CPSC 8220 MIDTERM EXAM, SPRING 2016

NAME: _____

1. (5) In our warm-up C code that was used to configure a NetApp file server,we see the line

   ```
   while(wait(&status)!=pid);
   ```

   The *wait* suspends us until we receive any signal. Upon receipt of one particular signal, *wait* will return the pid we seek. Which signal?

   A. SIGCONT
   B. SIGSTOP
   C. SIGURG
   D. SIGCHLD
   E. SIGUSR1
   F. SIGKILL
   G. SIGPID

2. (5) What is the size (in bytes) of the CONTROL region for the Kyouko3 graphics card?

3. (5) In the driver for the Kyouko3 graphics card, we used two indices, *fill* and *drain* into a circular queue of DMA buffers to keep track of the state of the buffer queue. If, upon entry to the DMA completion interrupt handler, we find *fill* == *drain*, what is the state of the buffer queue?

   A. empty
   B. full
   C. empty except for the 1 slot that just filled to generate the interrupt
   D. full except for the 1 slot that just emptied to generate the interrupt
   E. error condition: a queued buffer was empty
   F. error condition: a queued buffer had a faulty address
   G. error condition: SMP race condition detected

4. (5) The Kyouko3 graphics card has two registers devoted to interrupts. One of these is used to enable/disable interrupts, and the other is used to report status, i.e., whether an interrupt has just occurred and needs to be handled/cleared. What are the numerical offsets (in hex) for these two registers? Which is the status register?

5. (5) As noted in the previous question, the status register of the Kyouko3 card could be read to determine whether a DMA interrupt has just occurred. Which bit having which value indicates an outstanding interrupt?

   A. bit 0 value 0
   B. bit 0 value 1
   C. bit 1 value 0
   D. bit 1 value 1
   E. bit 2 value 0
   F. bit 2 value 1
   G. bit 4 value 0
   H. bit 4 value 1
   I. bit 31 value 0
   J. bit 31 value 1

6. (10) In our design of the driver for the Kyouko3 card, we used the *mmap* system call to memory map the device's control registers and framebuffer, and we used the *ioctl* system call with command BIND_DMA to memory map DMA buffers. This led to an asymmetry wherein we had to switch within *kyouko3_mmap* to accommodate all calling paths. The switch was based on a somewhat artificial offset recovered from *vma→vm_pgoff*. Instead, we could leave the driver's mmap function fixed as:

```
int kyouko3_mmap(struct file *fp, struct vm_area_struct *vma)
{
io_remap_pfn_range(vma,vma->vm_start,vma->vm_pgoff,
        vma->vm_end - vma->vm_start,vma->vm_page_prot);
return 0;
}
```

   and, instead of calling mmap() from user-level, call ioctl() with a new command, BIND_CONTROL, or a new command, BIND_FRAMEBUFFER, that memory maps the device's control region or framebuffer to user space and delivers, in the argument, the user-level base address of this region. Provide the kernel code for the new implementation of BIND_CONTROL. The return

value should be a binary success/failure indicator. You may NOT change the *kyouko3_mmap* function from that suggested above or the link to it in *kyouko3_fops*.

7. (10) Linux maintains time internally as the number of seconds (and a fraction thereof, in nanoseconds) that have elapsed since January 1, 1970. Values can be obtained by calling the kernel function, *current_kernel_time( )*, which returns a *struct timespec*, a structure consisting of two longs, *tv_sec* and *tv_nsec*. Complete the following code for a system call, *ticktock(long *arg)*, that returns to the user the number of seconds elapsed (since January 1, 1970) in its argument and a success/failure indicator as a value. You may ignore the nanoseconds.

```
#include <linux/linkage.h>
#include <linux/kernel.h>
#include <linux/syscalls.h>
#include <asm/uaccess.h>
#include <linux/time.h>

SYSCALL_DEFINE
{









return(0);
}
```

8. (5) Provide the full path file name for the system call table in which you must add an entry to implement the system call of the previous question. Also, provide the entry.

9. (10) Provide the user-level calling code for the system call of the previous question.

10. (10) A simple, but often useful kernel function is:

```
static inline struct pid *task_pid(struct task_struct *task)
{
return task->pids[PIDTYPE_PID].pid;
}
```

Suppose we have defined two process table pointers:

```
struct task_struct *p, *t;
```

Give a single line of kernel code that would print the numerical process id (as seen in the process namespace of the current process) for each thread t in process p. You may assume that all #include files from signal.c have been included.

11. (5) Suppose a target process with process table entry *struct task_struct *p* has received multiple signals that were sent to it via the *kill* system call. In which kernel variable would the pending signal bits be stored until *p* calls *do_signal()* to process pending signals?

    A. *p→pending.sig[0]*
    B. *p→pending.sig[1]*
    C. *p→pending.signal.sig[0]*
    D. *p→pending.signal.sig[1]*
    E. *p→signal→shared_pending.sig[0]*
    F. *p→signal→shared_pending.sig[1]*
    G. *p→signal→shared_pending.signal.sig[0]*
    H. *p→signal→shared_pending.signal.sig[1]*

12. (5) If we attempt to send signal number 0 to a process whose real user id in the global user namespace matches our own effective user id in the global user namespace, what value is returned by *sys_kill()*?

    A. EINVAL
    B. ESRCH
    C. EPERM
    D. the numerical pid of the target process
    E. the uid in the global user namespace of the target process
    F. -1
    G. 0
    H. 1

13. (5) Which signals are ignored when their handler is SIG_DFL?

14. (5) When a signal number below SIGRTMIN is sent to a process, which kernel function checks whether that signal is already pending for that process and returns a 1 in order to short-circuit the send?

    A. signalfd_notify
    B. complete_signal
    C. prepare_signal
    D. get_signal_to_deliver
    E. dequeue_signal
    F. legacy_queue
    G. is_si_special
    H. wants_signal

15. (5) The macro *sig_kernel_stop(sig)* evaluates to true for 4 signals. What are their names?

16. (5) When we are sending a stop signal by using *sys_kill*, which line of kernel code clears SIGCONT from the pending signals collection?