CSC-501: Operating Systems Principles (Spring 2015)

Flow Analysis

Please Read All Questions Carefully!

There are fifteen (15) total numbered pages.

Please put your Name, Unity ID, and student ID on THIS page, and JUST YOUR student ID (but NOT YOUR NAME or UNITY ID) on every other page.
Why are you doing this? So I can grade the exam anonymously. So, particularly important if you think I have something against you! But of course, I don't. Probably.

The exam is closed everything (books, notes, discussion, cell phone, and computer), but you can have one double-sided letter-size cheat-sheet. You have 180 minutes.

Your work must be individual. Cheating will be punished instantly. Please focus on your own exam.

Name: _____
Unity ID: _____
Student ID: _____

Student ID: _____

Grading Page

	Points	Total Possible
Q1		10
Q2		12
Q3		10
Q4		12
Q5		10
Q6		10
Q7		10
Q8		12
Q9		11
Q10		11
Q11		12
Q12		10
Q13		10
Total		140

Student ID: _____

This exam is about flow analysis …

> Data-flow analysis is a technique for gathering information about the possible set of values calculated at various points in a computer program. A program's control flow graph (CFG) is used to determine those parts of a program to which a particular value assigned to a variable might propagate. The information gathered is often used by compilers when optimizing a program. A canonical example of a data-flow analysis is reaching definitions.
>
> In computer science, control flow analysis (CFA) is a static code analysis technique for determining the control flow of a program. The control flow is expressed as a control flow graph (CFG). For many imperative programming languages, the control flow of a program is explicit in a program's source code. As a result, interprocedural control-flow analysis implicitly usually refers to a static analysis technique for determining the receiver(s) of function or method calls in computer programs written in a higher-order programming language.

Wait, this is an OS exam, but not PL exam. We are not going to develop automated flow analysis algorithm. Rather, you are asked to trace the **flow** of what is happening in some of the systems and code sequences we have studied.

For example, when we have a file system that uses *memory* for caching data, a *disk* for its persistent storage, and a *user buffer* in the application's address space to read data into, we might use these labels:

```
1. user-allocated memory
2. OS cache
3. disk
```

If a question then asked, "what is the flow of data when an application first reads data from disk?" you might answer, "3, 2, 1" because the data is read from the disk into the OS cache and then finally copied into the user-allocated memory. If the block is read again, the flow is simpler: "2, 1". Why? Because the data is fetched from the cache and avoids the disk altogether.

Of course, each question asks for slightly different flows and their meaning is specific to each question. Thus, please **read each question carefully!**

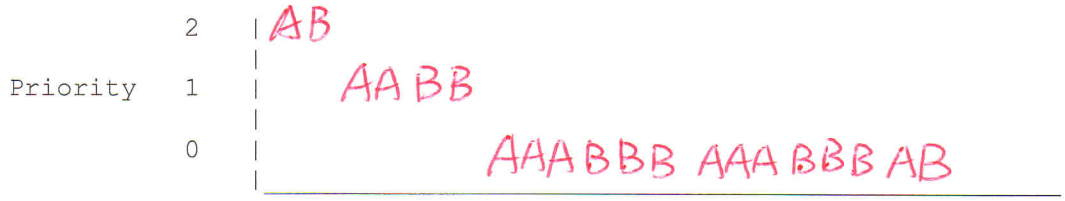
Good luck!

Student ID: _____

1. **MLFQ Flows:** Assume you have a multi-level feedback queue (MLFQ) scheduler. In this question, we'll draw a picture to show how the jobs flow through the system. In the drawing, the y-axis shows the PRIORTY of the jobs over time.

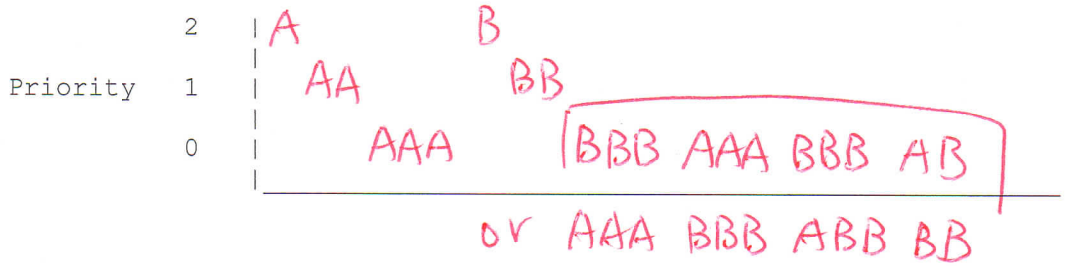
(a) Assume a 3-level MLFQ (high priority is 2, low priority is 0). Assume two jobs (A and B), both BATCH jobs (no I/O), each with a run-time of 10 time units, and both entering the system at T=0. Assume the quantum length at the highest priority level is 1, then 2 at the middle, and 3 for the lowest priority.

