CS 162 Operating Systems and System Programming
Fall 2024

MIDTERM 1

INSTRUCTIONS

Please do not open this exam until instructed to do so. Do not discuss exam questions for at least 24 hours after the exam ends, as some students may be taking the exam at a different time.

For questions with **circular bubbles**, you should select exactly *one* choice.

◯ You must choose either this option

◯ Or this one, but not both!

For questions with **square checkboxes**, you may select *multiple* choices.

☐ You could select this choice.

☐ You could select this one too!

GENERAL INFORMATION

This is a **closed book** exam. You are allowed 1 page of notes (both sides). You have 110 minutes to complete as much of the exam as possible. Make sure to read all of the questions first, as some of the questions are substantially more time consuming.

Write all of your answers directly on this paper. *Make your answers as concise as possible.* On programming questions, we will be looking for performance as well as correctness, so think through your answers carefully. If there is something about the questions that you believe is open to interpretation, please ask us about it!

Problem	**Possible**
0	1
1	18
2	20
3	24
4	23
5	14
Total	100

Preliminaries

This is a proctored, closed-book exam. You are allowed 1 page of notes (both sides). You may not use a calculator. You have 110 minutes to complete as much of the exam as possible. This exam is out of 100 points. Make sure to read all the questions first, as some are substantially more time-consuming.

If there is something about the questions you believe is open to interpretation, please ask us about it.

We will overlook minor syntax errors when grading coding questions. **There is a reference sheet at the end of the exam that you may find helpful.**

(a)

Name

Perilous Pintoast

(b)

Student ID

162.162.162.162

(c)

Discussion TA's Full Name

Professor Pintoaster

(d)

Please read the following honor code: "I understand this is a closed-book exam. I promise the answers I give on this exam are my own. I understand that I am allowed to use one 8.5x11, double-sided, handwritten cheat-sheets of my own making, but otherwise promise not to consult other people, physical resources (e.g. textbooks), or the internet in constructing my answers."

Write your full name below to acknowledge that you've read and agreed to this statement.

Perilous Pintoast

(e) (1.0 points) Staff Ed Username Suggestions

Sinister Sean Yang

1. (16.0 points) True/False

Please explain your answer in **TWO SENTENCES OR LESS**. Longer explanations will **GET NO CREDIT**. Answers without an explanation or that just rephrase the question will **GET NO CREDIT**.

If a statement is **TRUE**, explain why the provided statement is true, or provide an example when relevant.

If a statement is **FALSE**, explain why the provided statement is false, or provide a counterexample when relevant.

(a) (2.0 pt) Increasing the cache size can lead to a decreased cache hit rate.

● True. ○ False.

Explain.

In algorithms that don't have the stack property, Belady's anomaly can happen where cache hit rate can decrease. See Discussion 6 for an example.

(b) (2.0 pt) MLFQ prioritizes I/O bound tasks over CPU-intensive tasks.

● True. ○ False.

Explain.

True. MLFQ demotes tasks that expire their quanta on the CPU to a lower priority, which are CPU-bound tasks. It will promote tasks that do not expend their quanta, which tend to be more I/O bound.

(c) (2.0 pt) In a segmented memory model, information about the base and length of segments are stored in special registers.

○ True. ● False.

Explain.

False. They are stored in the segment descriptor table in memory. The segment selectors (taken from special registers in x86 such as the `CS` register or from a portion of the virtual address) indicate which segment the memory access corresponds to.
We were lenient with grading this question.

(d) (2.0 pt) MMU is a part of the OS kernel that translates virtual addresses into physical addresses.

○ True. ● False.

Explain.

False. The MMU is a piece of hardware that handles address translation.

(e) **(2.0 pt)** After the kernel handles a user page fault, the user application resumes execution at the instruction immediately following the instruction that caused the page fault.

○ True. ● False.

Explain.

False. Suppose a load/store instruction causes a page fault, and the page fault handler in the kernel brings in the correct page. Then, once control resumes in the user application side, it should replay the instruction so that the instruction executes successfully.

(f) **(2.0 pt)** SRTF scheduling does not suffer from the convoy effect, but SJF scheduling does.

● True. ○ False.

Explain.

True. SRTF is a preemptive scheduler that always schedules the shortest job, so it doesn't suffer from the convoy effect. SJF will not preempt a current long job that arrives before shorter jobs.

