXIT_FAILURE); //in case exec failed for any reason.
}
```

So what does `vfork` do? It is is an inexpensive `fork` . In implemenations without copy-on-write, the resulting process will share memory space with the original process (hence the undefined behavior). In implementations with copy-on-write, `vfork` is permitted to be identical to `fork()` , since copy-on-write implementations are fast.

There is also the optional `posix_spawn` function (and a `posix_spawnp` function) which can

directly create a new process. (It is also permissible to implement them with a library call using `fork` and `exec`, and an example implementation is provided.)

answered Feb 7 at 21:09

Kevin Cathcart
166 1

feedback

---

question feed