Draw a picture of how the scheduler behaves for these jobs. Make sure to LABEL the x-axis.



(b) Assume the same scheduling parameters as above. Now the jobs are different; both A and B are BATCH jobs that each run for 10 time units (no I/O again), but this time A enters at T=0 whereas B enters the system at T=6.

Draw a picture of how the scheduler behaves for these jobs. Make sure to LABEL the x-axis.



(c) Calculate the RESPOSE TIME and TURNAROUND TIME (in part b) for job A.

0 19

or 0 16

(d) Calculate the RESPOSE TIME and TURNAROUND TIME (in part b) for job B.

0 14

2. **TLB and Virtual Machine Monitor TLB Flows:** We have an application accessing memory running on an OS with a software managed TLB. Assume, for simplicity, a simple linear page table kept in physical memory. Here are some different hardware and software things that can happen during memory access:

```
 0. the hardware checks the TLB to see if the VPN-to-PFN (or VPN-to-MFN if
on a VMM) translation is present
 1. the hardware issues a load to a physical address(or machine address if
on a VMM)
 2. the OS code at the start of the TLB miss handler runs
 3. the OS code at the end of the TLB miss handler runs, which returns
from trap
 4. the OS code that accesses the page table to lookup a translation runs
 5. the OS code that updates the TLB with a new mapping runs
 6. the OS code that updates the page table with a new VPN-to-PPN mapping
runs
 7. a disk request is initiated
 8. a disk request completes
```

There are a few more possible events for these flows (for simplicity, just these three)

```
 9. the VMM code at the start of the VMM TLB miss handler runs
10. the VMM code at the end of the VMM TLB miss handler runs, which
returns from trap)
11. the VMM code that updates the TLB with a new mapping runs
```

At first, assume we are **not** running on a VMM, i.e., this is just an application running on top of the OS.

(a) Write down the flow that occurs when a user application encounters a **TLB hit**:

0, 1

(a) Write down the flow that occurs when a user application encounters a **TLB miss** to a valid page that is **present** in memory:

0, 2, 4, 3, 3, 0, 1

(b) Write down the flow that occurs when a user application encounters a **TLB miss** to a valid page that is **not present** in memory:

0, 2, 4, 7, 8, 6, 5, 3, 0, 1

Now assume that the OS is running on a virtual machine monitor.

(c) Write down the flow that occurs when a user application encounters a **TLB hit**:

We do not cover VMM this time.

(d) Write down the flow that occurs when a user application encounters a **TLB miss** to a valid page that is **present** in memory:

3. **Multiprocessor Scheduling:** Let us assume we have three jobs (A, B, C) to run, each job takes six time slices to finish, and two processors.

(a) Under some single-queue multiprocessor scheduling (SQMS) implementation, we have the following job schedule flows across CPUs:

```
CPU 0 [A C B A C B A C B]
CPU 1 [B A C B A C B A C]
```

Can you improve the above job schedule under SQMS? If yes, draw the improved schedule and explain the problem solved by the improved schedule.

This has cache affinity problem, and can be solved by

CPU 0: [AAABBBCCC]

CPU 1: [BBBACCCAAA] (There are many other possibilities)

(b) Under some multi-queue multiprocessor scheduling (MQMS) implementation, we have the following job schedule flows across CPUs:

```
CPU 0 [A A A A A A]
CPU 1 [B B C C B B C C B B C C]
```

Can you improve the above job schedule under MQMS? If yes, draw the improved schedule and explain the problem solved by the improved schedule.

