

Home

PUBLIC

Stack Overflow

Tags

Users

Jobs

Teams

Q&amp;A for work

Learn More

# fork() system call and memory space of the process

Ask Question

11

I quote "when a process creates a new process using fork() call, Only the shared memory segments are shared between the parent process and the newly forked child process. Copies of the stack and the heap are made for the newly created process" from "operating system concepts" solutions by Silberschatz.

But when I tried this program out

8

```
#include <stdio.h>
#include <sys/types.h>

#define MAX_COUNT 200

void ChildProcess(void); /* child process prototype */
void ParentProcess(void); /* parent process prototype */

void main(void)
{
    pid_t pid;
    char * x=(char *)malloc(10);

    pid = fork();
    if (pid == 0)
        ChildProcess();
    else
        ParentProcess();
    printf("the address is %p\n",x);
}

void ChildProcess(void)
{
    printf(" *** Child process ***\n");
}

void ParentProcess(void)
{
    printf("*** Parent*****\n");
}
```

the result is like:

```
*** Parent*****
the address is 0x1370010
*** Child process ***
the address is 0x1370010
```

both parent and child printing the same address which is in heap.

asked 4 years, 4 months ago

viewed 6,767 times

active 3 months ago

Blog

Predicting Stack Overflow Tags with Google's Cloud AI

Featured on Meta

Unicom Meta Zoo #2: What is the role of moderators?

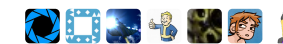
Should we burninate the [bioinformatics] tag?

Introducing a new option for Private Q&amp;A featuring single sign-on and advanced features for

55 people chatting

C++ Questions and Answers

14 mins ago - Mgetz



C

9 hours ago - Kamiccio

share improve this question

add a comment

asked Dec 15 '14 at 15:00



gman

642 1 6 24

## 6 Answers

active oldest votes

Quoting myself from another thread.

25



- When a `fork()` system call is issued, a copy of all the pages corresponding to the parent process is created, loaded into a separate memory location by the OS for the child process. But this is not needed in certain cases. Consider the case when a child executes an "exec" system call or exits very soon after the `fork()`. When the child is needed just to execute a command for the parent process, there is no need for copying the parent process' pages, since exec replaces the address space of the process which invoked it with the command to be executed.

In such cases, a technique called copy-on-write (COW) is used. With this technique, when a fork occurs, the parent process's pages are not copied for the child process. Instead, the pages are shared between the child and the parent process. Whenever a process (parent or child) modifies a page, a separate copy of that particular page alone is made for that process (parent or child) which performed the modification. This process will then use the newly copied page rather than the shared one in all future references. The other process (the one which did not modify the shared page) continues to use the original copy of the page (which is now no longer shared). This technique is called copy-on-write since the page is copied when some process writes to it.

- Also, to understand why these programs appear to be using the same space of memory (which is not the case), I would like to quote a part of the book "Operating Systems: Principles and Practice".

Most modern processors introduce a level of indirection, called virtual addresses. With virtual addresses, every process's memory starts at the "same" place, e.g., zero. Each process thinks that it has the entire machine to itself, although obviously that is not the case in reality.

So these virtual addresses are translations of physical addresses and doesn't represent the same physical memory space, to leave a more practical example we can do a test, if we compile and run multiple times a program that displays the direction of a static variable, such as this program.

```
#include <stdio.h>

int main() {
    static int a = 0;

    printf("%p\n", &a);

    getchar();
}
```

## Linked

0 Parallel processes not sharing same data

## Related

6 Difference between the address space of parent process and its child process in Linux?

183 The difference between `fork()`, `vfork()`, `exec()` and `clone()`

9 what happens to pointers to dynamically allocated memory after a UNIX fork?

1 address space after fork call

0 `fork()` in operating system3 `fork()` and `wait()` calls

0 Same addresses pointing to different values - fork system call

8 Is it possible to fork a process without inherit virtual memory space of parent process?

0 How to use memory when calling `fork()`

-3 Pid after fork in C

## Hot Network Questions

Would encrypting a database protect against a compromised admin account?

How did Captain Marvel know where to find these characters?

