Lab8-Creating Thread

Instructor: Yaoqing Liu

TA: Garegin Grigoryan

Goals

- Understand the differences between a process and a thread
- Understand the process of adding a real kernel thread to xv6
- Creating threads by your user program through a system call interface
- Learn how to use lock for synchronization
- Understand how to test a multi-thread program

Retrieve xv6 and Test Program

- Login to odin

$ssh YourName@odin.cslabs.clarkson.edu

$cd ~/cs444-s18/Lab8

- Download xv6.tar.gz file to your work directory Lab8

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

- Unzip it

$tar –xzvf xv6.tar.gz

- Download test program and move it to your user folder

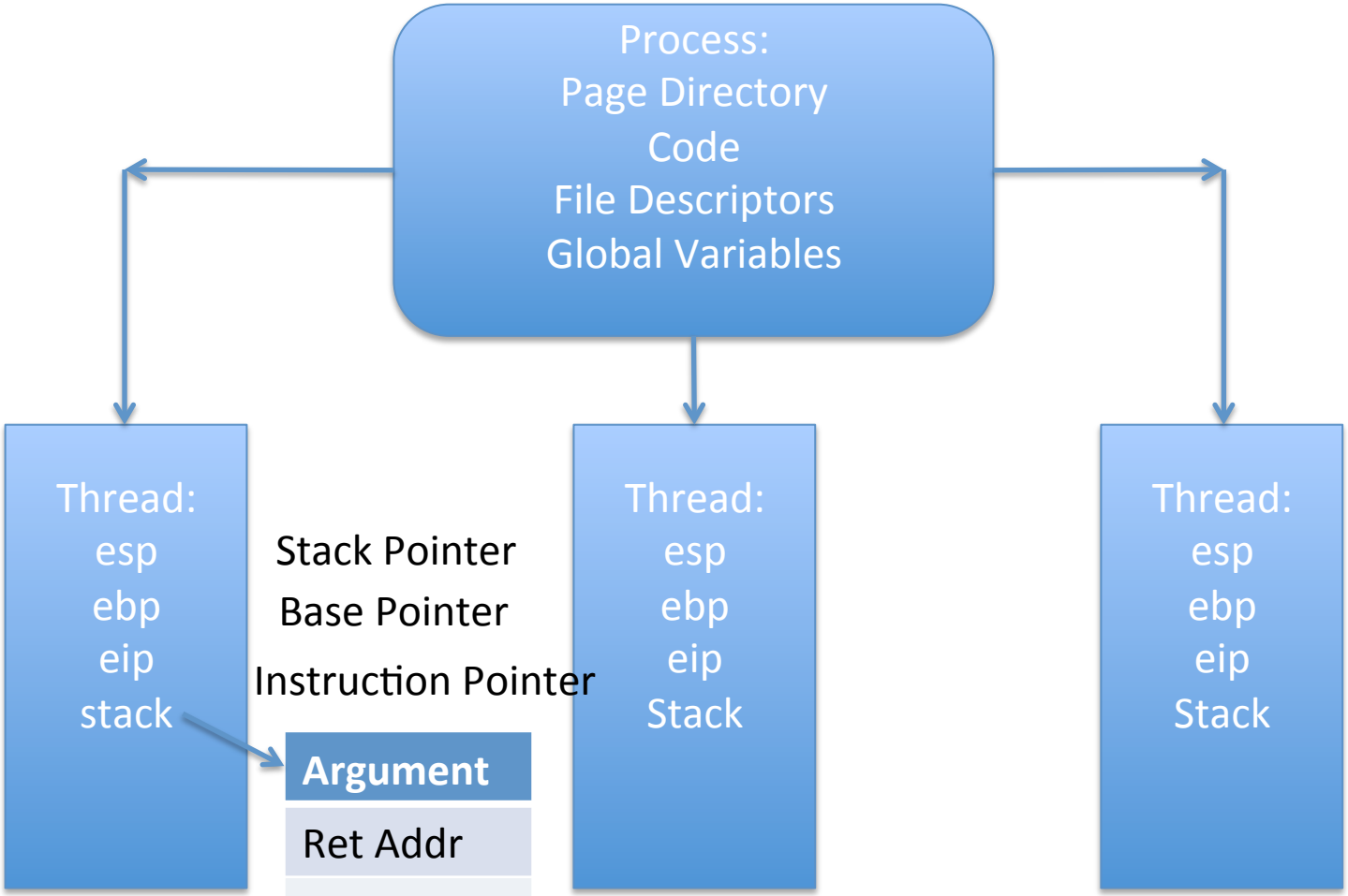
$wget http://people.clarkson.edu/~liu/CS444/Spring18/Lab8.c

