A fork() in the road

Andrew Baumann
Microsoft Research

Jonathan Appavoo
Boston University

Orran Krieger
Boston University

Timothy Roscoe
ETH Zurich
NAME

fork - create a new process

SYNOPSIS

#include <unistd.h>

pid_t fork(void);

DESCRIPTION

The *fork()* function shall create a new process. The new process (child process) shall be **an exact copy** of the calling process (parent process) except as detailed below:
Motivation

• We’ve first-hand experience of many research OSes
  L4, Wanda, Nemesis, Mungi, Hurricane, Tornado, K42, Barrelfish, Drawbridge ...
• Supporting fork, or choosing not to, repeatedly tied our hands
• This is common wisdom among those who have built non-Unix OSes
• And yet...
Motivation

Problem set 4: WeensyOS

In this assignment, you implement process memory isolation, virtual memory, and some system calls in a tiny operating system. This will introduce you to virtual memory and operating system design.

... 

Step 5: Fork

The fork system call is one of Unix’s great ideas. It starts a new process as a copy of an existing process. The fork system call appears to return twice, once to each process. To the child process, it returns 0. To the parent process, it returns the child’s process ID.
Motivation

5.4 Why? Motivating The API

Of course, one big question you might have: why would we build such an odd interface to what should be the simple act of creating a new process? Well, as it turns out, the separation of fork() and exec() is essential in building a UNIX shell, because it lets the shell run code after the call to fork() but before the call to exec(); this code can alter the environment of the about-to-be-run program, and thus enables a variety of interesting features to be readily built.

TIP: GETTING IT RIGHT (LAMPSON’S LAW)
As Lampson states in his well-regarded “Hints for Computer Systems Design” [L83], “Get it right. Neither abstraction nor simplicity is a substitute for getting it right.” Sometimes, you just have to do the right thing, and when you do, it is way better than the alternatives. There are lots of ways to design APIs for process creation; however, the combination of fork() and exec() are simple and immensely powerful. Here, the UNIX designers simply got it right. And because Lampson so often “got it right”, we name the law in his honor.
Why do people like fork?

• It’s simple: no parameters!
  • cf. Win32 CreateProcess

• It’s elegant: fork is orthogonal to exec
  • System calls that a process uses on itself also initialise a child
  • e.g. shell modifies FDs prior to exec

• It eased concurrency
  • Especially in the days before threads and async I/O
Fork today

• Fork is no longer simple
• Fork doesn’t compose
• Fork isn’t thread-safe
• Fork is insecure
• Fork is slow
• Fork doesn’t scale
• Fork encourages memory overcommit
• Fork is incompatible with a single address space
• Fork is incompatible with heterogeneous hardware
• Fork infects an entire system
Fork doesn’t compose

• Fork creates a process by cloning another
• Where is the state of a process?
  • In classic Unix:
    • CPU context
    • Address space
    • File descriptor table
  • Today:
    • User-mode libraries
    • Threads
    • Server processes
    • Hardware accelerator context
• Every component must support fork
  • Many don’t → undefined behaviour

Who would accept fork() today?
Fork is slow

![Graph showing the time taken for fork and exec operations against parent process size. The graph compares fork + exec (fragmented), fork + exec (dirty), and posix_spawn methods.](image)

- **fork + exec (fragmented)**
- **fork + exec (dirty)**
- **posix_spawn**
Fork infects a system: the K42 experience

• Scalable multiprocessor OS, developed at IBM Research
• Object-oriented kernel and libraries
  • Separation of concerns between files, memory management, etc.
  • Multiple implementations (e.g. single-core, scalable)
• Aimed to support multiple OS personalities
  • However, competitive Linux performance demanded efficient fork...
• Efficient fork requires:
  • Centralised state → lack of modularity, poor scalability
  • Lazy copying → complex object relationships
• Result: every interface, and every object, must support fork
  • Made a mess of the abstractions
  • Led to abandoning other OS personalities
So Ken, where did fork come from anyway?
Origins of fork

Unix designers credit Project Genie (Berkeley, 1964-68)

“The fork operation, essentially as we implemented it, was present in the GENIE time-sharing system”

[Ritchie & Thompson, CACM 1974]
Project Genie aka SDS 940

3. FORKS AND JOBS

CREATION OF FORKS

A fork may create new, dependent, entries in the PAC table by executing BRS 9. This BRS takes its argument in the A register, which contains the address of a seven-word panic table with the format given in Table 2.

