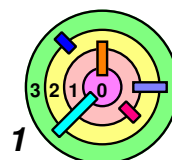


More On Kernel 1

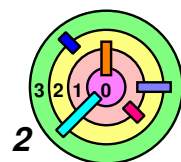
Bill Cheng

<http://merlot.usc.edu/william/usc/>



Kernel Code

- ➡ We will go over *some* kernel code now (we will not put all the code in "faber_test.c" and "sunghan_test.c" on these slides)
- we will not cover all the test code
 - we will probably cover a *small number of test cases*
 - you need to learn how to read all these code and use them to figure out what you need to do
 - if you have a specific test you want the instructor to talk about next Friday, please send him an e-mail before 5pm on Thursday
- ➡ **IMPORTANT:** at any line in `faber_thread_test()`, ask yourself
- 1) where are all the threads/processes?
 - ◆ i.e., in which *queue* is a thread sleeping
 - 2) if a thread is not in the run queue, *who* is going to *unblock* it and *when/how* (by calling what function)?
 - 3) exactly how and where will a particular process/thread die?

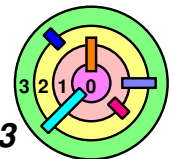


faber_thread_test ()

- ➡ Let's look at the first set of tests in `faber_thread_test ()`
- ▬ first subtest

```
dbg(DBG_TEST, "waitpid any test\n");
start_proc(&pt, "waitpid any test", waitpid_test, 23);
wait_for_any();

/*
 * Create a process and a thread with the given name
 * and calling teh given function. Arg1 is passed to
 * the function (arg2 is always NULL). The thread
 * is immediately placed on the run queue. A
 * proc_thread_t is returned, giving the caller a
 * pointer to the new process and thread to
 * coordinate tests. NB, the proc_thread_t is
 * returned by value, so there are no stack problems.
 */
static void start_proc(proc_thread_t *ppt, char *name,
    kthread_func_t f, int arg1) {
    ...
}
```



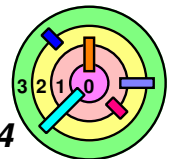
faber_thread_test()

- ➡ Let's look at the first set of tests in `faber_thread_test()`
- ▬ first subtest

```
static void start_proc(proc_thread_t *ppt, char *name,
    kthread_func_t f, int arg1) {
    proc_thread_t pt;

    pt.p = proc_create(name);
    pt.t = kthread_create(pt.p, f, arg1, NULL);
    KASSERT(pt.p && pt.t &&
        "Cannot create thread or process");
    sched_make_runnable(pt.t)
    if (ppt != NULL) {
        memcpy(ppt, &pt, sizeof(proc_thread_t));
    }
}

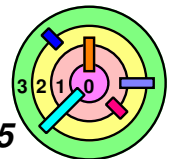
void *waitpid_test(int arg1, void *arg2) {
    do_exit(arg1);
    return NULL;
}
```



faber_thread_test()

- ➡ Let's look at the first set of tests in `faber_thread_test()`
- ▬ first subtest

```
/**
 * Call waitpid with a -1 pid and print a message
 * about any process that exits.
 * Returns the pid found, including -ECHILD when this
 * process has no children.
 */
static pid_t wait_for_any() {
    int rv;
    pid_t pid = do_waitpid(-1, 0, &rv);
    if (pid != -ECHILD)
        dbg(DBG_TEST, "child (%d) exited: %d\n", pid, rv);
    return pid;
}
```

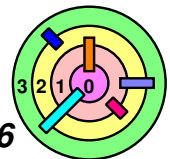


faber_thread_test()

- ➡ Let's look at the first set of tests in `faber_thread_test()`
- ▬ 2nd subtest

```
dbg(DBG_TEST, "waitpid test\n");
start_proc(&pt, "waitpid test", waitpid_test, 32);
pid = do_waitpid(2323, 0, &rv);
if ( pid != -ECHILD )
    dbg(DBG_TEST, "Allowed wait on non-existent pid\n");
wait_for_proc(pt.p);
```

```
void *waitpid_test(int arg1, void *arg2) {
    do_exit(arg1);
    return NULL;
}
```

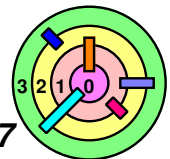


faber_thread_test()

- ➡ Let's look at the first set of tests in `faber_thread_test()`
- ▬ 2nd subtest

```
/**
 * Call do_waitpid with the process ID of the given
 * process. Print a debug message with the exiting
 * process's status.
 */
static void wait_for_proc(proc_t *p) {
    int rv;
    pid_t pid;
    char pname[PROC_NAME_LEN];

    strncpy(pname, p->p_comm, PROC_NAME_LEN);
    pname[PROC_NAME_LEN-1] = '\0';
    pid = do_waitpid(p->p_pid, 0, &rv);
    dbg(DBG_TEST, "%s (%d) exited: %d\n", pname, pid, rv);
}
```

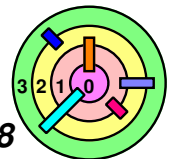


faber_thread_test ()

- ➡ Let's look at the first set of tests in `faber_thread_test ()`
- ▬ 3rd subtest

```
dbg(DBG_TEST, "kthread exit test\n");  
start_proc(&pt, "kthread exit test",  
          kthread_exit_test, 0);  
wait_for_proc(pt.p);
```

```
/*  
 * A thread function that returns NULL, silently  
 * invoking kthread_exit()  
 */  
void *kthread_exit_test(int arg1, void *arg2) {  
    return NULL;  
}
```



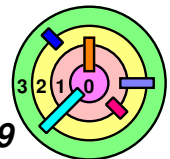
faber_thread_test()

- ➡ Let's look at the first set of tests in `faber_thread_test()`
- ▬ 4th subtest

```
dbg(DBG_TEST, "many test\n");
for (i = 0; i < 10; i++)
    start_proc(NULL, "many test", waitpid_test, i);
wait_for_all();
dbg(DBG_TEST, " (C.1) done\n");
```

```
void *waitpid_test(int arg1, void *arg2) {
    do_exit(arg1);
    return NULL;
}
```

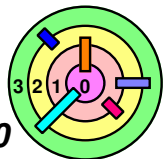
```
/* Repeatedly call wait_for_any() until it returns
 * -ECHILD */
static void wait_for_all() {
    while (wait_for_any() != -ECHILD)
        ;
}
```



faber_thread_test()

- ➡ Let's look at the first set of tests in `faber_thread_test()`
- ▢ 4th subtest

```
/**
 * Call waitpid with a -1 pid and print a message
 * about any process that exits.
 * Returns the pid found, including -ECHILD when this
 * process has no children.
 */
static pid_t wait_for_any() {
    int rv;
    pid_t pid = do_waitpid(-1, 0, &rv);
    if (pid != -ECHILD)
        dbg(DBG_TEST, "child (%d) exited: %d\n", pid, rv);
    return pid;
}
```



More Test Code

- ➡ We will not put all the code in "faber_test.c" and "sunghan_test.c" on these slides
 - in the discussion sections, we will discuss more of these test code
 - we will not cover all the test code
 - ◆ you need to learn how to read all these code and use them to figure out what you need to do
 - ◆ we will probably cover a *small number of test cases*
 - if you have a specific test you want the TA to talk about, please send him an e-mail before 5pm on Thursday
- ➡ **IMPORTANT:** at any line in `faber_thread_test()`, ask yourself
 - 1) where are all the threads/processes?
 - ◆ i.e., in which *queue* is a thread sleeping
 - 2) if a thread is not in the run queue, *who* is going to *unblock* it and *when/how* (by calling what function)?

