CS 354 Fall 2025

Lab 5: Kernel data structures [20 pts]

Due: 10/03/2025 (Fri), 11:59 PM

Background

Chapter 7,8 from the Xinu textbook.

You Will Learn

Key kernel data structures and their contents.

Source code

Start from a fresh copy of Xinu xinu-fall2025.tar.gz by running the command:
`tar zxvf /homes/cs354/xinu-fall2025.tar.gz`

The Problem To Be Solved

This lab has functions that use busy waiting, we need to compile it with no optimization (flag -O0). This step is exactly the same as the one in Lab1.

0. Modify the `Makefile`

- In `Makefile` located in the `compile/` directory, modify this line:

```
CFLAGS  = -mcpu=cortex-a8 -mno-unaligned-access -marm
-fno-builtin -fno-stack-protector -nostdlib -c -Wall
-O ${DEFS} ${INCLUDE}
```

to

```
CFLAGS  = -mcpu=cortex-a8 -mno-unaligned-access -marm
-fno-builtin -fno-stack-protector -nostdlib -c -Wall
-O0 ${DEFS} ${INCLUDE}
```

Note: This modification turns off compiler optimization so we can simulate concurrent modifications to the same global variables.
Note: The added character is the numeric '0'(zero), not the letter 'o'.

1. Write a function `xdump`, which uses `kprintf` to:

   Dump the process table by printing the following fields for each entry:
   - process ID
   - name
   - state (e.g. READY…)
   - priority
   - message, if one is waiting (or "NO MESSAGE")
   - stack base (print in hex e.g. "0xABCD1234")
   - stack pointer (except for the current process e.g. "0xABCD1234")
   - semaphore ID, if waiting on one

   Dump the semaphore table by printing the following
   - semaphore ID
   - count
   - list of process IDs, if any are waiting

   Dump the sleep queue by printing the following for each sleeping process
   - process ID
   - time remaining. Print seconds and milliseconds, such as "2sec 12ms"

2. To test `xdump`, As in previous labs, remove the items in `main.c` that run the Xinu shell. Then insert the following:

Create a `semaphore` with count 0 as a global variable `sid`.

Create seven processes. Resume each of them **except A** (in alphabetical order, B -> C, …, ->G)

Name	**Priority**	**Make it do the following**
A	15	directly exits ● Only create the process, but do not resume A
B	10	wait on `semaphore sid;` exit
C	11	wait on `semaphore sid;` exit
D	10	run a busy loop 100000 times; exit
E	12	run a busy loop 100000 times; exit
F	10	sleep for 4 seconds; exit
G	10	sleep for 1 seconds; exit

Subsequently, the main function should do the following **in order**

1. Send message `354` to process A
2. Call `xdump`
3. Resume process A
4. Signal `sid`
5. Sleep for 2 seconds
6. Call `xdump` again

Questions

Place **short answers** to the following questions in a file named `questions.pdf`. In the case of using document software (e.g., Word), please make sure to convert the file

to **PDF**.

We only consider the Main and the other created processes A,B,C,D,E,F,G

1. In the first `xdump` (step 2), list the process(es) you observe and the corresponding process states (e.g. READY).
2. In the second `xdump` (step 6), list the process(es) you observe and corresponding process states (e.g. READY).
3. In the second `xdump` (step 6), explain why any processes are sleeping, if they are, or explain why none are sleeping.
4. After `signal(sid)`, which waiting process (B or C) will acquire the semaphore first, and why?
5. If processes B and C have the same priority and are resumed in alphabetical order, which process will acquire the semaphore first when `signal(sid)` is called?

Submission

General instructions:

When implementing code in the labs, please maintain separate versions/copies of code so that mistakes such as unintentional overwriting or deletion of code is prevented. This is in addition to the efficiency that such organization provides. You may use any number of version control systems such as GIT and RCS. Please make sure that your code is protected from public access. For example, when using GIT, use git that manages code locally instead of its on-line counterpart github. If you prefer not to use version control tools, you may just use manual copy to keep track of different versions required for development and testing. More vigilance and discipline may be required when doing so.

You must submit a directory named `lab5` that contains both required files:

- Your final version of `main.c` from Xinu
- `questions.pdf`, containing your answers

Go to the directory where `lab5` is a subdirectory.

For example, if `/homes/alice/cs354/lab5` is your directory structure, go to `/homes/alice/cs354`.

Type the following command to submit the directory with `turnin`:

```
turnin -c cs354 -p lab5 lab5
```

Be sure the files inside the directory are named exactly `main.c` and `questions.pdf`.

You can check/list the submitted files using

```
turnin -c cs354 -p lab5 -v
```