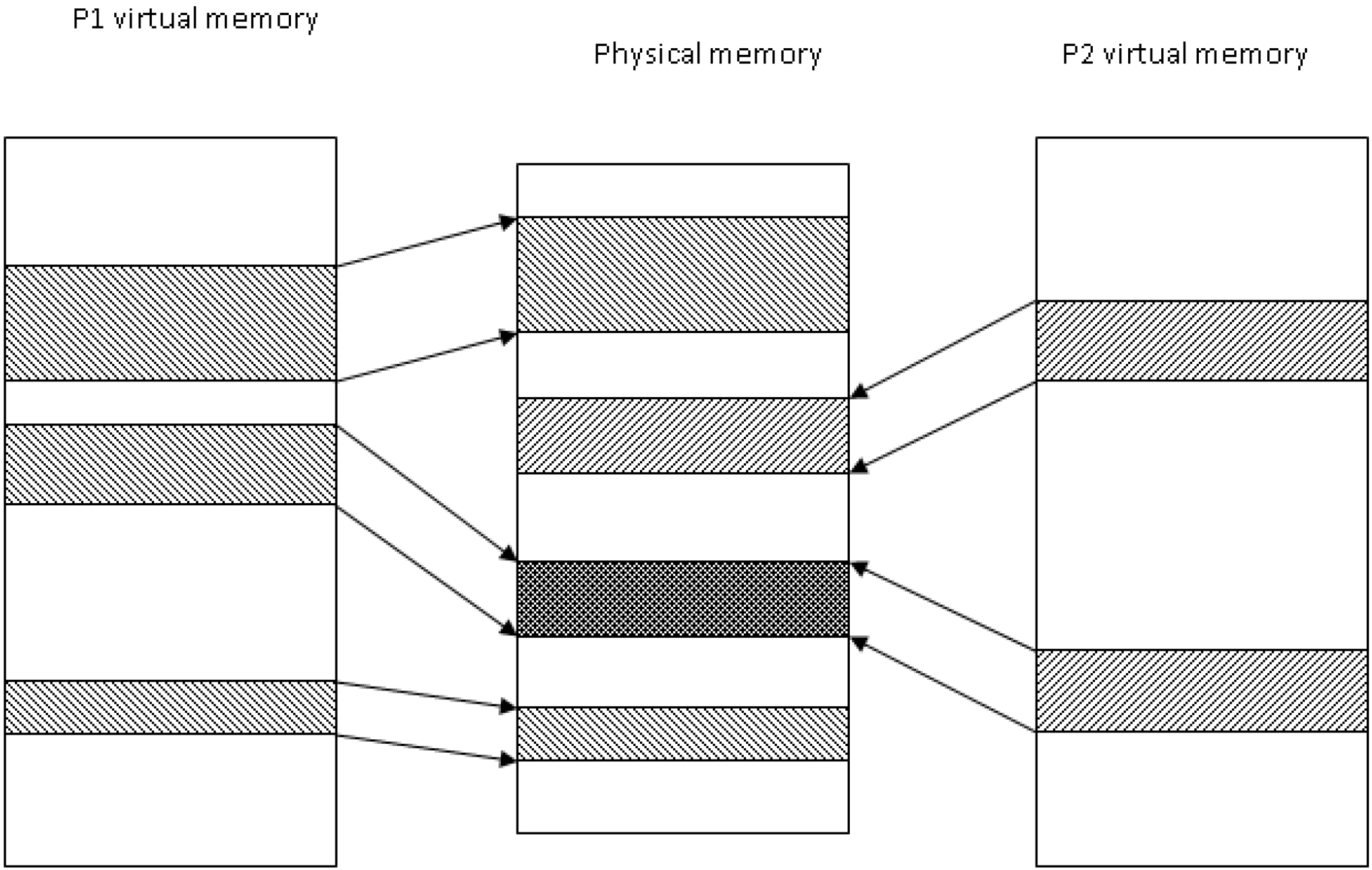
Xv6 Virtual Memory and Sharing

Instructor: Yaoqing Liu

TA: Garegin Grigoryan

Shared Memory



Download New Xv6

- ssh to odin and change directory to Lab7

$ssh YourName@odin.cslabs.clarkson.edu

$cd ~/cs444-s18/Lab7

- Download xv6.tar.gz file to your work directory Lab7

$wget http://people.clarkson.edu/~liu/CS444/Spring18/xv6.tar.gz

- Unzip it

$tar –xzvf xv6.tar.gz

What You Do

- Implement a system call called shm_yourname
- Allocates a shared memory segment

