fork() or *vfork()* is followed by an *exec()*. This is illustrated by the final pair of data rows in Table 28-3, where each child performs an *exec()*, rather than immediately exiting. The program execed was the *true* command (`/bin/true`, chosen because it produces no output). In this case, we see that the relative differences between *fork()* and *vfork()* are much lower.

> In fact, the data shown in Table 28-3 doesn't reveal the full cost of an *exec()*, because the child execs the same program in each loop of the test. As a result, the cost of disk I/O to read the program into memory is essentially eliminated, because the program will be read into the kernel buffer cache on the first *exec()*, and then remain there. If each loop of the test execed a different program (e.g., a differently named copy of the same program), then we would observe a greater cost for an *exec()*.

28.4 Effect of *exec()* and *fork()* on Process Attributes

A process has numerous attributes, some of which we have already described in earlier chapters, and others that we explore in later chapters. Regarding these attributes, two questions arise:

- What happens to these attributes when a process performs an *exec()*?
- Which attributes are inherited by a child when a *fork()* is performed?

Table 28-4 summarizes the answers to these questions. The *exec()* column indicates which attributes are preserved during an *exec()*. The *fork()* column indicates which attributes are inherited (or in some cases, shared) by a child after *fork()*. Other than the attributes indicated as being Linux-specific, all listed attributes appear in standard UNIX implementations, and their handling during *exec()* and *fork()* conforms to the requirements of SUSv3.

Table 28-4: Effect of *exec()* and *fork()* on process attributes

Process attribute	*exec()*	*fork()*	Interfaces affecting attribute; additional notes
Process address space			
Text segment	No	Shared	Child process shares text segment with parent.
Stack segment	No	Yes	Function entry/exit; *alloca()*, *longjmp()*, *siglongjmp()*.
Data and heap segments	No	Yes	*brk()*, *sbrk()*.
Environment variables	See notes	Yes	*putenv()*, *setenv()*; direct modification of *environ*. Overwritten by *execle()* and *execve()* and preserved by remaining *exec()* calls.
Memory mappings	No	Yes; see notes	*mmap()*, *munmap()*. A mapping's `MAP_NORESERVE` flag is inherited across *fork()*. Mappings that have been marked with *madvise(MADV_DONTFORK)* are not inherited across *fork()*.
Memory locks	No	No	*mlock()*, *munlock()*.

Table 28-4: Effect of *exec()* and *fork()* on process attributes (continued)

Process attribute	*exec()*	*fork()*	Interfaces affecting attribute; additional notes
Process identifiers and credentials			
Process ID	Yes	No	
Parent process ID	Yes	No	
Process group ID	Yes	Yes	*setpgid()*.
Session ID	Yes	Yes	*setsid()*.
Real IDs	Yes	Yes	*setuid()*, *setgid()*, and related calls.
Effective and saved set IDs	See notes	Yes	*setuid()*, *setgid()*, and related calls. Chapter 9 explains how *exec()* affects these IDs.
Supplementary group IDs	Yes	Yes	*setgroups()*, *initgroups()*.
Files, file I/O, and directories			
Open file descriptors	See notes	Yes	*open()*, *close()*, *dup()*, *pipe()*, *socket()*, and so on. File descriptors are preserved across *exec()* unless marked close-on-exec. Descriptors in child and parent refer to same open file descriptions; see Section 5.4.
Close-on-exec flag	Yes (if off)	Yes	*fcntl(F_SETFD)*.
File offsets	Yes	Shared	*lseek()*, *read()*, *write()*, *readv()*, *writev()*. Child shares file offsets with parent.
Open file status flags	Yes	Shared	*open()*, *fcntl(F_SETFL)*. Child shares open file status flags with parent.
Asynchronous I/O operations	See notes	No	*aio_read()*, *aio_write()*, and related calls. Outstanding operations are canceled during an *exec()*.
Directory streams	No	Yes; see notes	*opendir()*, *readdir()*. SUSv3 states that child gets a copy of parent's directory streams, but these copies may or may not share the directory stream position. On Linux, the directory stream position is not shared.
File system			
Current working directory	Yes	Yes	*chdir()*.
Root directory	Yes	Yes	*chroot()*.
File mode creation mask	Yes	Yes	*umask()*.
Signals			
Signal dispositions	See notes	Yes	*signal()*, *sigaction()*. During an *exec()*, signals with dispositions set to default or ignore are unchanged; caught signals revert to their default dispositions. See Section 27.5.
Signal mask	Yes	Yes	Signal delivery, *sigprocmask()*, *sigaction()*.
Pending signal set	Yes	No	Signal delivery; *raise()*, *kill()*, *sigqueue()*.
Alternate signal stack	No	Yes	*sigaltstack()*.

