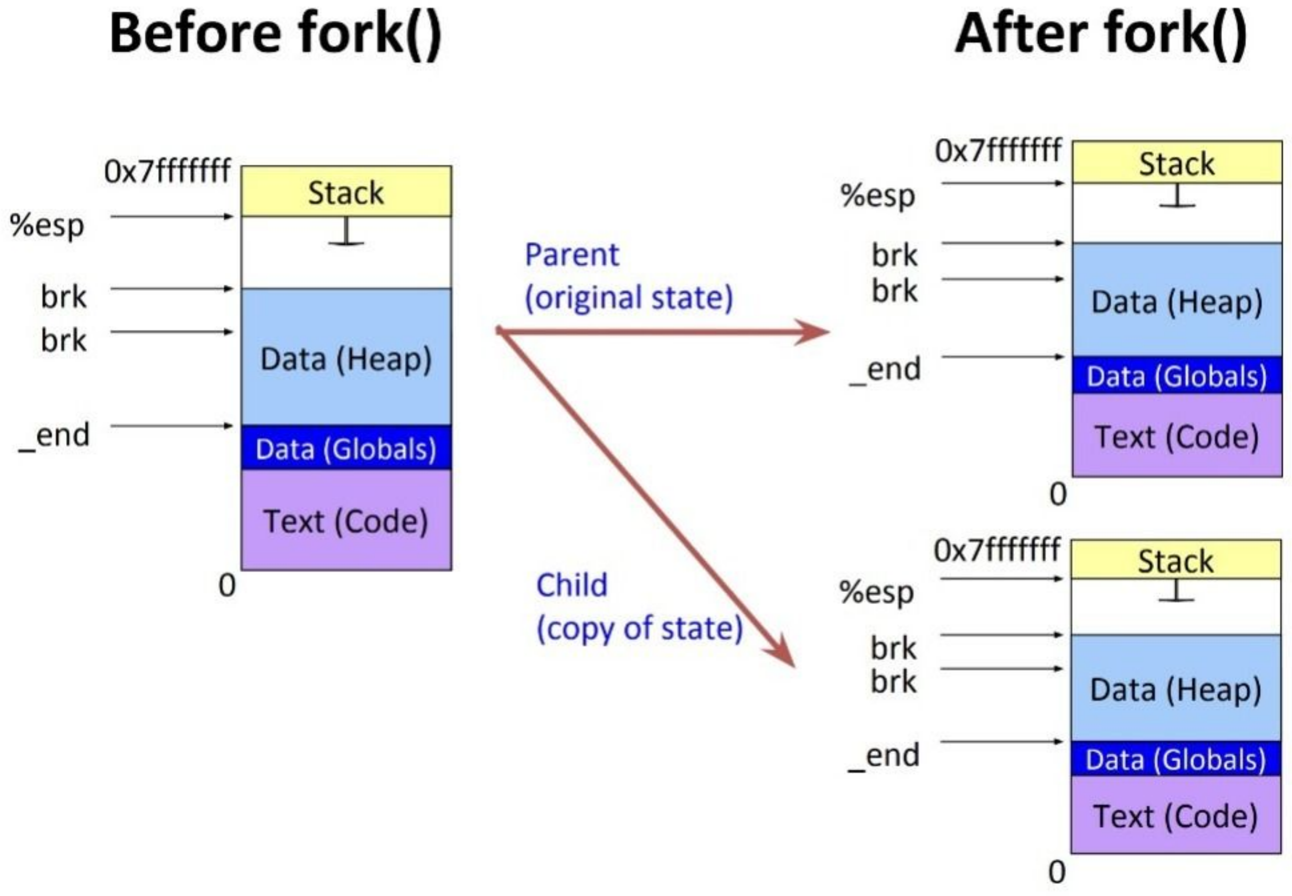
OSN Tutorial 8

Copy-On-Write (CoW)

Back to Basics

fork()



Copy-On-Write (CoW)

What is CoW ?

It is mainly a resource management technique that allows the parent and child process to share the same pages of the memory initially.

Why CoW ?

Inefficient to physically copy memory from parent to child.
The work is often largely wasted: fork() is commonly followed by exec() in the child, which discards the copied memory, usually without using most of it.
If both parent and child use a copied page, and one or both writes it, a copy would be needed though.

– Code (text section) remains identical after fork
– Even portions of data section, heap, and stack may remain identical after fork
– Lazily copy pages only when they are modified

Technique

– The main intention behind the CoW technique is that whenever a parent process creates a child process both parent and child process initially will share the same pages in the memory.