SYNOPSIS

Void *shm_yourname(int key, int num_pages)

- if processes call **shm_yourname** with the same **key** for the first argument, then they will share the specified number of physical pages
- Using different keys in different calls to **shm_yourname()** corresponds to different physical pages

More Description

- **shm_yourname** returns the virtual address of the shared pages to the caller, so the process can read/write them
- **shm_yourname** should map the shared phyiscal pages to the next available virtual pages, starting at the high end of that process' address space
- 0x80000000 and above is reserved for the kernel

Example

- When a process calls **shm_yourname(0, 1):**
  - The OS should map 1 physical page into the virtual address space of the caller, starting at the very high end of the address space
- If another process then calls **shm_yourname(0, ANY_VALUE):**
  - this process should also get that same 1 page mapped into its virtual address space (possibly at a different virtual address)
  - Second argument is ignored if the key has been already existing
- The two processes can then each read and write to this page and thus communicate
- A third or fourth process calls **shm_yourname(0, ANY_VALUE)**
  - Same page mapped into their address spaces as well

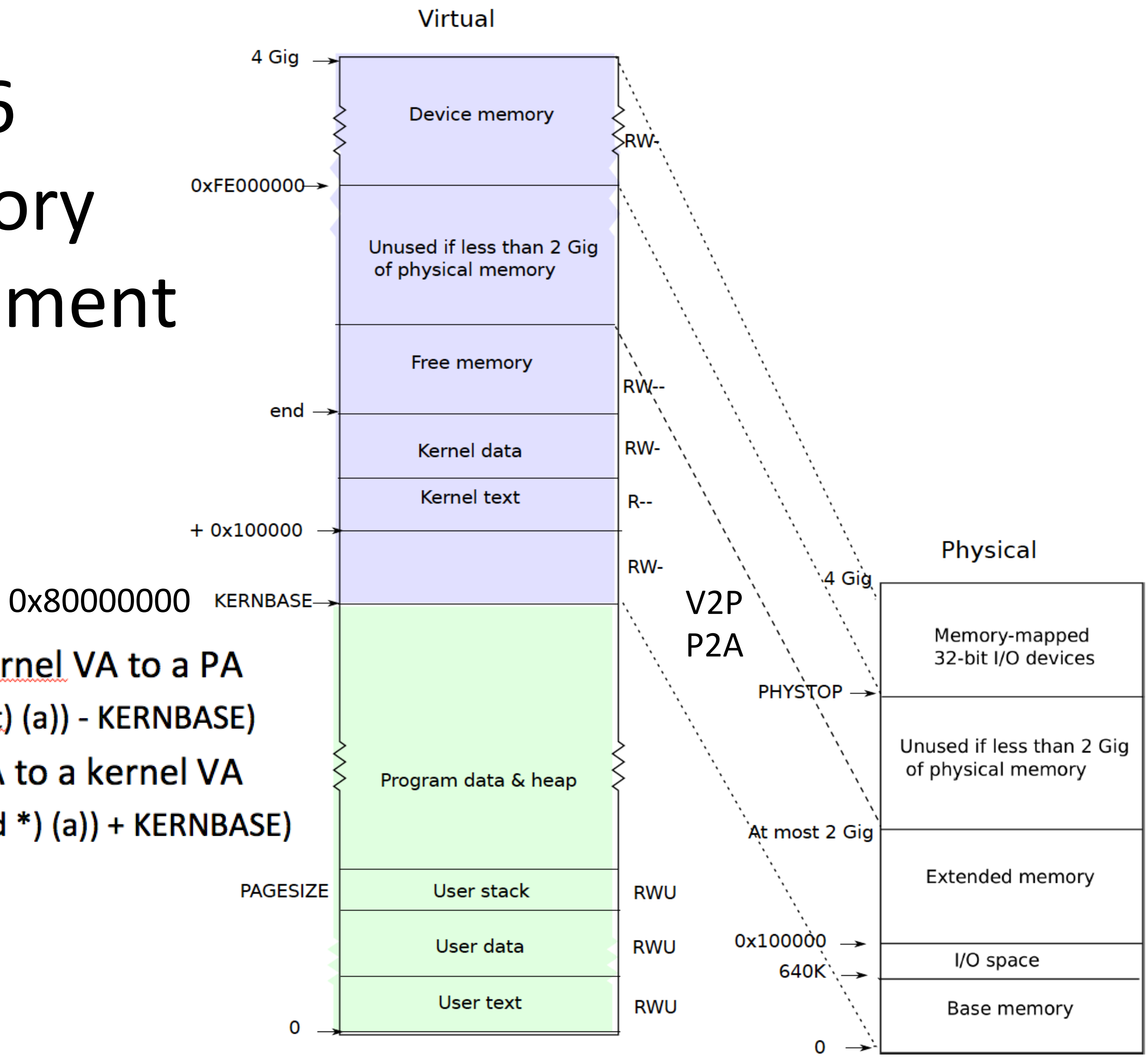
Another example

- Another key is used, **shm_yourname(1, 3)**