(g) **(2.0 pt)** You can implement the MIN replacement policy in a real operating system.

○ True. ● False.

Explain.

False. The MIN replacement policy requires knowledge of which task/-page will be requested furthest into the future. This is not possible in a real operating system.

(h) **(2.0 pt)** In a system with hold-and-wait, no preemption, and mutual exclusion, we can detect deadlock by checking the resource allocation graph for cycles.

○ True. ● False.

Explain.

False. Cycles are a necessary but not sufficient condition for deadlock in a resource allocation graph, assuming the other conditions hold.

(i) (2.0 pt) You can design a physically-indexed cache in a way that you can look up the TLB in parallel with cache access.

● True. ○ False.

Explain.

True. Since offset bits are the same for both virtual and physical addresses, you can have the index be in the offset. Then, you can access the cache with the index without having the physical address.

2. (20.0 points) Multiple Select

(a) (2.5 pt) Select all true statements.

- [x] A process's virtual address space can be larger than the size of the physical address space.
- [x] The physical address space can be larger than a process's virtual address space.
- [x] A process cannot access more memory than the size of its virtual address space.
- [ ] A process cannot access more memory than the size of the physical address space.
- [ ] You cannot use segmentation and paging at the same time.

(b) (2.5 pt) Select all ways that one can use to avoid/prevent deadlock.

- [x] Allocate virtually "infinite" (i.e. a very large number of) resources that can be shared among threads.
- [x] Dynamically delay resource requests that would otherwise put the system in an unsafe state.
- [x] Request resources in a fixed order across all threads.
- [ ] Create a new copy of a resource if the resource is not available.
- [x] Write code that doesn't deadlock.

(c) (2.5 pt) Select all true statements.

- [x] The clock algorithm is an approximation of LRU.
- [x] The clock algorithm implicitly partitions pages into "old" and "new" pages.
- [x] The second-chance list algorithm partitions pages into "old" and "new" pages.
- [ ] The larger n is for the n^{th} chance algorithm, it becomes more efficient time-wise to find and evict a page.
- [ ] If the clock hand moves slowly, it is an indication that the clock algorithm is performing poorly.

(d) (2.0 pt) Select all true statements.

- [x] Programs tend to alternate between bursts of CPU and I/O.
- [x] Round robin can suffer from poor cache performance.
- [x] FCFS is prone to starvation.
- [ ] As a system's utilization approaches 1, its response time approaches 1.
- [ ] None of the above.

(e) (2.5 pt) Select all of the following true statements.

- [ ] If all threads immediately block when unable to acquire a lock, priority inversion can occur when threads can only be one of two priorities.
- [x] The average completion time of SRTF is always lower or equal to that of FCFS.
- [ ] Higher stride jobs run more frequently compared to lower-stride jobs.
- [ ] Stride scheduling is prone to starvation.
- [ ] As the value of the quanta used in Round Robin increases, the Round Robin algorithm begins to resemble SRTF.

(f) **(2.0 pt)** Select all valid approaches to dealing with deadlocks.

- [x] Deadlock avoidance
- [x] Deadlock prevention
- [x] Deadlock recovery
- [x] Deadlock denial
- [ ] None of the above.

(g) **(2.0 pt)** Select all true statements about EEVDF.

- [ ] EEVDF is designed to maximize throughput of all tasks.
- [x] When a thread has negative lag, its next eligible virtual time gets postponed.
- [x] The virtual deadline of a thread is its completion time in an ideal fluid flow model.
- [ ] If a new thread with nonzero weight enters the system, you would have to recompute the virtual eligible time of another thread.
- [ ] None of the above.

(h) **(2.0 pt)** Select all true statements under the assumption that pages are all the same size.

- [x] A paging memory model may suffer from internal fragmentation.
- [ ] A paging memory model may suffer from external fragmentation.
- [x] A segmentation memory model may suffer from external fragmentation.
- [ ] A single level page table dynamically increases in size when new pages are mapped.
- [ ] None of the above.

(i) **(2.0 pt)** Select all true statements about Homeworks 2 and 3.

