Homework: sleep and wakeup

This assignment requires the files xv6.pdf and xv6_rev0.zip. You may download them from the Assignments page.

Read: pipe.c

Hand-In Procedure

You are to turn in this homework at the beginning of lecture. Please write up your answers to the questions below and hand them in to a 6.828 staff member at the beginning of lecture.

Introduction

Remember in lecture 7 we discussed locking a linked list implementation. The insert code was:

```
struct list *l;
l = list_alloc();
l->next = list_head;
list_head = l;
```

and if we run the insert on multiple processors simultaneously with no locking, this ordering of instructions can cause one of the inserts to be lost:

```
CPU1                                CPU2

struct list *l;
l = list_alloc();
l->next = list_head;
                                    struct list *l;
                                    l = list_alloc();
                                    l->next = list_head;
                                    list_head = l;
list_head = l;
```

(Even though the instructions can happen simultaneously, we write out orderings where only one CPU is "executing" at a time, to avoid complicating things more than necessary.)

In this case, the list element allocated by CPU2 is lost from the list by CPU1's update of list_head. Adding a lock that protects the final two instructions makes the read and write of list_head atomic, so that this ordering is impossible.

The reading for this lecture is the implementation of sleep and wakeup, which are used for coordination between different processes executing in the kernel, perhaps simultaneously.

If there were no locking at all in sleep and wakeup, it would be possible for a sleep and its corresponding wakeup, if executing simultaneously on different processors, to miss each other, so that the wakeup didn't find any process to wake up, and yet the process calling sleep does go to sleep, never to awake. Obviously this is something we'd like to avoid.

Read the code with this in mind.

Questions

(Answer and hand in.)

1. How does the proc_table_lock help avoid this problem? Give an ordering of instructions (like the above example for linked list insertion) that could result in a wakeup being missed if the proc_table_lock were not used. You need only include the relevant lines of code.

2. sleep is also protected by a second lock, its second argument, which need not be the proc_table_lock. Look at the example in ide.c, which uses the ide_lock. Give an ordering of instructions that could result in a wakeup being missed if the ide_lock were not being used. (Hint: this should not be the same as your answer to question 2. The two locks serve different purposes.)

This completes the homework.