  - This corresponds to a new shared region
- if any other process calls **shm_yourname(1, 3)**
  - OS will map 3 (new) physical pages into the address space of the calling process
  - Associate these 3 pages with key value 1
  - Subsequent calls that use **key=1** will map these three pages into the calling process' address space
- To reduce the complexity, start num=1

Required Features

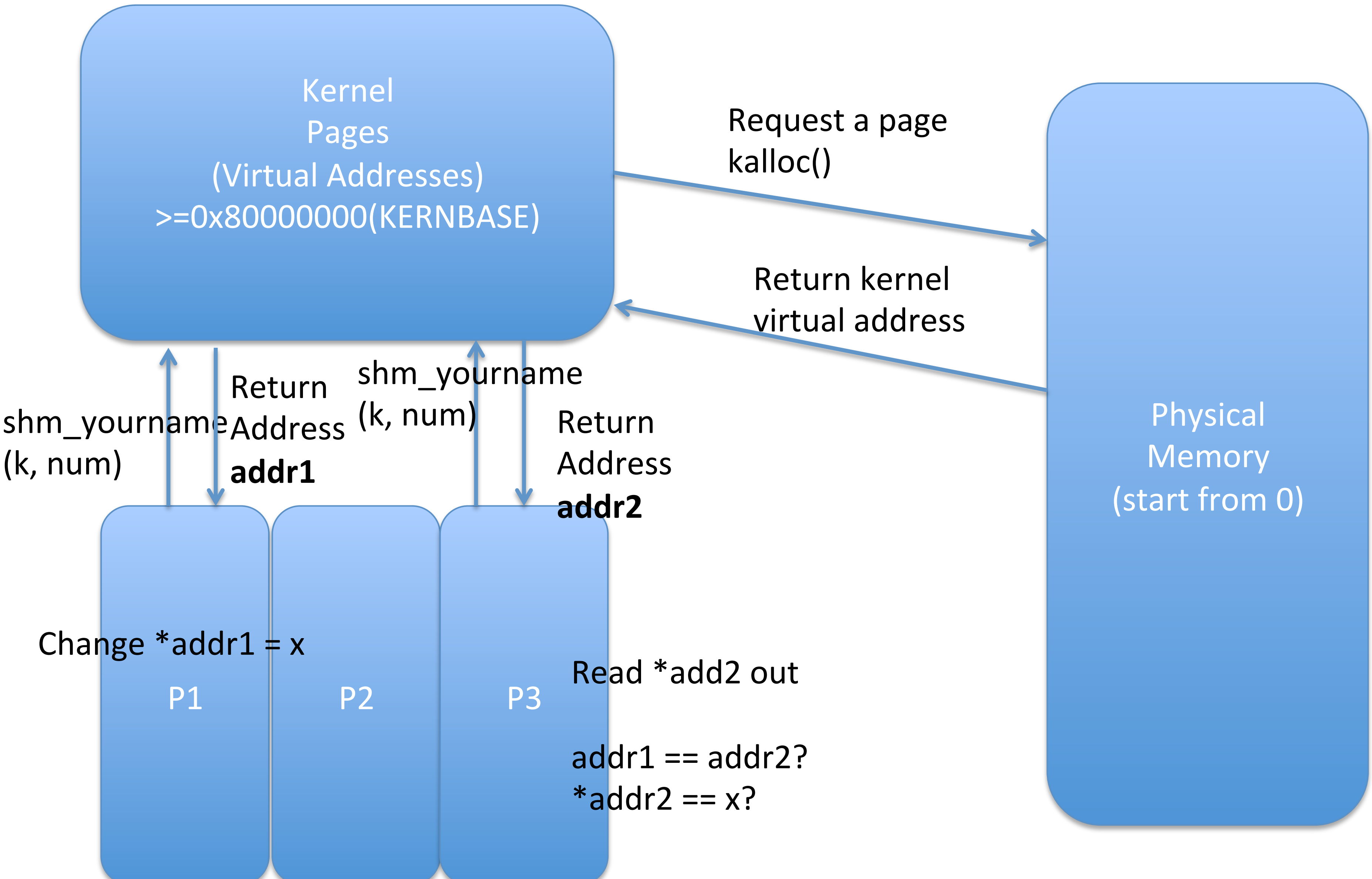
- Same process can call shm_yourname more than once
- Keys are globally visible, not for a particular process
- When a fork is called, every shared region has to be accessible to the new process **without** calling shm_yourname() again
- When a process exits, make sure that shared pages stay