This has load-balance problem, and can be solved by

CPU 0: [AAABBBCCC]

CPU 1: [BBBCCCAAA] (There are many other possibilities)

(c) For your answers on the improved schedules under (a) and (b), are they the same?
- If yes, does it conflict with the fact that one is for SQMS and the other is for MQMS?
- If no, does it imply that one of SQMS and MQMS is better than the other?

Explain your answer.

Yes or No: depends on your solution to a and b.

If yes: it does not conflict with...

If no: it does not imply that...

Student ID: _____

4. **Simple Paging Flow.** In this question, let's investigate the flow of memory being accessed during the execution of an instruction.

For this question, assume a linear page table, with a 1-byte page-table entry. Assume an address space of size 128 bytes with 32-byte pages. Assume a physical memory of size 128 bytes. The page-table base register is set to physical address 16. The contents of the page table are:

VPN	PFN
0	1
1	Not valid
2	3
3	Not valid

Now, finally assume we have the following instruction, which resides at virtual address 10 within the address space of the process, and loads a SIGNLE BYTE from virtual address 70 into register R1:

```
10: LOAD 70, R1
```

In the diagram of physical memory below:

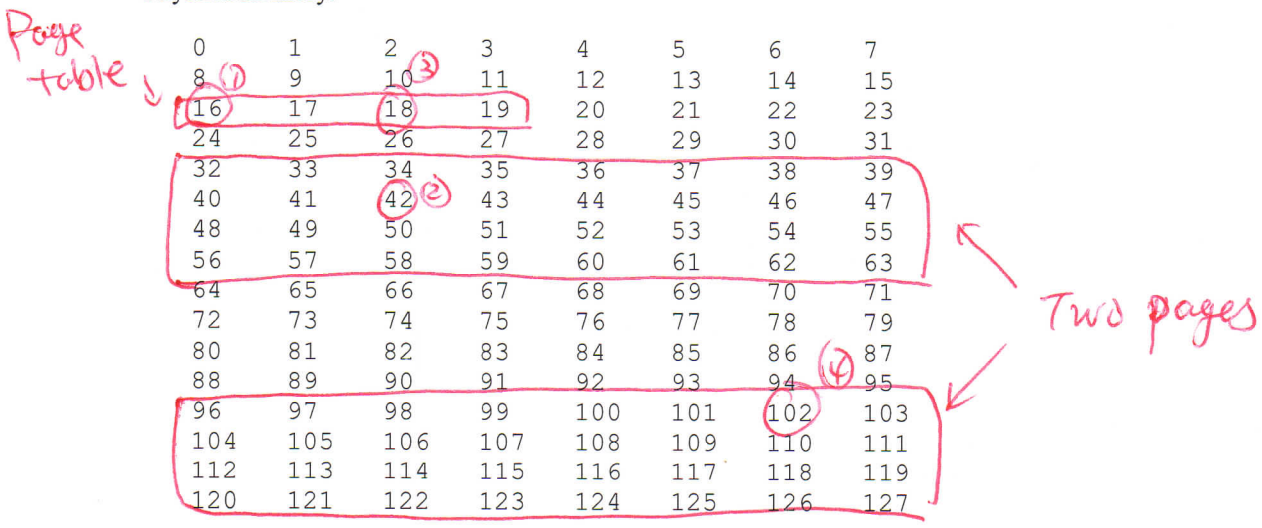
(a) Put a BOX around each valid virtual page (and label them)

(b) Put a BOX around the page table (and label it)

(c) Circle the memory addresses that get referenced during the execution of the instruction, including both instruction fetch and data access (there is no TLB).

(d) LABEL these addresses with a NUBMER that indicates the ORDER in which various physical addresses get referenced.

Physical memory:



0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71
72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87
88	89	90	91	92	93	94	95
96	97	98	99	100	101	102	103
104	105	106	107	108	109	110	111
112	113	114	115	116	117	118	119
120	121	122	123	124	125	126	127

Student ID: _____

5. **Virtual Memory Flow.** In this question, let's continue investigating the series (a.k.a., the flows) of memory and disk accesses a simple load instruction (e.g., `mov VirtualAddress, register`) takes to execute, given some different assumptions about the virtual memory system.

(a) Assume we have a linear page table per process, but the system we are using has no TLB and no swapping to disk. What is the flow that it takes to execute a single load instruction in terms of memory accesses? You will need to list the sequence of memory accesses.