- [ ] `cat test.txt | head -3` shows you test.txt's file headers (e.g. creation date, size, permissions).
- [x] Typing `wc` in your shell searches for the executable `wc` in a directory specified the `PATH` environment variable.
- [x] You can make HTTP requests with `curl`.
- [x] When you make an HTTP GET request for a directory, the server may respond with an `index.html` file.
- [ ] None of the above.

3. (24.0 points) Short Answer

(a) **(3.0 pt)** Explain in detail how you can implement copy-on-write. Be explicit on how you would change page table entries and appropriate handlers in the kernel.

1. Mark all pages as read-only.
2. Copy the page table to the newly created process.
3. Once either of them attempts to write to a page, it will trigger a page fault.
4. In the page fault handler, make a physical copy of the page. Install this new page in the page table of the process that triggered the page fault.

This way, we do not have to make a physical copy of the entire virtual address space upon `fork`. Instead, we take advantage of demand paging to copy pages out only when necessary, which is particularly useful when we `fork` a new process and immediately `exec`.

(b) **(8.5 pt)** Consider a system with a **preemptive priority-based scheduler** that implements priority donation. Suppose a set of threads enter the system and execute in this order. Assume no other threads enter the system after Step 5.

1. Thread A is created with priority 1 successfully acquires Lock 1.
2. Thread B is created with priority 50, successfully acquires Lock 2, and calls acquire on Lock 1.
3. Thread C is created with priority 100, and calls acquire on Lock 2.
4. Thread D is created with priority 80, and calls acquire on Lock 1.
5. Thread A runs and releases Lock 1, but does *not* terminate.

i. **(2.5 pt)** What is the effective priority of Thread A after each of these 5 steps?

1, 50, 100, 100, 1

ii. **(2.5 pt)** What is the effective priority of Thread B after each step, for steps (2) through (5)?

50, 100, 100, 100

iii. **(3.5 pt)** What is the order of execution of Threads A, B, C, and D after Step (5) until each thread runs to completion?

Assume that when Thread B is rescheduled to run, it releases Lock 2, releases Lock 1, then terminates, in that order. *Example Answer: <B, C, D, A, B>*

B, C, B, D, B, A

(c) **(4.5 pt)** Suppose you investigated a multi-processed program and find out it swaps pages in and out extremely frequently. This is strange, since this was written to be a heavily CPU-bound program.

i. **(1.5 pt)** What is the most likely phenomenon that is happening? Answer in five words or fewer.

Thrashing

ii. **(1.5 pt)** Given the scenario above, provide an inequality in terms of the following terms:

- n, the number of processes
- RSS, the resident set size of each process (assume it is identical for every process)
- PHYS_MEM, the size of physical memory

$n * \text{RSS} > \text{PHYS\_MEM}$

iii. **(1.5 pt)** You ended up simply serializing the processes and found out it led to no swap activity within a single process. Give an inequality between the working set size (WSS) and resident set size (RSS) of each process and PHYS_MEM. (e.g. PHYS_MEM $\leq$ RSS $\leq$ WSS).

$\text{WSS} \leq \text{RSS} \leq \text{PHYS\_MEM}$

(d) **(3.0 pt)** Below shows some code that attempts to free the elements of a Pintos list that implements an FDT. Assume this correctly compiles, and `&t->pcb->fdt` is a valid pointer to a correctly formed Pintos list. Explain the issue with the code below, and propose a solution. Be detailed, mentioning how a Pintos list works. Vague or trivial responses (i.e. buggy `free`, kernel stack overflow, etc.) will not be given credit.

```
struct fdt_elem* fd;
struct list_elem* e;
struct thread* t = thread_current();
for (e = list_begin(&t->pcb->fdt); e != list_end(&t->pcb->fdt); e = list_next(e)) {
    fd = list_entry(e, struct fdt_elem, elem);
    free(fd);
}
```

Once the enclosing struct (`struct fdt_elem`) is freed, the Pintos `list_elem` struct inside is also freed, which contains pointers to the next and previous elements. Therefore, iterating to the next element will fail. The simplest fix is to move `e = list_next(e)` in the for-loop to the line following `list_entry` and preceding `free`.