Xv6 memory Management



- Translate a kernel VA to a PA
  - V2P(a) (((uint) (a)) - KERNBASE)
- Translate a PA to a kernel VA
  - P2V(a) (((void *) (a)) + KERNBASE)

One way to do page sharing?



One Way for Sharing Pages

Initialization

- Define a global variable SharedMem[PAGES]
- Allocate pages (*kalloc*) using a function ShareMemInit() in vm.c
- Assign the kernel virtual addresses to SharedMem[PAGES] during booting
- Call ShareMemInit() during booting in ***main.c***

Add System Call shm_yourname(key, num)

- Change these files: defs.h, syscall.c, syscall.h, sysproc.c, user.h
- Implement it in vm.c

Implementation

- Start from top of virtual address in user space (0x8000000)
- The first virtual address should be the starting point of the page: 0x80000000-PAGESIZE(4K, 0x1000) = 0x7ffff000
- Walk through the page table and map virtual addresses to their physical addresses (walkpgdir and mappages functions)
- Physical addresses can be obtained from SharedMem[PAGES] (V2P function)
- **You can use your own way to implement it**

Testing

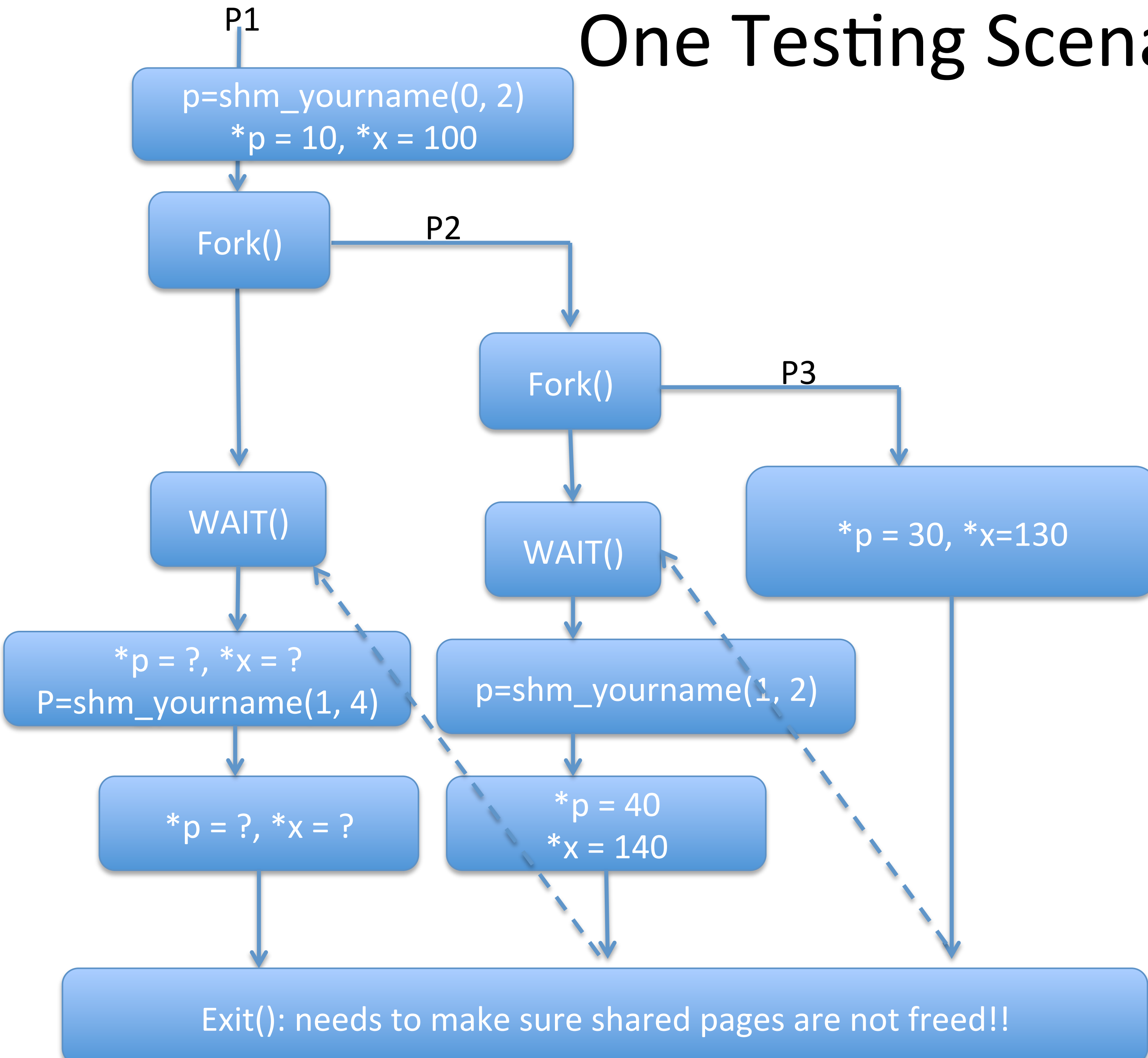
Test Page Sharing

- Write a user program test_shm_yourname.c in directory userspace to call the system call
- Fork() is needed to test the shared memory between processes
- One process assigns a value to a location in the shared memory and another process read it out, the two values should be the same

Test Key Identification

- Process 1 uses key 0
- Process 2 uses key 1 and writes to the shared page X
- process 3 uses key 1 and read the shared page value, which should be X as well