 Getting a error after using `setState` with a promise `date -d 'previous Monday'` to display the preceding Monday

Was Mohammed the most popular first name for boys born in Berlin in 2018?

What was the plan for an abort of the Enola Gay's mission to drop the atomic bomb?

Program for finding longest run of zeros from a list of 100 random integers which are either 0 or 1

Was the Highlands Ranch shooting the 115th mass shooting in the US in 2019

How is CoreiX like Corei5, i7 is related to Haswell, Ivy Bridge?

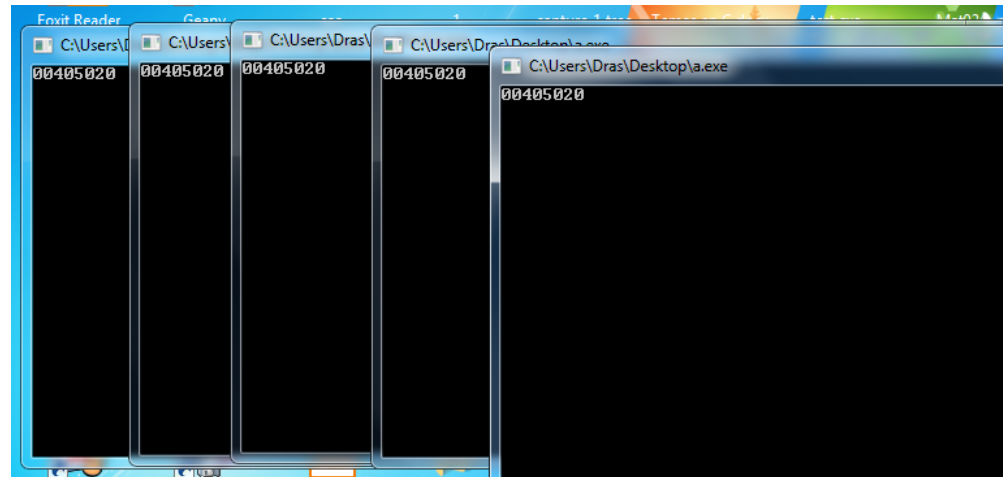
is it permitted to swallow spit on a fast day?

Why use steam instead of just hot air?

```
return 0;
}
```

It would be impossible to obtain the same memory address in two different programs if we deal with the physical memory directly.

And the results obtained from running the program several times are...



share improve this answer

edited Dec 16 '14 at 23:24

answered Dec 15 '14 at 15:18



Patricio Sard

1,069 ● 2 ● 12 ● 33

1 Excellent explanation! Thanks. Now I got a book to read :) – Solidak Apr 30 '18 at 9:56

1 Unfortunately, though, I was not able to reproduce your experiment. I ran the same program on an ArchLinux x64 machine and a MacOS High Sierra machine: both yielded very different address spaces every time I ran them – Solidak Apr 30 '18 at 10:10

Everything can be acceptable up to the image results. It is not certain, that is to say, it is subject to the OS. Glance at the following image -> [i.stack.imgur.com/7sod0.gif](https://i.stack.imgur.com/7sod0.gif) – snr Apr 15 at 13:22

add a comment



Yes, both processes are using the same address for this variable, but these addresses are used by different processes, and therefore aren't in the same [virtual address space](#).

10



This means that the addresses are the same, but they aren't pointing to the same physical memory. You should read more about virtual memory to understand this.

share improve this answer

answered Dec 15 '14 at 15:09



ElderBug

4,388 ● 10 ● 21

4 As an optimization technique called [copy-on-write](#) though, the virtual memory starts out as pointing to the same physical memory after `fork()`, but the memory will be copied and the mappings will be changed if one of the processes writes to the memory. – nos Dec 15 '14 at 15:20

Why did they go to Dragonstone?

Two researchers want to work on the same extension to my paper. Who to help?

How to handle DM constantly stealing everything from sleeping characters?

Exception propagation

Is it bad writing or bad story telling if first person narrative contains more information than the narrator knows?

Can 'sudo apt-get remove [write]' destroy my Ubuntu?

We are two immediate neighbors who forged our own powers to form concatenated relationship. Who are we?

