

Injae Kang  
([abcinje@snu.ac.kr](mailto:abcinje@snu.ac.kr))

Systems Software &  
Architecture Lab.

Seoul National University

2023.11.09.

# Project #4: mmap() with Huge Pages



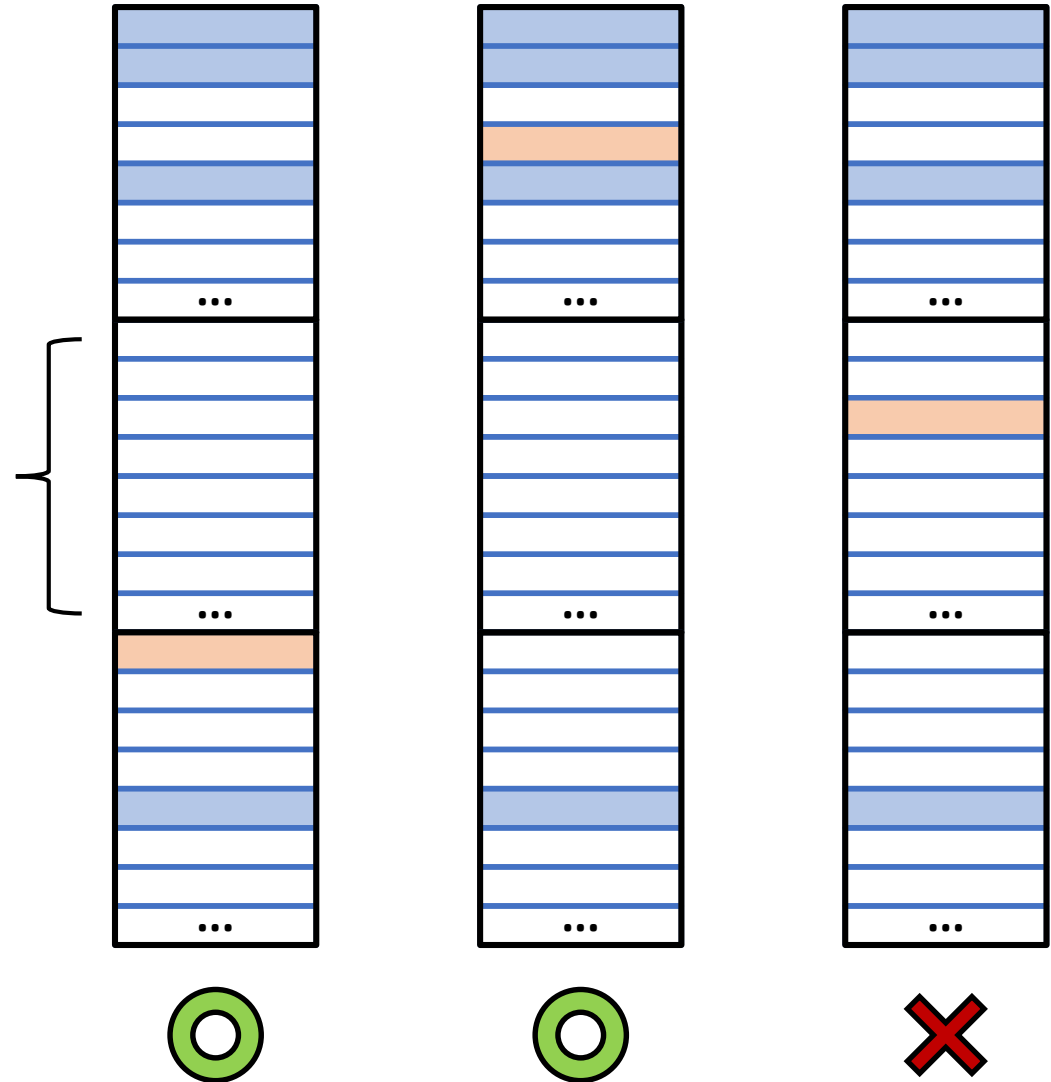
# Pages and Huge Pages

- In this project, you need to implement huge page allocation/deallocation
- Base page: 4KiB
- Huge page: 2MiB
- The size of 512 base pages is same as the size of a single huge page
  
- The new allocator is to support four functions as follows:
  - `void *kalloc();`
  - `void kfree(void *pa);`
  - `void *kalloc_huge();`
  - `void kfree_huge(void *pa);`

# Pages and Huge Pages

- **Restriction**
  - You should maximize the number of allocatable 2MiB frames

Unallocated 2MiB



# Memory Mapping

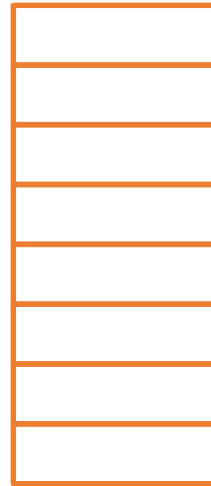
- `mmap()` system call creates a new mapping in the virtual address space
- You need to implement following system calls:
  - `void *mmap(void *addr, int length, int prot, int flags);`
  - `int munmap(void *addr);`
- We only consider anonymous mapping
- You don't have to care about file-backed mapping