One Testing Scenario



```
int p = shmgetat(0, 2);
int* test;
test = (int *)p;
*test = 10;
int t = 100;
int* x = &t;
if(fork()==0){
      if(fork() == 0){
              //Current limitation: key needs to be used continuously, 0 first, then 1, 2...
              //p = shmgetat(0, 3);
              test = (int *)p;
              *test = 30;
              *x = 130;
              printf(1, "Grand Child: %x, %d, %x, %d\n", test, *test, x, *x);
      }
      else{
              wait();
              p = shmgetat(1, 2);
              printf(1, "Child sharing key 1\n");
              test = (int *)p;
              *test = 40;
              *x = 140;
              printf(1, "Child: %x, %d, %x, %d\n", test, *test, x, *x);
      }
}
else{
      wait();
      printf(1, "Before sharing key 1\n");
      printf(1, "Parent: %x, %d, %x, %d\n", test, *test, x, *x);
      p = shmgetat(1, 4);
      printf(1, "After sharing key 1\n");
      test = (int *)p;
      printf(1, "Parent: %x, %d, %x, %d\n", test, *test, x, *x);
}
exit();
```

Results

```
$ shmgetat
This page will be shared:0, 8dfbb000, 7ffff000
This page will be shared:1, 8dfba000, 7fffe000
Grand Child: 7FFFE000, 30, 2FBC, 130
This page will be shared:2, 8dfb9000, 7fffd000
This page will be shared:3, 8dfb8000, 7fffc000
Child sharing key 1
Child: 7FFFC000, 40, 2FBC, 140
Before sharing key 1
Parent: 7FFFE000, 30, 2FBC, 100
This page will be shared:2, 8dfb9000, 7fffd000
This page will be shared:3, 8dfb8000, 7fffc000
After sharing key 1
Parent: 7FFFC000, 40, 2FBC, 100
```

Usage of Functions

- V2P(a) (((uint) (a)) – KERNBASE)
  - Translate a kernel VA to a PA
- P2V(a) (((void *) (a)) + KERNBASE)
  - Translate a PA to a kernel VA
- PTE_ADDR(pte *pte)
  - Translate from a PTE to physical page number
- Char* kalloc(void)
  - Allocate one 4096-byte page of physical memory
  - Returns a pointer that the kernel can use
  - Returns 0 if the memory cannot be allocated
- static int mappages(pde_t *pgdir, void *va, uint size, uint pa, int perm)
  - Create PTEs for virtual addresses starting at va that refer to physical addresses starting at pa
  - va and size might not be page-aligned
  - Perm: PTE_U, PTE_W, PTE_P…

More Functions

- walkpgdir(pde_t *pgdir, const void *va, int alloc)
  - Search a PTE or create a new PTE
  - If allocate == 0, return the address of the PTE in page table pgdir that corresponds to virtual address va
  - If alloc!=0, create a new page and initialize a new PTE based on va, set the new PTE with permission PTE_P, PTE_W and PTE_U
- Int deallocuvm(pde_t *pgdir, uint oldsz, uint newsz)
  - Deallocate user pages to bring the process size from oldsz to newsz
  - **You need to modify this function to make sure shared pages won't be freed when one process exits**
- pde_t* copyuvm(pde_t *pgdir, uint sz)
  - Given a parent process's page table, create a copy of it for a child
  - Modify it to make sure a child can inherit shared pages from parent

One way to implement key

SharedMem[PAGES]

index	Virtual Addr
0	8dfbb000
1	8dfba000
2	8dfb9000
3	8dfaa000
4	8dfc8000
5	8dffe000
6	8dfbe000
7	8dfbf000
8	8dfc0000
9	8dfc1000 ….

KN[KEYS]

key	Total Num
0	1
1	3
2	4
3	6
4	9
5	…
6	…
7	…
8	…
9	… ….

num

1

2

1

2

3

Work items to enable keys

- Define a global variable g_k and a global array KN[KEYS] to track which pages have been shared
- Manipulate g_k and KN to map their items to allocated memory pages ShareMem[PAGES]
- Key Num KN[key]

Key	Num	KN[key]
0	3	3
1	2	5
2	1	6
3	1	7

KN[key] keeps the total number of pages up to now shared by different processes, the values can be used as indices to track which items in ShareMem[PAGES] have been shared.

For example, key 3 has one item shared, which is corresponding to index 6 in ShareMem[PAGES]

Submission

- Capture screenshots for source code, compiling process, and results
- Combine them into a pdf file and submit it to moodle

- Due: March 26 (Monday), 11:55pm