15-410, Spring 2025, Homework Assignment 1.
Due Monday, February 24, 20:59:59

Please observe the non-standard submission time, **which is not midnight**. As we intend to make solutions available on the web site promptly thereafter for exam-study purposes, please turn your solutions in on time.

Homework must be submitted in either PostScript or PDF format (not: Microsoft Word, Word Perfect, Apple Works, LaTeX, XyWrite, WordStar, etc.). Submit your answers by placing them in the appropriate hand-in directory, e.g., `/afs/cs.cmu.edu/academic/class/15410-s25-users/$USER/hw1/$USER.ps` or `/afs/cs.cmu.edu/academic/class/15410-s25-users/$USER/hw1/$USER.pdf`. A plain text file (.text or .txt) is also acceptable, though it must conform to Unix expectations, meaning lines of no more than 120 characters separated by newline characters (note that this is *not* the Windows convention). Please avoid creative filenames such as `hw1/my_15-410_homework.PdF`.

1 Tape drives (4 pts.)

Consider a system with four processes and seven tape drives. The maximal needs of each process are declared below:

Resource Declarations

Process A	Process B	Process C	Process D
5 tape drives	5 tape drives	3 tape drives	2 tape drives

Imagine the system is in the state depicted below. List one request which the system should grant right away, and one request which the system should react to by blocking the process making the request. Briefly justify each of your answers.

Who	Max	Has	Room
A	5	1	4
B	5	2	3
C	3	1	2
D	2	1	1
System	7	2	-

2 Device queue (6 pts.)

This semester you have had a bit of experience with writing drivers to interact with hardware devices. Historically speaking, device drivers constitute a large fraction of the effort of writing an operating system. In this question you will consider the operation of a simple device and a miniature driver for that device. The operation of the device will be presented in pseudo-code form, as if it were executed by a thread. This is not unreasonable, because hardware devices are complex state machines, and many of them actually contain processors which run large bodies of firmware written in languages like C (see "Indestructible malware by Equation cyberspies is out there — but don't panic (yet)", Serge Malenkovich, Kaspersky Daily, 2015-02-17).

Here is a summary of the "code base" shown below.

- The device can read host memory (using a technique frequently referred to as "direct memory access", aka "DMA").
- When it is not idle, the device operates on a queue of requests from the host. "Processing" each item in the queue takes "a while." You might imagine that each item in the queue is a beep tone which is to be played for 125 milliseconds.
- When the device reaches the end of the queue it becomes idle.
- From time to time the device driver on the host decides to enqueue a work item.
  - If the device is idle, the device driver on the host forms a new queue and activates the device.
  - If the device is not idle, the device driver on the host attaches the new item to the end of the queue.
- The host can inspect the device to see whether or not it is idle.
- As is the case for some hardware devices, it is "not ok" for the host to tell the device to start work when it is already working. In the code below, this requirement is manifested in the form of `affirm()` "statements" in the device "code"; in real hardware, activating an already-active device is more likely to cause it to forget what it was previously doing.
- Both `device()` and `enq()` are run by only one "thread" at a time.

(Continued on next page)

```
struct item;
typedef struct item item_t;
struct item {
  int todo;
  struct item *next;
};

int device_running = 0;   // status
int device_start = 0;     // command
item_t *first = NULL;     // producer/consumer-ish

void device() { // firmware inside device
    item_t *cur;

    while (1) {
        while (!device_start) continue;

        cur = first;
        device_start = 0;
        device_running = 1;

        while (cur) {
            perform(cur->todo); // takes milliseconds (a long time)
            cur = cur->next;
            affirm(!device_start);
        }
        device_running = 0;
    }
}

void enq(item_t *ip) { // run by host device driver
    item_t *cur, *prev, *clean;

    clean = NULL;
    ip->next = NULL;

    if (!device_running) {
        clean = first;
        first = ip;
        device_start = 1;
        while (!device_running) continue;
    } else {
        cur = first;
        while (cur) {
            prev = cur;
            cur = cur->next;
        }
        if (device_running) {
            prev->next = ip;
        } else {
            clean = first;
            first = ip;
            device_start = 1;
            while (!device_running) continue;
        }
    }
    while (clean) {
        cur = clean;
        clean = clean->next;
        free(cur);
    }
}
```

2.1 1 pt

What is the purpose of the `!device_running` busy-wait loops in the driver code? What problem could happen if they were deleted?

2.2 1 pt

Are those busy-wait loops "ok"? Why or why not?

2.3 4 pts

There is a concurrency problem with the "code base" shown above. Briefly state something that could go wrong and then present a trace which supports your claim. You may use more or fewer lines in your trace.

Execution Trace

time	`enq()`	`device()`
0		
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		