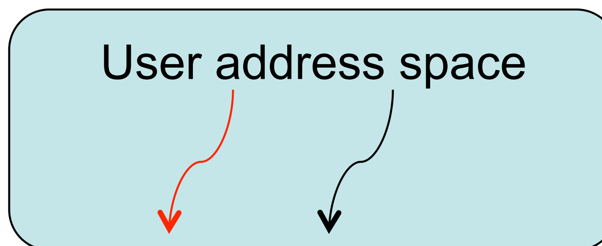
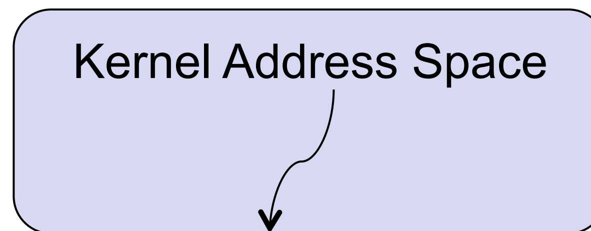
Group Exercises

- Learning Objectives:
  - Be able to follow threads of execution across domain crossings
  - Have a clear mental model of the critical data structures and state you'll need to maintain in the operating system in order to implement user-level processes.

Process Cartoons

- If we ask you to draw “process cartoons,” we mean diagrams like this one – you may have to draw a sequence of them or come up with a suitable way to represent animation. You should also include a bit more detail such as kernel stacks. As you draw the diagrams, as you discover new things you want to keep track of, think about what data structures will hold these new “things.”





Exercise 0: Warmup

1. What does OS161 use instead of RUNNABLE for processes/threads that are not currently running, but could?
2. List all the things that you think have to happen to create a new process.

Exercise 1: Getting warmer

- Draw a cartoon of a user process making a system call (e.g., getpid) that the OS can handle immediately, returning to the invoking process.
  7. Does this involve a thread switch?
  8. Does this involve a domain crossing?
  9. Does this involve more than one domain crossing?
  10. Must the kernel return to the same process that made this call?
- After you have completed your drawing, think about what your drawing implies for assignment 2. Does it suggest any particular data structures or standard functions you'll need?

Exercise 2: A bit trickier

- This time, draw a cartoon of a process making a system call that is going to block (e.g., read), causing the kernel to run some other thread.

- Again, after you've completed your drawing, discuss any implications this sequence of events has on the design of assignment 2.

Exercise 3: The Biggie

- Draw a cartoon of a fork system call.
  - Think carefully about what it means to create a new process. What structures do you have to conjure up? What data structures do you need to allocate? Where should those data structures live?
- And, once again, after you've got a diagram or sequence of diagrams, think about the implications for your design and implementation in assignment 2.

A Note on Error Cleanup

- Cleanup in the kernel is especially important!
  - Why?
- Designing constructor/destructor function pairs is typically a good idea (write them at the same time so you are sure to free all resources you allocate).
- Errors during creation often require partial cleanup; there are two commonly used strategies:
  1. Have a common exit path and branch into it at the proper place so you free the resources you've allocated so far.
  2. Repeat the code that reclaims resources at each error point.
- Much of the OS161 code uses approach #2, but either is fine; make a conscious decision which works better for you.

Compare and Contrast (discuss)

```
struct semaphore *
sem_create(const char *name, unsigned initial_count)
{
        struct semaphore *sem;
        sem = kmalloc(sizeof(*sem));
        if (sem == NULL) {
                return NULL;
        }

        sem->sem_name = kstrdup(name);
        if (sem->sem_name == NULL) {
                kfree(sem);
                return NULL;
        }
        sem->sem_wchan = wchan_create(sem->sem_name);
        if (sem->sem_wchan == NULL) {
                kfree(sem->sem_name);
                kfree(sem);
                return NULL;
        }
        spinlock_init(&sem->sem_lock);
        sem->sem_count = initial_count;

        return sem;
}
```

```
struct semaphore *
sem_create(const char *name, unsigned initial_count)
{
        struct semaphore *sem;
        sem = kmalloc(sizeof(*sem));
        if (sem == NULL)
              goto err;

        sem->sem_name = kstrdup(name);
        if (sem->sem_name == NULL)
                goto err;

        sem->sem_wchan = wchan_create(sem->sem_name);
        if (sem->sem_wchan == NULL)
                goto err1;

        spinlock_init(&sem->sem_lock);
        sem->sem_count = initial_count;

        return sem;
err1: kfree(sem_name);
err:  if (sem != NULL)
                kfree(sem);
        return NULL;
}
```