$mv Lab8.c xv6/userspace/YourName.c

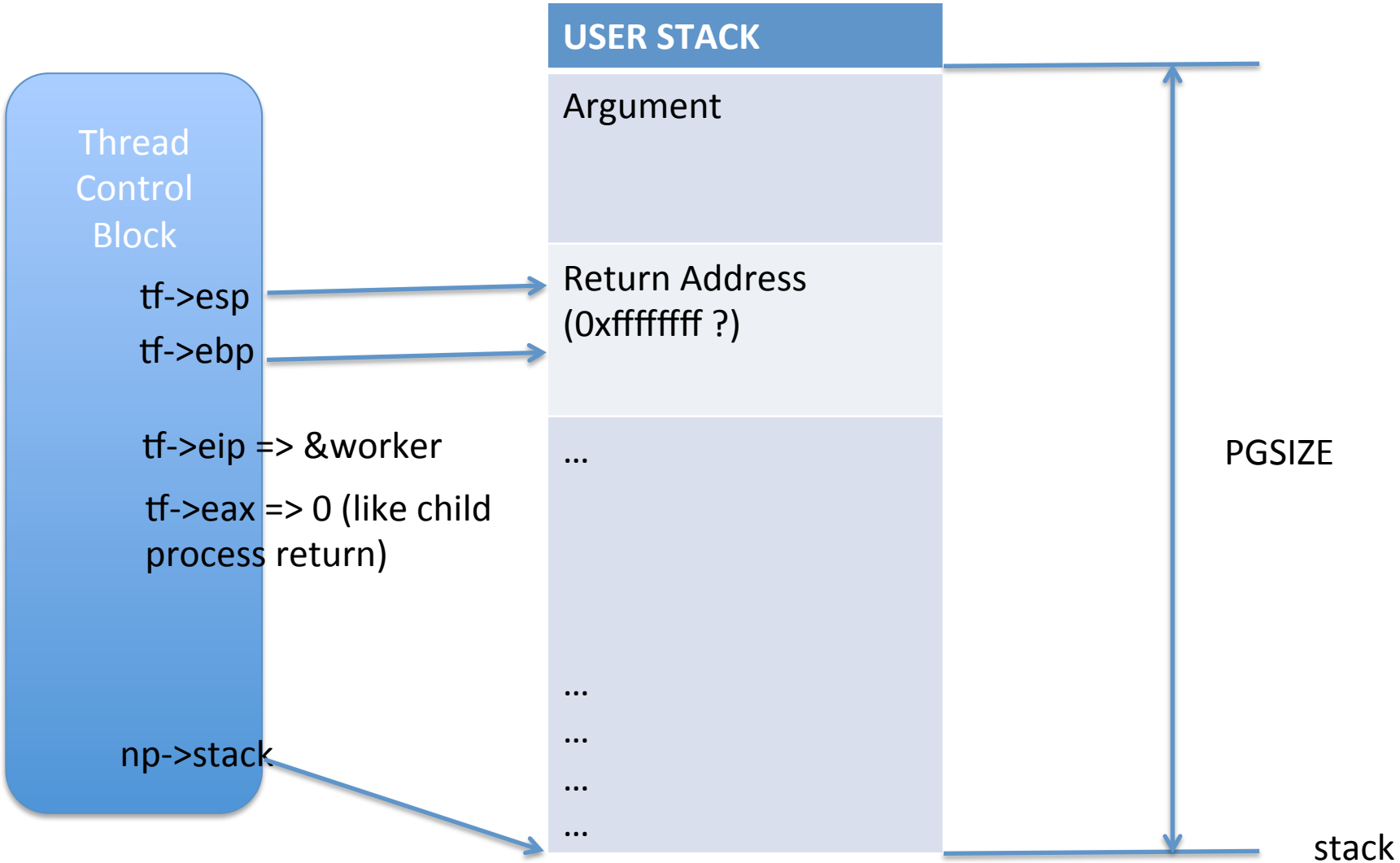
- In userspace/user.h file, add the following structure:

```
typedef struct __lock_t{
            uint flag;
 }lock_t;
```