<table>
<thead>
<tr>
<th>Word</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Program counter</td>
</tr>
<tr>
<td>1</td>
<td>A register</td>
</tr>
<tr>
<td>2</td>
<td>B register</td>
</tr>
<tr>
<td>3</td>
<td>X register</td>
</tr>
<tr>
<td>4</td>
<td>First relabeling register</td>
</tr>
<tr>
<td>5</td>
<td>Second relabeling register</td>
</tr>
<tr>
<td>6</td>
<td>Status</td>
</tr>
</tbody>
</table>

Table 2. Panic Table

The status word may be

-2 Dismissed for input/output
-1 Running
0 Dismissed on escape or BRS 10
1 Dismissed on Illegal Instruction panic
2 Dismissed on memory panic

The panic table address must not be the same for two dependent forks of the same fork, or overlap a page boundary. If it is, BRS 9 is illegal. The first six bits of the A register have the following significance as shown in Figure 3.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Significance</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Make fork Executive if current fork is Executive.</td>
</tr>
<tr>
<td>1</td>
<td>Set fork relabeling from panic table. Otherwise, use current relabeling.</td>
</tr>
<tr>
<td>2</td>
<td>Propagate escape assignment to fork (see BRS 90).</td>
</tr>
<tr>
<td>3</td>
<td>Make fork fixed memory. It is not allowed to obtain any more memory than it is started with.</td>
</tr>
<tr>
<td>4</td>
<td>Make fork local memory. New memory will be assigned to it independently of the controlling fork.</td>
</tr>
<tr>
<td>5</td>
<td>Make fork Exec type 1 if current fork is Exec.</td>
</tr>
</tbody>
</table>

Figure 3. Significance of Bits in A Register

When BRS 9 is executed, a new entry in the PAC table is created. This new fork is said to be a fork of the fork creating it. This is called the controlling fork. The fork is said to be lower in the hierarchy of forks than the controlling fork. The latter may itself be a fork of some still higher fork. A job may have a maximum of eight forks including the executive. The A, B and X registers for the fork are set up from the current contents of the panic table. The address at which execution of the fork is to be started is also taken from the panic table. The relabeling registers are set up either from the current contents of the panic table or from the relabeling registers of the currently running program. An executive fork may change the relabeling. A user fork is restricted to changing relabeling in the manner permitted by BRS 44. The status word is set to -1 by BRS 9. The fork number that is assigned is kept in PIM. This number is an index to the fork parameters kept in the TS block.

The fork structure is kept track of by pointers in PACT. For each fork PFORK points to the controlling fork, PDOWN to one of the subsidiary forks, and PPAR to a fork on the same level. All the subsidiary forks of a single fork are chained in a list.
For implementation expedience [Ritchie, 1979]

- fork was 27 lines of PDP-7 assembly
- One process resident at a time
- Copy parent’s memory out to swap
- Continue running child
- exec didn’t exist – it was part of the shell
- Would have been more work to combine them
Fork was a hack!

• Fork is not an inspired design, but an accident of history
• Only Unix implemented it this way
• We may be stuck with fork for a long time to come
• But, let’s not pretend that it’s still a good idea today!
Get the fork out of my OS!

• Deprecate fork!
• Improve the alternatives
  • posix_spawn(), cross-process APIs
• Please, stop teaching students that fork is good design
  • Begin with spawn
  • Teach fork, but include historical context

• See our paper for:
  • Alternatives to fork, specific use cases, war stories, and more