ECE 350 W25 Midterm Exam Solutions

(1.1) This works perfectly well – as long as the register data is put in a known location in a well-understood order so it can be restored later, that location could be the stack for the thread. Stack space *is* limited, but a CPU has so few registers it's very unlikely to be a problem.

(1.2) No, this does not work. The code does not consider whether the lock provided as the `rwl` argument is a read lock or a write lock. That means if a reader calls unlock after reading and the writer queue is not empty, the first writer in the writer queue will get unblocked even if there are 1+ other readers still reading.

(1.3) Security: if system services are in kernel mode and exploited, it gives the attacker full access to the system; the damage is mitigated if the services have no more privilege than other user programs.

Reliability: running these things in kernel mode means an error could lead to a whole system crash (like blue screen of death) instead of just the service breaking.

(1.4) The validation of permissions should take place in the kernel. Things that are in user-space can be manipulated by user programs and therefore an attacker could bypass the check (or fake the result).

(This is analogous to not trusting anything from the front-end in an application)

(2.1) The question does ask you to be specific about how memory is managed, but any reasonable answer there is fine. Regardless of how you chose to organize memory, you have some sort of management structure that tracks what memory is allocated and what is not.

A double free is easy to spot because there is a request to deallocate memory that either doesn't have an entry in the management data, or an entry in the management data where it shows up as free.

(2.2)

```
selected = NULL
for each free block f in memory
  if f.size >= request_size AND ( selected is null OR f.size < selected.size )
    selected = f
end for
return selected
```

(2.3)

Virtual Address	**Physical Address**	**Evaluation**
0x00001701		Page Fault
0x00004886	N/A	Segmentation Fault
0x10001C41	N/A	Segmentation Fault
0x000033BC	0x000043BC	OK

Part 2 The diagram must show that the page number is 19 bits and offset is 13 bits. Note: it's impossible to neatly represent this with hex digits; 8 KB pages does NOT mean it's 4 hex digits.

(2.4)

- On creation, N pages are allocated based on the request size. Also, the OS needs a page table for the segment. (2 marks)
- Attaching to a shared memory segment means adding the shared segment's page table to the process page table. (1 mark)
- Detaching from a shared memory segment means removing the shared segment's page table from the process page table. (1 mark)
- Deleting the segment means deallocating the various pages from the shared segment and cleaning up the OS's page table for it (1 mark). But you might also remember that it can't be cleaned up right away if the shared memory segment is attached here – so if you remember this and reference it, that's worth 1 mark too.
- Frames are used to store the pages in main memory – but the important thing here is that the page tables of all processes attached to the shared segment reference the same frame (that's how the memory is really shared). (1 mark)

(2.5) We understand that memory accesses look different for instructions and data. Many programs operate using the same instructions on different data, so we would expect to see much more temporal locality in instructions whereas data may be needed once and never again.

(3.1)

Part 1: It ignores priority entirely; it pays no attention to whether processes are CPU-bound or I/O-bound (so it may use resources inefficiently); and it doesn't work well in situations where 1 (or a minority of) user(s) make a LOT of processes.

Part 2: Something like treating all the processes of the same user as a group would improve fairness; that way a user who has 10 processes doesn't get 10 times as much CPU time as a user who has 1 process. That's a better user experience! As long as your answer addresses something in part 1, has some justification, and doesn't just resort to "use a better algorithm"; that will work.

(3.2) For scheduler (6 marks):

- Draw random ticket by calling the function with size minus one (2)
- Iterate over the `list` that many times (2)
- Remove that thread ID from the list with remove (1)
- Return its id (1)

Important observation: `draw_ticket` says it returns a number in the range $[0, max]$ which means it needs to be called with the size of the linked list minus one.

For add (4 marks):

- Create new node, assign its values 2
- Update head or tail pointer, next 2