4 memory accesses.

instruction page table then page, data page table then page.

(b) Now assume the same system, but with a TLB (still no swapping). What is the fastest (i.e., best case) memory-access flow the load instruction above will execute, assuming TLB hits?

2 memory accesses

instruction page, data page.

(c) Assume now we have a two-level page table with a page directory. Assume each reference to the TLB is a miss, but that all referenced pages are found in memory (no swapping again). What is the slowest (i.e., worst case) memory-access flow for the load instruction above?

6 memory accesses
instruction page dir, page table, then page
data page dir, page table, then page.

(d) Assume the same system again. This time the memory location(s) referenced by the load instruction also might be on disk (due to swapping). We further assume the page directory and page tables are always in memory. What is the slowest memory-access and disk-access flow for the instruction to execute?

4 memory accesses and 4 disk accesses.
instruction page dir (mem), page table (mem),
page (swap out), then page (swap in),
data page dir (mem), page table (mem),
page (swap out), then page (swap in).

Student ID: _____

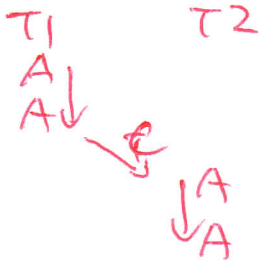
6. **Concurrency Flows.** Given a basic spin lock, assume that locking the spin lock takes *A* time units (if no one is holding the lock); unlock also takes *A* time units. Assume further that a context switch takes *C* time units, and that a time slice is *T* time units long. Also assume *2*A* is smaller than *T*.

Assume this code sequence, executed by two threads on one processor at roughly the same time:

```
mutex_lock();
do_something(); // takes no time to execute
mutex_unlock();
exit();         // takes no time to execute
```

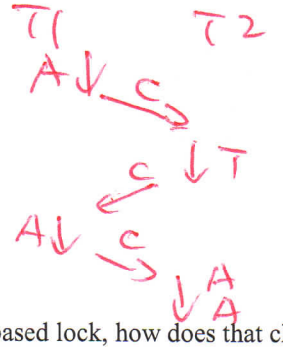
(a) What is the best-case time for the two threads on one CPU to finish this code sequence? Draw the corresponding flow.

4A+C



(b) What is the worst-case time for the two threads to finish this code sequence? Assume that only three context switches can occur at a maximum. Draw the corresponding flow.

4A+3C+T



(c) If the spin lock is instead changed to a queue-based lock, how does that change the worst-case time? Assume that only three context switches can occur at a maximum. Draw the corresponding flow.

4A+3C.

Student ID: _____

7. **Race to the Finish**: Assume we are in an environment with many threads running. Take the following C code snippet:

```
int z = 0; // global variable, shared among threads
void update (int x, int y) {
  z += x + y;
}
```

Assume that threads may all be calling update with different values for x and y.

(a) Write assembly code that implements the function update(). Assume you have three instructions at your disposal: (1) **load [address], Rdest**, (2) **add Rdest, Rsrc1, Rsrc2**, and (3) **store Rsrc, [address]**. Also, feel free to assume that when update() is called, the value of 'x' is already in R1, and the value of 'y' is in R2.

```
add r2, r1, r2
ld 'z', r3
add r2, r2, r3
store r2, 'z'
```

untimely time interrupts.

(b) Because this code is not guarded with a lock or other synchronization primitive, an "atomicity violation" could occur. Describe what this means.

Because the ld and store are not guarded by a lock, other threads could execute these two instructions in between, violating the assumption that the ld and store are atomic.

(c) Now, label places in the **assembly** code where a timer interrupt and switch to another thread could result in such an atomicity violation occurring.

(d) Now, assume we change the C code as follows:

```
void update (int x, int y) {
  z = x + y; // note we just set z equal to x+y (not additive)
}
```

If two threads call update() at "nearly" the same time, the first like this: 'update(3,4)', and second like this: 'update(10,20)', what are the possible outcomes? If we place a lock around the routine (e.g., before setting z = x + y, we acquire a lock, and after, we release it), does this change the behavior of this snippet?

7 or 30.

adding locks does not change the result.