(continued)

Table 28-4: Effect of *exec()* and *fork()* on process attributes (continued)

Process attribute	*exec()*	*fork()*	Interfaces affecting attribute; additional notes
Timers			
Interval timers	Yes	No	*setitimer()*.
Timers set by *alarm()*	Yes	No	*alarm()*.
POSIX timers	No	No	*timer_create()* and related calls.
POSIX threads			
Threads	No	See notes	During *fork()*, only calling thread is replicated in child.
Thread cancelability state and type	No	Yes	After an *exec()*, the cancelability type and state are reset to `PTHREAD_CANCEL_ENABLE` and `PTHREAD_CANCEL_DEFERRED`, respectively
Mutexes and condition variables	No	Yes	See Section 33.3 for details of the treatment of mutexes and other thread resources during *fork()*.
Priority and scheduling			
Nice value	Yes	Yes	*nice()*, *setpriority()*.
Scheduling policy and priority	Yes	Yes	*sched_setscheduler()*, *sched_setparam()*.
Resources and CPU time			
Resource limits	Yes	Yes	*setrlimit()*.
Process and child CPU times	Yes	No	As returned by *times()*.
Resource usages	Yes	No	As returned by *getrusage()*.
Interprocess communication			
System V shared memory segments	No	Yes	*shmat()*, *shmdt()*.
POSIX shared memory	No	Yes	*shm_open()* and related calls.
POSIX message queues	No	Yes	*mq_open()* and related calls. Descriptors in child and parent refer to same open message queue descriptions. A child doesn't inherit its parent's message notification registrations.
POSIX named semaphores	No	Shared	*sem_open()* and related calls. Child shares references to same semaphores as parent.
POSIX unnamed semaphores	No	See notes	*sem_init()* and related calls. If semaphores are in a shared memory region, then child shares semaphores with parent; otherwise, child has its own copy of the semaphores.
System V semaphore adjustments	Yes	No	*semop()*. See Section 47.8.
File locks	Yes	See notes	*flock()*. Child inherits a reference to the same lock as parent.
Record locks	See notes	No	*fcntl(F_SETLK)*. Locks are preserved across *exec()* unless a file descriptor referring to the file is marked close-on-exec; see Section 55.3.5.

Table 28-4: Effect of *exec()* and *fork()* on process attributes (continued)

Process attribute	*exec()*	*fork()*	Interfaces affecting attribute; additional notes
Miscellaneous			
Locale settings	No	Yes	*setlocale()*. As part of C run-time initialization, the equivalent of *setlocale(LC_ALL, "C")* is executed after a new program is execed.
Floating-point environment	No	Yes	When a new program is execed, the state of the floating-point environment is reset to the default; see *fenv(3)*.
Controlling terminal	Yes	Yes	
Exit handlers	No	Yes	*atexit()*, *on_exit()*.
Linux-specific			
File-system IDs	See notes	Yes	*setfsuid()*, *setfsgid()*. These IDs are also changed any time the corresponding effective IDs are changed.
timerfd timers	Yes	See notes	*timerfd_create()*; child inherits file descriptors referring to same timers as parent.
Capabilities	See notes	Yes	*capset()*. The handling of capabilities during an *exec()* is described in Section 39.5.
Capability bounding set	Yes	Yes	
Capabilities *securebits* flags	See notes	Yes	All *securebits* flags are preserved during an *exec()* except `SECBIT_KEEP_CAPS`, which is always cleared.
CPU affinity	Yes	Yes	*sched_setaffinity()*.
`SCHED_RESET_ON_FORK`	Yes	No	See Section 35.3.2.
Allowed CPUs	Yes	Yes	See *cpuset(7)*.
Allowed memory nodes	Yes	Yes	See *cpuset(7)*.
Memory policy	Yes	Yes	See *set_mempolicy(2)*.
File leases	Yes	See notes	*fcntl(F_SETLEASE)*. Child inherits a reference to the same lease as parent.
Directory change notifications	Yes	No	The *dnotify* API, available via *fcntl(F_NOTIFY)*.
prctl(PR_SET_DUMPABLE)	See notes	Yes	During an *exec()*, the `PR_SET_DUMPABLE` flag is set, unless execing a set-user-ID or set-group-ID program, in which case it is cleared.
prctl(PR_SET_PDEATHSIG)	Yes	No	
prctl(PR_SET_NAME)	No	Yes	
`oom_adj`	Yes	Yes	See Section 49.9.
`coredump_filter`	Yes	Yes	See Section 22.1.

28.5 Summary

When process accounting is enabled, the kernel writes an accounting record to a file for each process that terminates on the system. This record contains statistics on the resources used by the process.

Like *fork()*, the Linux-specific *clone()* system call creates a new process, but allows finer control over which attributes are shared between the parent and child. This system call is used primarily for implementing threading libraries.

We compared the speed of process creation using *fork()*, *vfork()*, and *clone()*. Although *vfork()* is faster than *fork()*, the time difference between these system calls is small by comparison with the time required for a child process to do a subsequent *exec()*.

When a child process is created via *fork()*, it inherits copies of (or in some cases shares) certain process attributes from its parent, while other process attributes are not inherited. For example, a child inherits copies of its parent's file descriptor table and signal dispositions, but doesn't inherit its parent's interval timers, record locks, or set of pending signals. Correspondingly, when a process performs an *exec()*, certain process attributes remain unchanged, while others are reset to defaults. For example, the process ID remains the same, file descriptors remain open (unless marked close-on-exec), interval timers are preserved, and pending signals remain pending, but handled signals are reset to their default disposition and shared memory segments are detached.

Further information

Refer to the sources of further information listed in Section 24.6. Chapter 17 of [Frisch, 2002] describes the administration of process accounting, as well as some of the variations across UNIX implementations. [Bovet & Cesati, 2005] describes the implementation of the *clone()* system call.

28.6 Exercise

28-1. Write a program to see how fast the *fork()* and *vfork()* system calls are on your system. Each child process should immediately exit, and the parent should *wait()* on each child before creating the next. Compare the relative differences for these two system calls with those of Table 28-3. The shell built-in command *time* can be used to measure the execution time of a program.