# Shared Mapping

- `mmap()`



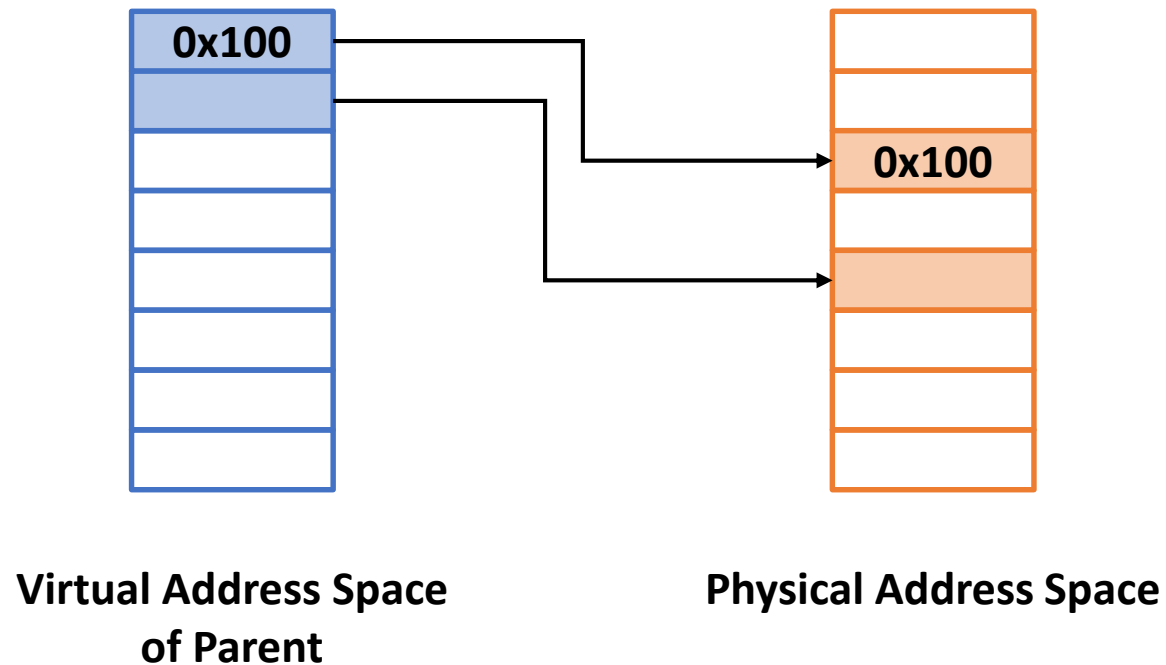
**Virtual Address Space  
of Parent**



**Physical Address Space**

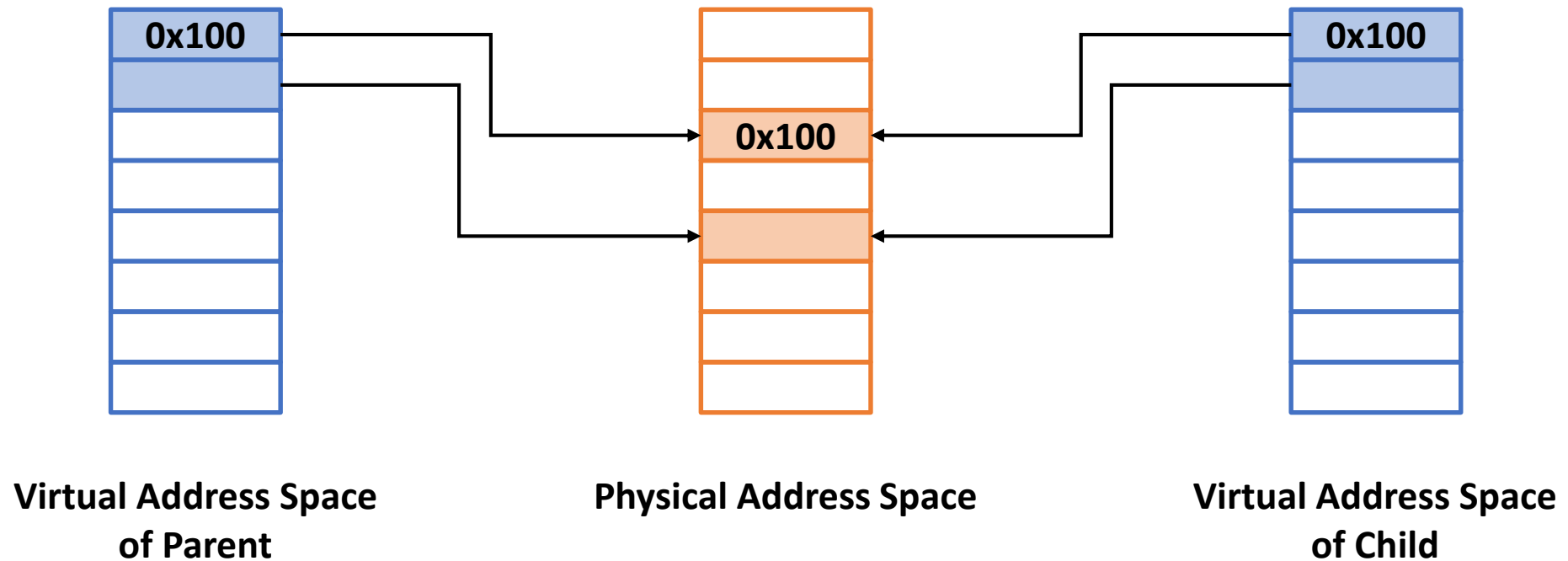
# Shared Mapping

- Parent writes 0x100
  - Assume that each `mmap()`-ed page is writable



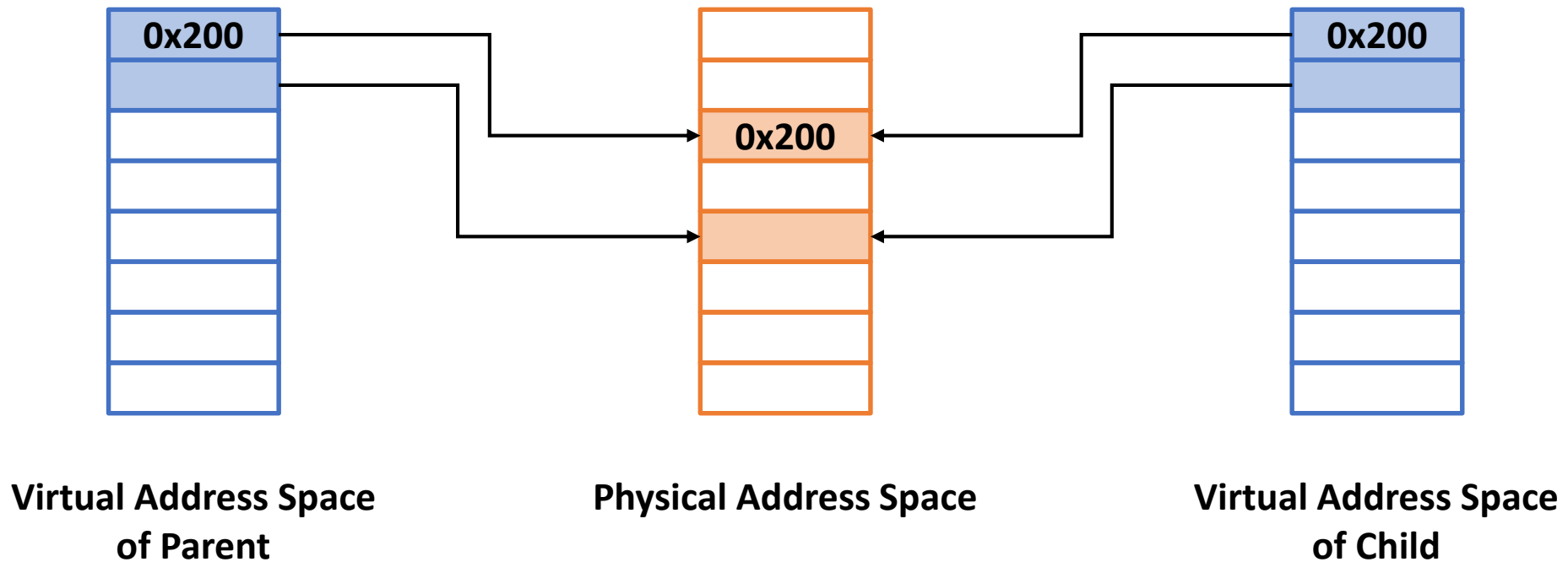
# Shared Mapping

- `fork()`