8. **RAID Flows.** This question is about flows of I/Os in a RAID system. In RAIDs, some I/Os can happen in parallel, whereas some happen in sequence. To indicate two I/Os (to blocks 0 and 1, for example) in a flow can happen at the same time, we write: "(0 1)". To indicate two I/Os must happen in sequence (i.e., one before the other), we would use this notation: "0, 1". These flows can be built into larger chains; for example, consider the sequence "(0 1), (2 3)", which would indicate I/Os to blocks 0 and 1 could be issued in parallel, followed by I/Os to 2 and 3 in parallel.

   We can also indicate read and write operations in a flow with "r" and "w". Thus, "(r0 r1), (w2 w3)" is used to indicate we are reading blocks 0 and 1 in parallel, and, when that is finished, writing blocks 2 and 3 in parallel.

   Assume we have the following RAID-4, with a single parity disk:

```
 0     1     2     3    P0
 4     5     6     7    P1
 8     9    10    11    P2
12    13    14    15    P3
  ... (and so forth) ...
```

   (a) Assume we must **read** blocks 0, 5, 10, and 15 as fast as we can. What is the flow of these blocks?

   (r0 r5 r10 r15)

   (b) Assume we must **read** blocks 100, 109, 126, and 85 as fast as we can. What is the flow of these blocks?

   (r100 r109 r126), (r85)

   (c) Assume we must **read** blocks 0, 4, 8, and 12. What is the flow of these blocks?

   r0, r4, r8, r12

   (d) Now assume we will be writing. Assume we must **write** block 5. What is the I/O flow?

   (r5, rp1), (w5 wp1)

   (e) Assume we must **write** blocks 5 and 10. What is the I/O flow?

   (r5 rp1), (r10 rp2), (w5 wp1), (w10 wp2)

   (f) Now assume that we changed from a RAID-4 to a RAID-5 system with rotating parity. How can we arrange the data and parity blocks across the disks so as to make the writes to 5 and 10 go as fast as possible? **(draw a picture)**

   0 1 2 3 P0
   P4* 4 5* 6 7
   11 P2* 8 9 10*
   14 15 P3 12 13

Student ID: _____

9. **FFS and File System Consistency Flows:** This question is about the Berkeley Fast File System, FFS. We first investigate questions focus on the flow of data to disk, i.e., where does FFS place files?

Assume we have 10 cylinder groups (or block groups) on disk. As you might recall, each group has four basic components: an inode bitmap (Bi), a data bitmap (Bd), some inodes (I), and some data blocks (D).

This question focuses on which data structures are touched when certain operations occur. For example, if an application read a 1-block file in the second block group, the flow would be: "2I, 2D", because to read that file, the application first has to read its inode and then its data block, both of which are in the second block group. Assume for these questions that all files and directories are 1 block in size unless otherwise specified. Also assume the root directory is found in block group 1.

(a) There is an existing file "/foo" in block group 3. An application wishes to read it from disk. What is the flow of reads from disk needed to satisfy this request? (Remember to start at the root)

1i, 1d (read root dir), 3i, 3d (read foo's inode, data),
3i (update foo's last-access time)

(b) Now an empty file "/bar" is created in the same directory as foo (i.e., the root directory). What flow describes the writes to disk needed to create "/bar"? (Note: the exact ordering does not matter in this case.)

1i, 1d (update root dir), 3bi, 3i (allocate inode for bar)

Now, let's explore issues of on-disk consistency.

(c) The on-disk state of a file system can be corrupted with an untimely crash. Show **one example write ordering** (a.k.a., flow), with a crash labeled at an untimely point, which leads to an **inconsistent** on-disk state for the meta-data of the file system. For example, if you are writing blocks A, B, and C to disk (in that order), you might write down A, crash, B, C to indicate that a crash after A was written leads to an inconsistency.

write the inode for foo, which points to the data block of foo, but then crash before writing the data bitmap for foo.

(d) Now, show **one example write ordering** where the file system meta-data is perfectly consistent, but where a crash leads the file to have the wrong contents.

Write inode for foo and then crash before the data block for foo is written.

(e) Finally, **journaling** can be used to solve some of these problems. Describe how data and meta-data must be written to the journal and then to their final on-disk locations in order to avoid the two problems you demonstrated above.

Describe figure 42.1 or 42.2

10. **LFS Flows:** This question is about the log-structured file system, LFS. Remember, the file system that treats the disk kind of like a tape?