Process and Thread



Your Code After Clone



Tasks

- Implement a clone_YourName system call:

int clone(void (*func)(void *), void *arg, void *stack)

  - It creates a new kernel thread sharing the same address space as its parent process
  - Func: the thread function to run
  - Arg: the argument to pass to the thread function
  - Stack: user defined stack, given in the test program
  - It returns the newly created thread ID (similar to process ID)

Steps

- Add a new attribute “stack” in the process data structure (a thread has the same data structure as a process)
- Define a new system call clone_YourName to pass the three parameters correctly to the kernel
- Implement the system call (similar to fork), but need to change some values (registers, stack, etc.) specifically for the thread
- When the thread exits, DO NOT free its parent process’s memory, how to differentiate a thread?
- Use the test program to test

System Call Routine Steps

- **Userspace/Makefile**
  Register your user program
- **userspace/usys.S**
  SYSCALL(*sc*)
- **syscall.h**
  #define SYS_*sc* <newID>
- **sysproc.c**
  int sys_*sc*(void) { return *sc*(); }
- **userspace/user.h**
  int *sc*(void);
- **syscall.c**
  extern int sys_sc(void);
  [SYS_*sc*] sys_*sc*,
- **defs.h**
  int *sc*(void);
- **Implement** int *sc*(void) {….your code...} in **proc.c**

Part of the Test Program

```
// total = 1
void worker(void *arg){
        int *test = (int *)arg;
        lock_t lk;
        lock_init(&lk);
        lock_acquire(&lk);
        total += *test;
        printf(1, "In worker: %d\n", total);
        lock_release(&lk);
}
int
main(void)
{
        int arg = 10;
        int result = thread_create(worker, &arg);
        result = thread_create(worker, &arg);
        result = thread_create(worker, &arg);
        result = thread_create(worker, &arg);
        while(total == 1){
                printf(1, "Result: %d, %d\n", result, total);
        }
        printf(1, "Result: %d, %d\n", result, total);
        exit();
}
```

```
int total = 1;

int lock_init(lock_t *lk)
{
        lk->flag = 0;
        return 0;
}

void lock_acquire(lock_t *lk){
        while(xchg(&lk->flag, 1) != 0);
}

void lock_release(lock_t *lk){
        xchg(&lk->flag, 0);
}
int thread_create(void (*start_routine)(void*), void *arg){
        lock_t lk;
        lock_init(&lk);
        lock_acquire(&lk);
        void *stack= malloc(PGSIZE*2);
        lock_release(&lk);
        if((uint)stack % PGSIZE)
                stack = stack + (PGSIZE - (uint)stack % PGSIZE);
        int result = clone(start_routine,arg,stack);
        free(stack);
        return result;
}
```

Testing Results

Initial total = 1

```
$ liu
func: 90, arg: 2fcc, 10, stack: 9000
func: 90, arg: 2fcc, 10, stack: 9000
func: 90, arg: 2fcc, 10, stack: 9000
func: 90, arg: 2fcc, 10, stack: 9000
Result: 7, 1
ReIn worker: 21
pid 5 liu: trap 14 err 5 on cpu 1 eip 0xffffffff addr 0xffffffff--kill proc
In worker: 31
pid 6 liu: trap 14 err 5 on cpu 1 eip 0xffffffff addr 0xffffffff--kill proc
In worker: 41
pid 7 liu: trap 14 err 5 on cpu 1 eip 0xffffffff addr 0xffffffff--kill proc
sult: 7, 1
Result: 7, 41
```

Walking Through

Submission

- Capture screenshots for your source code, compiling process, and results
- Convert them into a PDF file and submit on moodle

- Due: April 7 (Monday), 11:55pm