A Cunning Riley Riddle

How did Arya survive this confrontation unscathed?

What is the name of meteoroids which hit Moon, Mars, or pretty much anything that isn't the Earth?

Renting a house to a graduate student in my department

Is it a Munchausen Number?

Rotate shape around defined point instead of centroid in QGIS 3

question feed

[add a comment](#)

5

You're probably running your program on an operating system with virtual memory. After the `fork()` call, the parent and child have separate address spaces, so the address `0x1370010` is not pointing to the same place. If one process wrote to `*x`, the other process would not see the change. (In fact those may be the same page of memory, or even the same block in a swap-file, until it's changed, but the OS makes sure that the page is copied as soon as either the parent or the child writes to it, so as far as the program can tell it's dealing with its own copy.)

[share](#) [improve this answer](#)

answered Dec 15 '14 at 15:10



david

854 ● 5 ● 13

[add a comment](#)

4

The address is the same, but the address space is not. Each process has its own address space, so parent's `0x1370010` is not the same as child's `0x1370010`.

[share](#) [improve this answer](#)

answered Dec 15 '14 at 15:09



Sergej Christoforov

1,153 ● 6 ● 5

[add a comment](#)

4

When the kernel `fork()` s the process, the copied memory information inherits the same address information since the heap is effectively copied as-is. If addresses were different, how would you update pointers inside of custom structs? The kernel knows nothing about that information so those pointers would then be invalidated. Therefore, the *physical* address may change (and in fact often will change even during the lifetime of your executable even without `fork()` ing, but the *logical* address remains the same.

[share](#) [improve this answer](#)

answered Dec 15 '14 at 15:10



inetknight

2,270 ● 1 ● 16 ● 39

[add a comment](#)

1

Yes address in both the case is same. But if you assign different value for `x` in child process and parent process and then also prints the value of `x` along with address of `x`, You will get your answer.

```
#include <stdio.h>
#include <sys/types.h>
#include <stdlib.h>
#include <unistd.h>
#define MAX_COUNT 200

void ChildProcess(void);          /* child process prototype */
void ParentProcess(void);        /* parent process prototype */

void main(void)
```

```

{
    pid_t pid;
    int * x = (int *)malloc(10);

    pid = fork();
    if (pid == 0) {
        *x = 100;
        ChildProcess();
    }
    else {
        *x = 200;
        ParentProcess();
    }
    printf("the address is %p and value is %d\n", x, *x);
}

void ChildProcess(void)
{
    printf("    *** Child process    ***\n");
}

void ParentProcess(void)
{
    printf("*** Parent*****\n");
}

```

Output of this will be:

```

*** Parent*****
the address is 0xf70260 and value is 200
*** Child process    ***
the address is 0xf70260 and value is 100

```

Now, You can see that value is different but address is same. So The address space for both the process is different. These addresses are not actual address but logical address so these could be same for different processes.

share improve this answer

answered Jan 15 at 13:30




SJ26  
23 ● 5


add a comment


Your Answer

**B** *I*

Sign up or [log in](#)

 Sign up using Google

 Sign up using Facebook

 Sign up using Email and Password

Post Your Answer

By clicking "Post Your Answer", you agree to our [terms of service](#), [privacy policy](#) and [cookie policy](#)

Not the answer you're looking for? Browse other questions tagged [c](#) [operating-system](#) [fork](#) or [ask your own question](#).

Post as a guest

**Name**

**Email**  
Required, but never shown



STACK OVERFLOW

- Questions
- Jobs
- Developer Jobs Directory
- Salary Calculator
- Help
- Mobile
- Disable Responsiveness

PRODUCTS

- Teams
- Talent
- Advertising
- Enterprise

COMPANY

- About
- Press
- Work Here
- Legal
- Privacy Policy
- Contact Us

STACK EXCHANGE NETWORK

- Technology >
- Life / Arts >
- Culture / Recreation >
- Science >
- Other >

[Blog](#) [Facebook](#) [Twitter](#) [LinkedIn](#)

site design / logo © 2019 Stack Exchange Inc; user contributions licensed under cc by-sa 3.0 with attribution required.  
rev 2019.5.9.33647