    LFS has a bunch of data structures, including some new ones that typical file systems like FFS don't have. Here is a list of some important ones:

    ```
    1. inode
    2. data block
    3. segment summary block
    4. inode map
    5. checkpoint region
    ```

    In this question, you will write down which data structures are accessed **from the disk** to complete a particular operation. Assume you start with nothing in memory; thus, you must read everything you need to read in order to complete the request.

    (a) Given the inode number of a file, which structures must you read in order to calculate the location of the inode on disk? Briefly describe the flow.

    5.4

    (b) Given the inode number of a file, which structures must you read in order to read the inode from disk? Briefly describe the flow.

    5.4.1

    (c) Given the inode number of a file, which structures must you read in order to read the first data block of the file from disk? Briefly describe the flow.

    5.4.1.2

    (d) Given the inode of a directory containing a file, which data structures must you read in order to read the first data block of the file? Briefly describe the flow.

    5.4.1(dir), 2(dir), 4.1(file), 2(file)

    (e) Given a data block on disk, which data structures must you read in order to determine whether the block is live or not? Briefly describe the flow.

    3.5.4.1

Student ID: _____

11. **NFS Flows:** This question is about flows in NFS, Sun's Network File System. Remember NFS, the block-based protocol with a focus on simple crash recovery? There are just two machines we are considering, a client and a server. Here are the possible components of a flow:

```
1. application-allocated memory
2. client memory
3. client disk
4. network
5. server memory
6. server disk
```

(a) An application (on the client) opens a file *F* which contains a data block *D*. What is the flow of *D* through the system?

Nothing

(b) An application (on the client) reads block *D* from file *F* for the first time. What is the flow of *D* through the system?

6, 5, 4, 2, 1

(c) The application reads *D* from *F* again, right after reading it the first time. What is the flow of *D*?

2, 1

(d) The application waits a *long* time, and then reads *D* from *F* again. What is the flow of *D* now? Does this read take longer than the previous re-read? (Why or why not?)

2, 1,
It could be longer if the client needs to check with ther server on the validity of its cache.

(e) The application now writes a block *D2* to an existing file *G*. What is the flow of block *D2*?

1, 2.

(f) The application, after writing *D2* to file *G*, closes the file. What is the flow of *D2*?

2, 4, 5, 6

Student ID: _____

12. **Linux Scalability to Many Cores:** The authors adapted the following flow to study the Linux scalability on many-core machines: they run application, find bottlenecks, fix bottlenecks, and repeat until a non-trivial application fix is required or the bottleneck is about shared hardware.

(a) At the end of their study, what is the conclusion about Linux's scalability on many-core machines?

Up to 48 cores, the scalability issue in the Linux kernel can all be fixed.

(b) This paper proposed sloppy counter. With per-core counters and the shared counter, what is the actual counter value?

The shared counter minus the ~~total~~ of per-core counters

(c) What is the condition that sloppy counter can be used to replace standard counter to improve scalability?

If the value can be off a little bit.

(d) Why sloppy counter is able to improve scalability?

As it reduce the frequency of shared resource (lock) accessing

(e) Other than sloppy counters, what are the other techniques they found to be useful to fix the scalability? List two of them would be enough.

Lock-free comparison, Per-core data structure.
Eliminating false sharing, Avoiding unnecessary locking.

(f) The paper we are reading is published in 2010. In 2008, the similar group of authors published a paper arguing that the OS structure should be changed to scale on many-core machines. What's your opinion on the conflicting results from these two papers?

This is an open-ended question

Student ID: _____

13. **Last Question Flows:** The last question is about four sub-questions asking your opinion! Note: this is not a flow question. Rather, it just seeks your opinion.

    (a) What is the one most important feature that an operating system absolutely should have but in your experience it does not?

Survey Questions

    (b) What is the most important lesson you leant from this class? What is the most useful part of the class?

    (c) Is this your first OS class?
    - If yes, how should the class be improved to help you master the material?
    - If no, are you bothered by the fact we spent most of our time reviewing what you may have already learnt before, and how should the class be improved in this regard?

    (d) How do you like the projects in this class? Also
    - for those asked for kernel-level projects at the beginning, how does the class projects meet your needs?
    - for those wanted user-level projects at the beginning, how does the class projects help you or fail to help you learn?

Student ID: _____