# Shared Mapping

- Child writes 0x200
  - Assume that each `mmap()`-ed page is writable



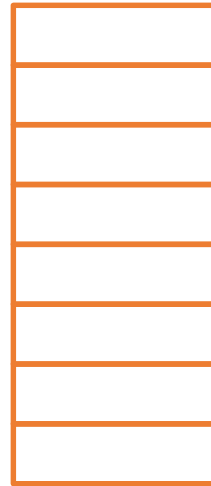


# Private Mapping

- `mmap()`



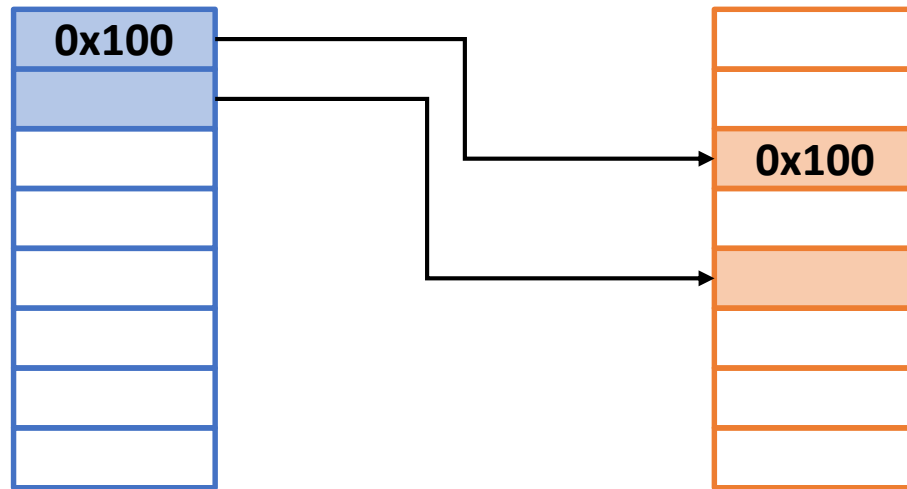
**Virtual Address Space  
of Parent**



**Physical Address Space**

# Private Mapping

- Parent writes 0x100
  - Assume that each `mmap()`-ed page is writable