– These shared pages between parent and child process will be marked as *copy-on-write* which means that if the parent or child process will attempt to modify the shared pages then a copy of these pages will be created and the modifications will be done only on the copy of pages by that process and it will not affect other processes.

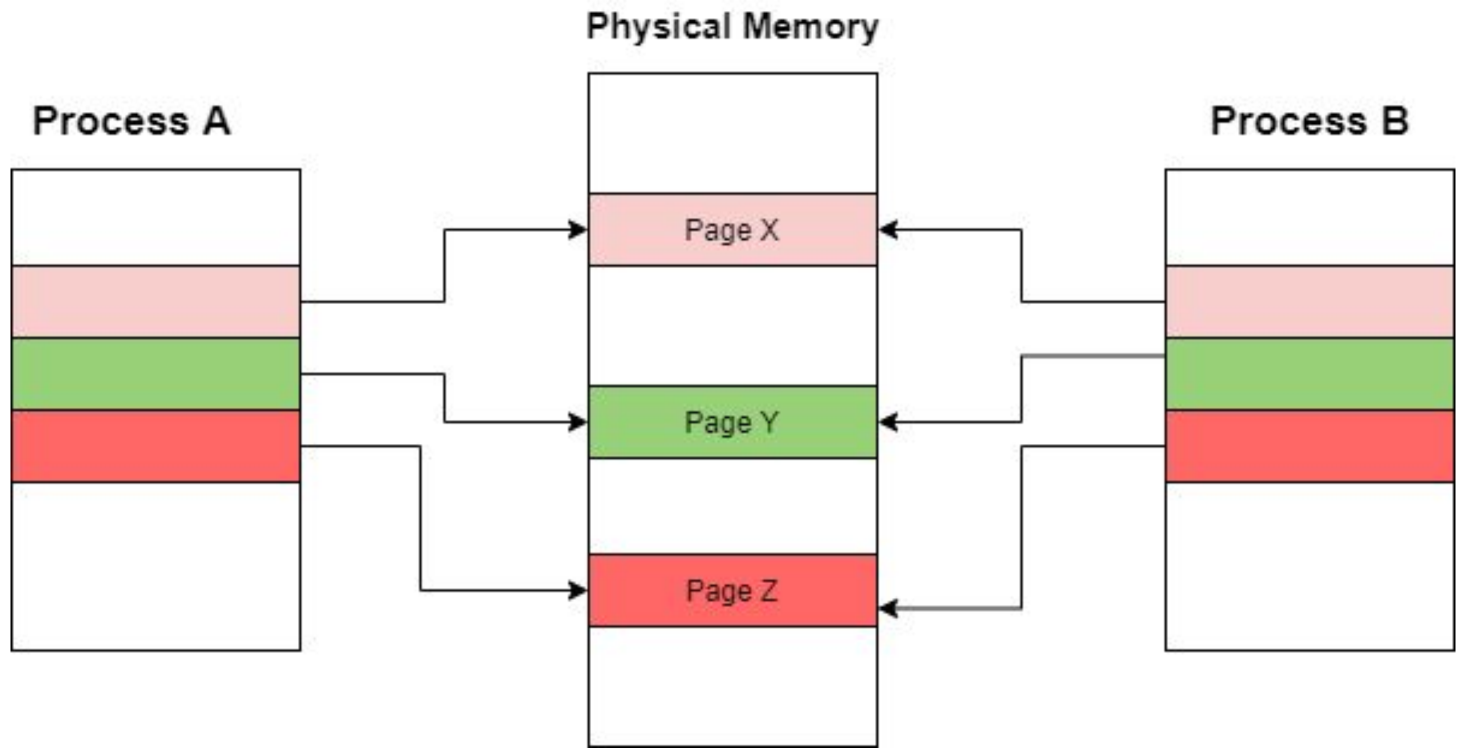
– Initially map same physical page to child virtual memory space (but in read mode).

– Write to child virtual page triggers page protection violation (exception).

– OS handles exception by making physical copy of page and remapping child virtual page to that page.