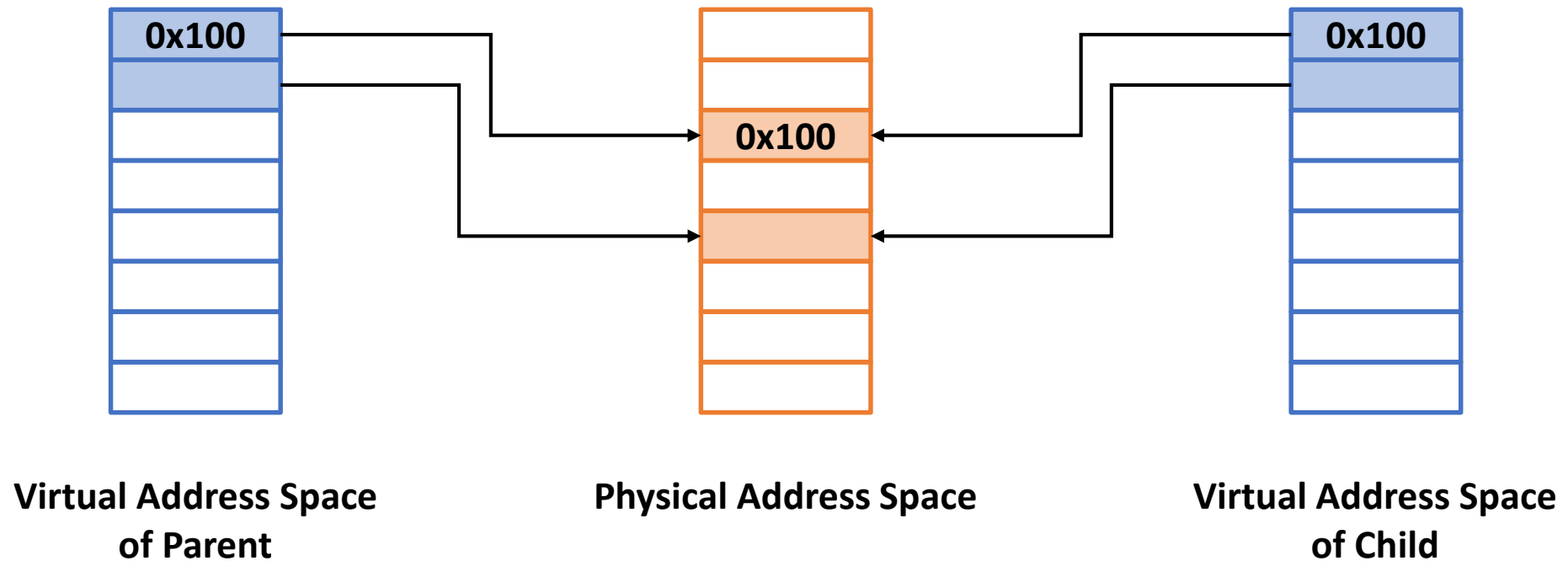


Virtual Address Space  
of Parent

Physical Address Space

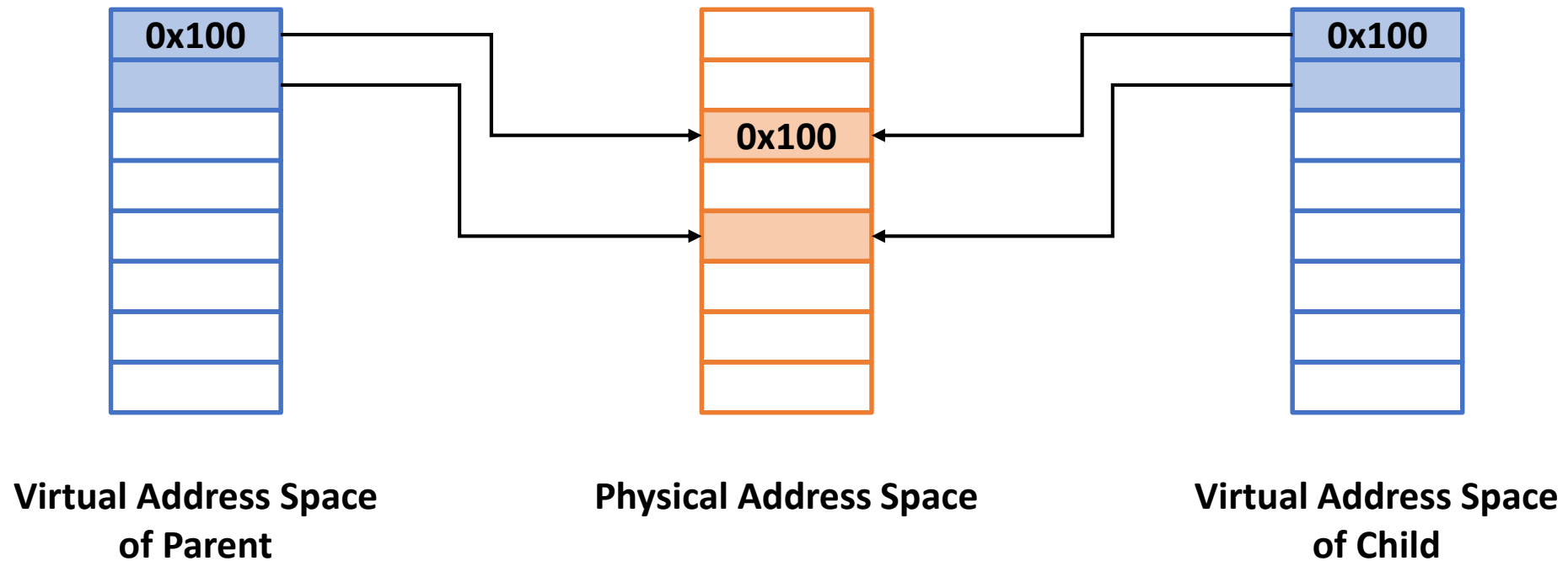
# Private Mapping

- `fork()`



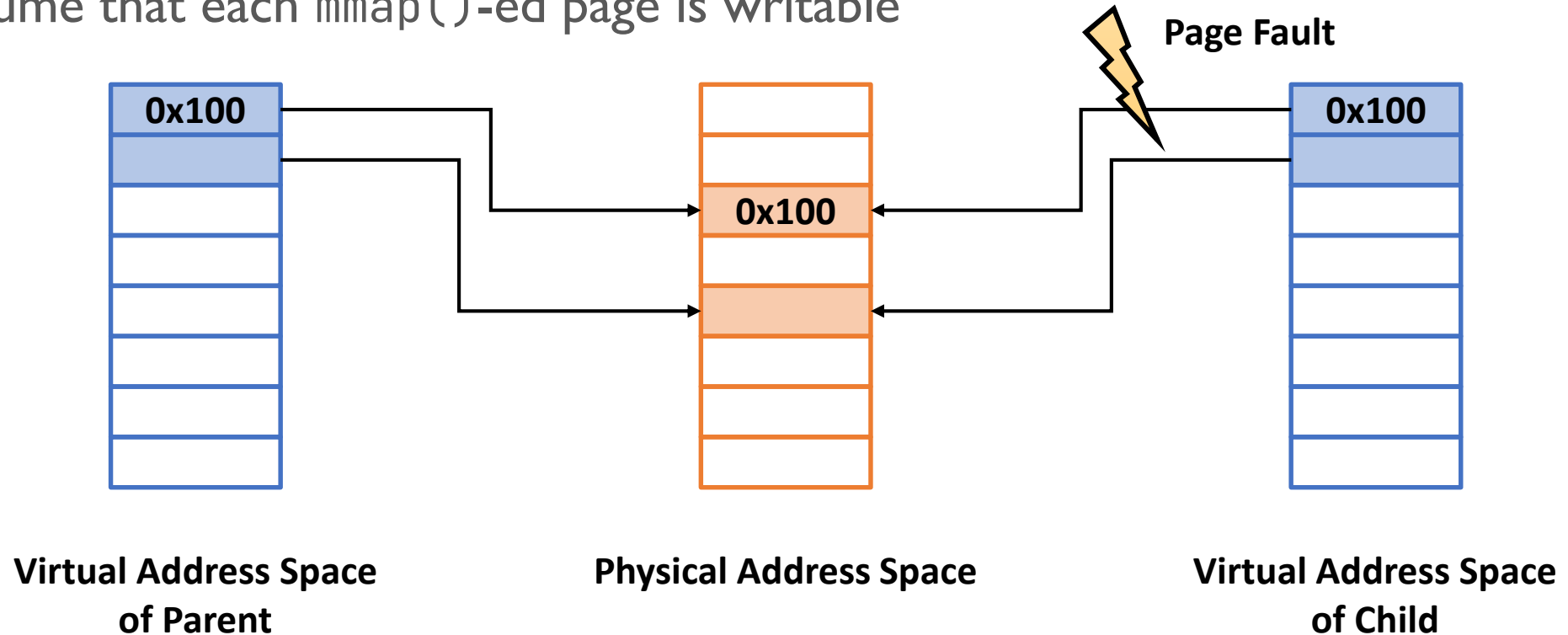
# Private Mapping

- Child reads the content
  - Result: 0x100



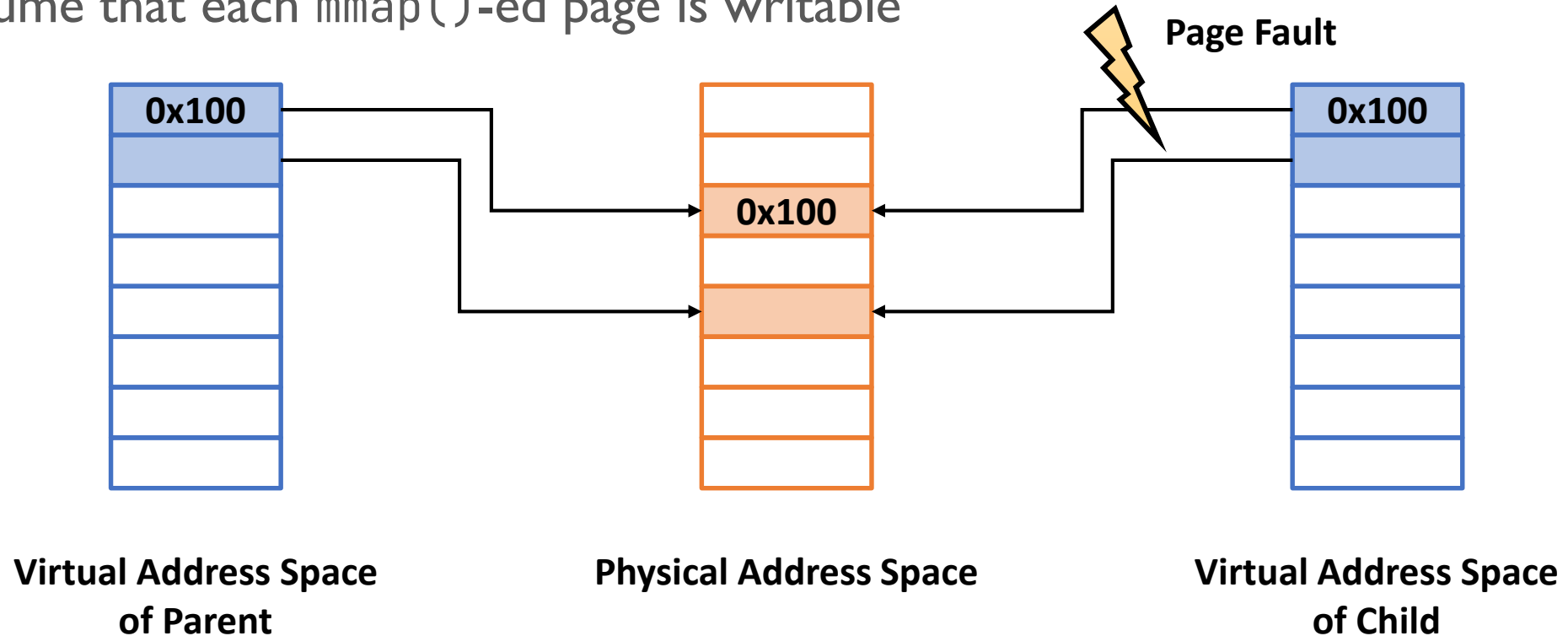
# Private Mapping

- Child writes 0x200
  - Assume that each `mmap()`-ed page is writable



# Private Mapping

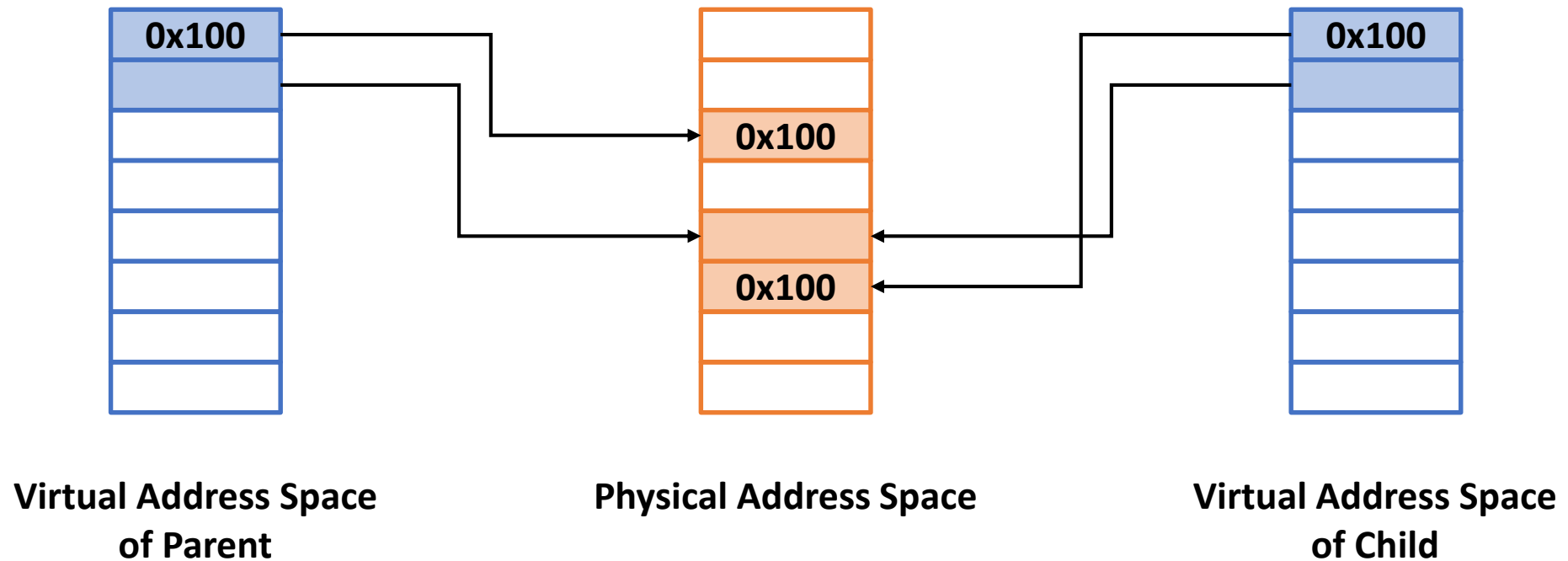
- Child writes 0x200
  - Assume that each `mmap()`-ed page is writable



**The page fault is due to permission violation, because the page table entry indicates that the page is read-only, although the page appears to be writable to the user application.**