Explanation via an Example

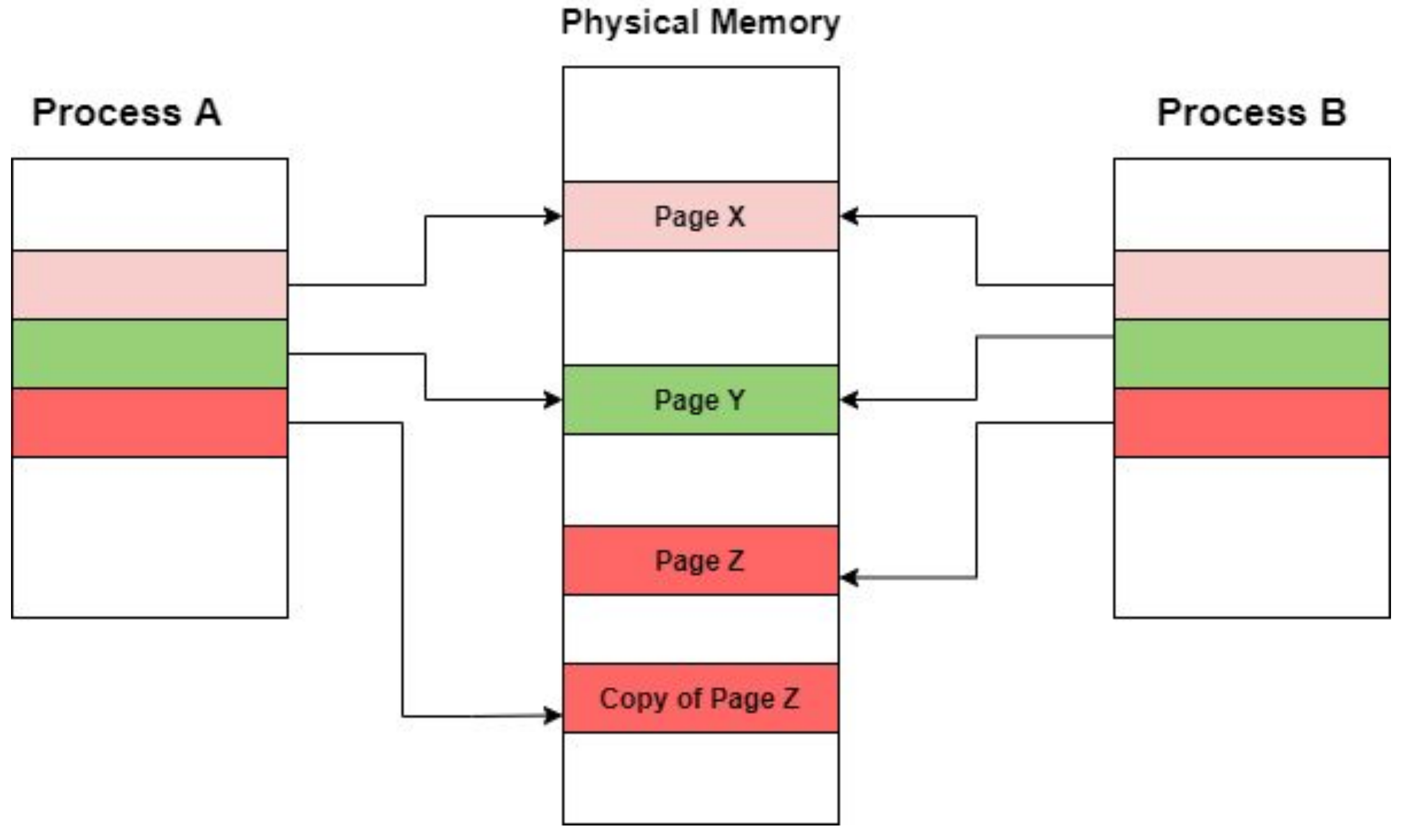
Let us take an example where Process A creates a new process that is Process B, initially both these processes will share the same pages of the memory.



Initial state where process A and B are sharing the same pages.

Explanation via an Example

Now consider the following situation: Process A now wants to modify a page (lets say page-Z) in the memory.
When the *Copy-on-write* technique is used, only those pages that are modified by either process are copied; all the unmodified pages can be easily shared by the parent and child process.



After Page-Z is modified by Process-A.

Zero-fill-on-demand

– Whenever it is determined that a page is going to be duplicated using the copy-on-write technique, then it is important to note the location from where the free pages will be allocated. There is a pool of free pages for such requests; provided by many operating systems.

– And these free pages are allocated typically when the stack/heap for a process must expand or when there are copy-on-write pages to manage.
These pages are typically allocated using the technique that is known as *zero-fill-on-demand*. And the *zero-fill-on-demand* pages are zeroed-out before being allocated and thus erasing the previous content.

– Zero fill on demand simply means to give a region of address space to a process, in which zero-filled pages materialize as they are accessed. It is at the access time that the frames are allocated and filled with zeros before being installed into the address range at the faulting location.

Resources

For a better theoretical understanding:

https://www.codementor.io/@arpitbhayani/copy-on-write-semantics-163hhtuax4

https://www.sobyte.net/post/2022-10/fork-cow/

For practical implementation on the top of xv6 OS:

https://pdos.csail.mit.edu/6.828/2019/labs/cow.html