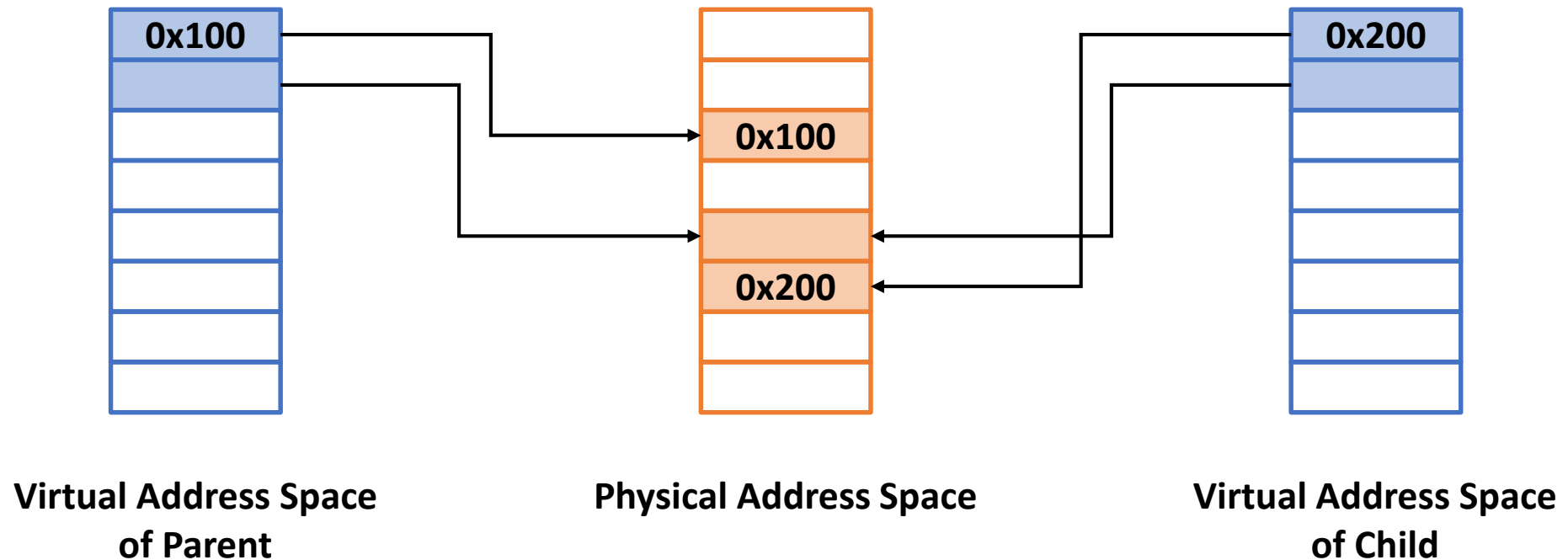
# Private Mapping

- Page fault handling
  - Copy-on-write



# Private Mapping

- Child writes 0x200 (Retry)
  - Assume that each `mmap()`-ed page is writable





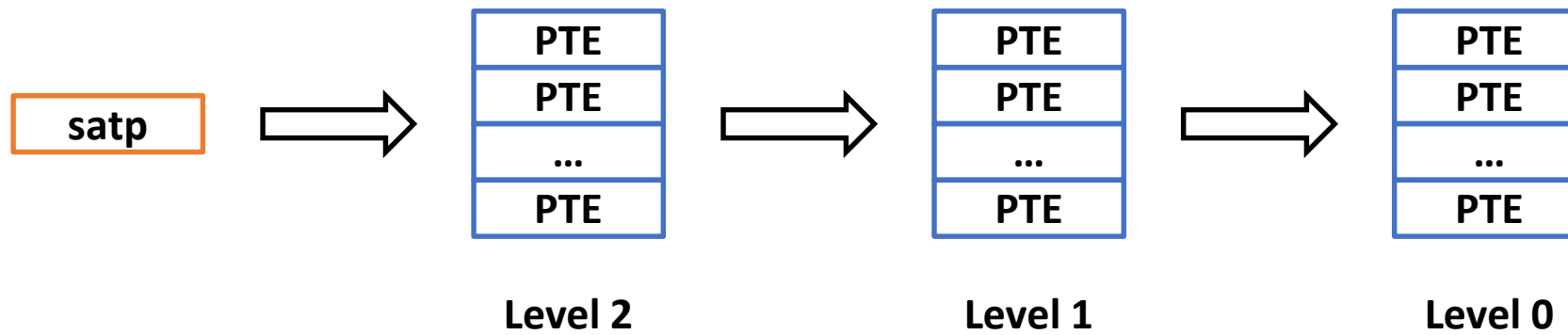
# Cleanup

- All of the `mmap()`-ed regions should be unmapped on process termination
- Memory unmapping does not always require memory deallocation
  - When should we deallocate physical pages in case of shared mapping?
  - When should we deallocate physical pages in case of private mapping?

# Sv39

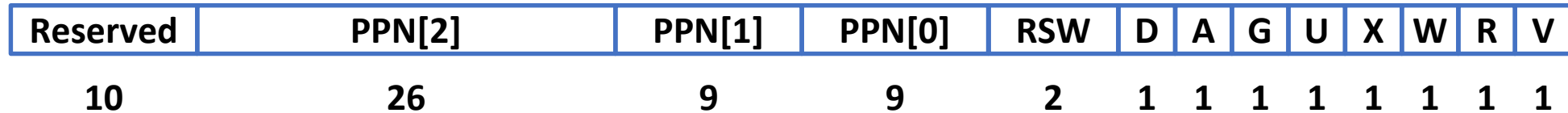
- xv6 uses 39-bit address system called Sv39
- 3-level page table
- It supports 1GiB and 2MiB huge pages
  - If a level 2 entry is a leaf, it represents an 1GiB-sized huge page
  - If a level 1 entry is a leaf, it represents a 2MiB-sized huge page
  - If a level 0 entry is a leaf, it represents a 4KiB base page

# Sv39 Page Table



Reserved	PPN[2]	PPN[1]	PPN[0]	RSW	D	A	G	U	X	W	R	V
10	26	9	9	2	1	1	1	1	1	1	1	1

# Sv39 Page Table Entry



## ■ Page table entry bits

- D: Dirty bit
  - A: Access bit
  - G: Global bit
  - U: User bit
  - X: Execute bit
  - W: Write bit
  - R: Read bit
  - V: Valid bit
- 
- If X, W, and R are all 0, the PTE is a pointer to next level

# Project #4

- Tips

- Read Chap. 3 and 4 of the [xv6 book](#) to understand the virtual memory subsystem and page-fault exceptions in xv6

# Project #4

## ■ Assumptions

- The range of the target virtual address in `mmap()` is from `PHYSTOP` to `MAXVA-0x100000000`
  - The maximum size in `mmap()` is limited to 64MiB
  - Each process can have up to 4 memory-mapped regions
  - The system can support up to 64 distinct memory-mapped regions in total
- 
- You may assume that no test scenarios break the assumptions

# Project #4

## ■ Restrictions

- On `exit()` or `exec()`, all the memory-mapped regions should be unmapped
- Your implementation should work on multi-core systems
- Do not add any other system calls
- You only need to modify those files in the `./kernel` directory except for the `./kernel/ktest.c` file

# Project #4

## ■ Skeleton Code

- You should work on the `pa4` branch of the `xv6-riscv-snu` repository as follows:

```
$ git clone https://github.com/snu-csl/xv6-riscv-snu
$ git checkout pa4
```

- The `pa4` branch has three user-level programs (`mmaptest1` ~ `mmaptest3`) which can be built from `./user/mmaptest1.c` ~ `./user/mmaptest3.c`, respectively



# Project #4

## ■ Due

- 11:59 PM, November 25 (Saturday)

## ■ Submission

- Run `make submit` command to generate a tarball named `xv6-pa4- $\{STUDENTID\}$ .tar.gz` in the `xv6-riscv-snu` directory
- Upload the compressed file to the submission server
- The total number of submissions for this project will be limited to 30
- Only the version marked **FINAL** will be considered for the project score
- In this project, you need to submit a design report

# Using GDB with QEMU

# GDB with QEMU

- In the `xv6-riscv-snu` directory,
- Run `make qemu-gdb` to run QEMU
- In another shell, run `riscv64-unknown-elf-gdb ./kernel/kernel`

```
csl@sys.snu.ac.kr x + v - □ ×
csl@sys ~/injae/xv6-riscv-snu % make qemu-gdb
*** Now run 'gdb' in another window.
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 4 -nographic -global virtio-mmio.force-legacy=false -drive file=fs.img,if=none,format=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0 -S -gdb tcp::26000
█
```

```
csl@sys.snu.ac.kr x + v - □ ×
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "--host=x86_64-pc-linux-gnu --target=riscv64-unknown-elf".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from kernel/kernel...
warning: File "/home/csl/injae/xv6-riscv-snu/.gdbinit" auto-loading has been declined
by your `auto-load safe-path' set to "$debugdir:$datadir/auto-load".
To enable execution of this file add
    add-auto-load-safe-path /home/csl/injae/xv6-riscv-snu/.gdbinit
line to your configuration file "/home/csl/.gdbinit".
To completely disable this security protection add
    set auto-load safe-path /
line to your configuration file "/home/csl/.gdbinit".
For more information about this security protection see the
"Auto-loading safe path" section in the GDB manual.  E.g., run from the shell:
--Type <RET> for more, q to quit, c to continue without paging--
info "(gdb)Auto-loading safe path"
(gdb) █
```

# GDB with QEMU

- In GDB, enter target remote :<port>
- You can find TCP port in the QEMU log

```
csl@sys.snu.ac.kr x + v
csl@sys ~/injae/xv6-riscv-snu % make qemu-gdb
*** Now run 'gdb' in another window.
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 4 -nog
raphic -global virtio-mmio.force-legacy=false -drive file=fs.img,if=none,format=raw,id
=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0 -S -gdb tcp::26000
```

```
csl@sys.snu.ac.kr x + v
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from kernel/kernel...
warning: File "/home/csl/injae/xv6-riscv-snu/.gdbinit" auto-loading has been declined
by your `auto-load safe-path' set to "$debugdir:$datadir/auto-load".
To enable execution of this file add
  add-auto-load-safe-path /home/csl/injae/xv6-riscv-snu/.gdbinit
line to your configuration file "/home/csl/.gdbinit".
To completely disable this security protection add
  set auto-load safe-path /
line to your configuration file "/home/csl/.gdbinit".
For more information about this security protection see the
"Auto-loading safe path" section in the GDB manual.  E.g., run from the shell:
--Type <RET> for more, q to quit, c to continue without paging--
info "(gdb)Auto-loading safe path"
(gdb) target remote :26000
Remote debugging using :26000
0x00000000000001000 in ?? ()
(gdb)
```

# GDB with QEMU

- The xv6 virtual machine has stopped at 0x1000 (the very beginning of the text section)
- To continue, enter c in GDB

```
csl@sys.snu.ac.kr x + v
csl@sys ~/injae/xv6-riscv-snu % make qemu-gdb
*** Now run 'gdb' in another window.
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 4 -nographic -global virtio-mmio.force-legacy=false -drive file=fs.img,if=none,format=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0 -S -gdb tcp::26000

xv6 kernel is booting

hart 3 starting
hart 2 starting
hart 1 starting
init: starting sh
$ █
```

(Running)

```
csl@sys.snu.ac.kr x + v
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
  <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from kernel/kernel...
warning: File "/home/csl/injae/xv6-riscv-snu/.gdbinit" auto-loading has been declined
by your `auto-load safe-path' set to "$debugdir:$datadir/auto-load".
To enable execution of this file add
  add-auto-load-safe-path /home/csl/injae/xv6-riscv-snu/.gdbinit
line to your configuration file "/home/csl/.gdbinit".
To completely disable this security protection add
  set auto-load safe-path /
line to your configuration file "/home/csl/.gdbinit".
For more information about this security protection see the
"Auto-loading safe path" section in the GDB manual.  E.g., run from the shell:
--Type <RET> for more, q to quit, c to continue without paging--
info "(gdb)Auto-loading safe path"
(gdb) target remote :26000
Remote debugging using :26000
0x0000000000000100 in ?? ()
(gdb) c
Continuing.
█
```

# GDB with QEMU

- To stop again, enter `Ctrl-C` in GDB
- Then the xv6 virtual machine stops immediately

```
csl@sys.snu.ac.kr x + v
csl@sys ~/injae/xv6-riscv-snu % make qemu-gdb
*** Now run 'gdb' in another window.
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 4 -nographic -global virtio-mmio.force-legacy=false -drive file=fs.img,if=none,format=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0 -S -gdb tcp::26000

xv6 kernel is booting

hart 3 starting
hart 2 starting
hart 1 starting
init: starting sh
$ █

(Stopped)
```

```
csl@sys.snu.ac.kr x + v
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from kernel/kernel...
warning: File "/home/csl/injae/xv6-riscv-snu/.gdbinit" auto-loading has been declined by your `auto-load safe-path' set to "$debugdir:$datadir/auto-load".
To enable execution of this file add
    add-auto-load-safe-path /home/csl/injae/xv6-riscv-snu/.gdbinit
line to your configuration file "/home/csl/.gdbinit".
To completely disable this security protection add
    set auto-load safe-path /
line to your configuration file "/home/csl/.gdbinit".
For more information about this security protection see the
"Auto-loading safe path" section in the GDB manual.  E.g., run from the shell:
--Type <RET> for more, q to quit, c to continue without paging--
info "(gdb)Auto-loading safe path"
(gdb) target remote :26000
Remote debugging using :26000
0x0000000000001000 in ?? ()
(gdb) c
Continuing.
^C
Thread 1 received signal SIGINT, Interrupt.
mycpu () at kernel/proc.c:79
79     {
(gdb) █
```

# GDB with QEMU

- Let's set a breakpoint at `exec()`
- Enter `b exec` in GDB

```
csl@sys.snu.ac.kr x + v
csl@sys ~/injae/xv6-riscv-snu % make qemu-gdb
*** Now run 'gdb' in another window.
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 4 -nographic -global virtio-mmio.force-legacy=false -drive file=fs.img,if=none,format=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0 -S -gdb tcp::26000

xv6 kernel is booting

hart 3 starting
hart 2 starting
hart 1 starting
init: starting sh
$ █
```

(Stopped)

```
csl@sys.snu.ac.kr x + v
Reading symbols from kernel/kernel...
warning: File "/home/csl/injae/xv6-riscv-snu/.gdbinit" auto-loading has been declined by your `auto-load safe-path' set to "$debugdir:$datadir/auto-load".
To enable execution of this file add
  add-auto-load-safe-path /home/csl/injae/xv6-riscv-snu/.gdbinit
line to your configuration file "/home/csl/.gdbinit".
To completely disable this security protection add
  set auto-load safe-path /
line to your configuration file "/home/csl/.gdbinit".
For more information about this security protection see the
"Auto-loading safe path" section in the GDB manual.  E.g., run from the shell:
--Type <RET> for more, q to quit, c to continue without paging--
info "(gdb)Auto-loading safe path"
(gdb) target remote :26000
Remote debugging using :26000
0x0000000000001000 in ?? ()
(gdb) c
Continuing.
^C
Thread 1 received signal SIGINT, Interrupt.
mycpu () at kernel/proc.c:79
79   {
(gdb) b exec
Breakpoint 1 at 0x80004ec0: file kernel/exec.c, line 24.
(gdb) █
```

# GDB with QEMU

- Enter c in GDB to resume the xv6 machine

```
csl@sys.snu.ac.kr x + v
csl@sys ~/injae/xv6-riscv-snu % make qemu-gdb
*** Now run 'gdb' in another window.
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 4 -nographic -global virtio-mmio.force-legacy=false -drive file=fs.img,if=none,format=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0 -S -gdb tcp::26000

xv6 kernel is booting

hart 3 starting
hart 2 starting
hart 1 starting
init: starting sh
$ █
```

(Running)

```
csl@sys.snu.ac.kr x + v
by your `auto-load safe-path' set to "$debugdir:$datadir/auto-load".
To enable execution of this file add
    add-auto-load-safe-path /home/csl/injae/xv6-riscv-snu/.gdbinit
line to your configuration file "/home/csl/.gdbinit".
To completely disable this security protection add
    set auto-load safe-path /
line to your configuration file "/home/csl/.gdbinit".
For more information about this security protection see the
"Auto-loading safe path" section in the GDB manual.  E.g., run from the shell:
--Type <RET> for more, q to quit, c to continue without paging--
info "(gdb)Auto-loading safe path"
(gdb) target remote :26000
Remote debugging using :26000
0x0000000000001000 in ?? ()
(gdb) c
Continuing.
^C
Thread 1 received signal SIGINT, Interrupt.
mycpu () at kernel/proc.c:79
79     {
(gdb) b exec
Breakpoint 1 at 0x80004ec0: file kernel/exec.c, line 24.
(gdb) c
Continuing.
█
```



# GDB with QEMU

- Run `ls` command in the `xv6` machine
- Then the `xv6` machine hits the breakpoint and stops right before starting `exec()` function

```
csl@sys.snu.ac.kr x + v
csl@sys ~/injae/xv6-riscv-snu % make qemu-gdb
*** Now run 'gdb' in another window.
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 4 -nographic -global virtio-mmio.force-legacy=false -drive file=fs.img,if=none,format=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0 -S -gdb tcp::26000

xv6 kernel is booting

hart 3 starting
hart 2 starting
hart 1 starting
init: starting sh
$ ls
█
```

(Stopped)

```
csl@sys.snu.ac.kr x + v
set auto-load safe-path /
line to your configuration file "/home/csl/.gdbinit".
For more information about this security protection see the
"Auto-loading safe path" section in the GDB manual. E.g., run from the shell:
--Type <RET> for more, q to quit, c to continue without paging--
info "(gdb)Auto-loading safe path"
(gdb) target remote :26000
Remote debugging using :26000
0x0000000000001000 in ?? ()
(gdb) c
Continuing.
^C
Thread 1 received signal SIGINT, Interrupt.
mycpu () at kernel/proc.c:79
79      {
(gdb) b exec
Breakpoint 1 at 0x80004ec0: file kernel/exec.c, line 24.
(gdb) c
Continuing.
[Switching to Thread 1.2]

Thread 2 hit Breakpoint 1, exec (path=path@entry=0x3fffff9f00 "ls",
      argv=argv@entry=0x3fffff9e00) at kernel/exec.c:24
24      {
(gdb) █
```

# More about GDB

- To learn GDB in detail, search for GDB on Google
- There are many useful videos about GDB in YouTube
- [\[JT\]의 리눅스탐험\] GDB 활용하기](#)

Thank you!