(e) **(3.0 pt)** Suppose that we have the following resources: A, B, C and threads T1, T2, T3. The total number of each resource is:

	A	B	C
Total	8	10	5
Available	1	5	1

Further, assume that the processes have the following maximum requirements and current allocations:

Thread ID	Current			Maximum			Needed		
	A	B	C	A	B	C	A	B	C
T1	2	1	1	4	9	4	2	8	3
T2	1	2	2	6	3	3	5	1	1
T3	4	2	1	5	2	2	1	0	1

Note: the tables above are provided for your convenience. You do not need to fill them out.
Is the system in a SAFE state? If so, give a potential order of thread completions that makes it SAFE.

Yes this system is in a safe state. First, T3 can run -> Available <A, B, C> = <0, 5, 0> Then T3 donates its resources back to the available pool -> <5, 7, 2> Now we have enough resources to run T2. When T2 finishes, Available <A, B, C> = <5+1, 7+2, 2+2> = <6, 9, 4> T1 now can satisfy all requests, so T1 runs.
Ordering: T3, T2, T1.

(f) **(2.0 pt)** Given the following information: A single-level page table scheme with 20 ns to access main memory and 3 ns to look up an entry in the TLB.

What is the required hit rate for the TLB to achieve AMAT that is 10% of the memory access time for a page fault without a TLB?

A page fault without a TLB requires 2 memory accesses which takes 40 ns. To achieve 10% of that, we need to have an AMAT of 4 ns. As the access time for main memory is 20 ns, no hit rate will result in an AMAT of 4 ns, so this question is impossible.

This question was NOT graded.

4. (23.0 points) Take a Page out of My Book

You may leave your answers to the following questions as an arithmetic expression involving terms that are powers of 2.

We have a system where virtual addresses are 48 bits wide, and we have 16 KiB sized pages. We also have 64 GiB of physical memory available. Each PTE (page table entry) is 4 Bytes. Assume that we're working with little-endian, byte-addressable memory.

Here's the schema of the virtual address, which is 48 bits:

UNUSED (X bits)	**VPN 1** ([C] bits)	**VPN 2** ([C] bits)	**Offset** ([A] bits)

And of the page table entry, which is 32 bits (4 Bytes):

PPN ([D] bits)	**UNUSED** (Y bits)	**Dirty** (1 bit)	**Kernel-only** (1 bit)	**Read-only** (1 bit)	**Present** (1 bit)	**Valid** (1 bit)

(a) **(2.0 pt)** How many bits are in the offset?

16 KiB $\Rightarrow 2^{14} \Rightarrow$ 14 bits in offset

(b) **(2.0 pt)** If we wanted a single page table to fit inside of a page, how many PTEs would be in a page table?

2^{14} bytes / 4 bytes per PTE $= 2^{12}$ PTEs

(c) **(1.0 pt)** How many bits should there be for each VPN? **12**

(d) **(2.0 pt)** How many bits should there be for the PPN?

64 GiB $\Rightarrow 2^{36}$ bit physical address space. 2^{36}/offset bits $= 2^{36}/2^{14} = 2^{22} \Rightarrow$ 22.

(e) **(1.5 pt)** Describe one potential use case for the bits left unused in the PTE.
Vague responses or explaining an existing mechanism in the schema will be given no credit.

We can have a "reference" bit to see if the page was recently referenced. This can be used for the clock algorithm for page eviction policies.

(f) **(1.5 pt)** Provide one potential usage case for the bits left unused in the virtual address.
Vague responses or explaining an existing mechanism in the schema will be given no credit.

We can store a process ID in the high bits. This allows us to avoid flushing the TLB or virtually-tagged caches upon a context switch.

(g) **(14.0 pt)** Note: this subpart is independent of the previous subparts.

Assume we have a non-"magic" memory scheme with 16 MiB of virtual memory, 1 MiB of physical memory and a page size of 256B.

The virtual address is 24 bits wide.

VPN 1 (8 bits)	**VPN 2** (8 bits)	**Offset** (8 bits)

This translates to a physical address which is 20 bits wide.

PPN (12 bits)	**Offset** (8 bits)

Our page table entries are 16 bits wide.

PPN (12 bits)	**Metadata** (4 bits)

And the metadata is formatted as follows

Kernel Only 1 bit	**Use** 1 bit	**Writable** 1 bit	**Valid** 1 bit

Fill out the following table of memory **addresses** by performing a page table walk with the physical memory given in the following page. Assume you are running in user mode on a little endian machine with the PTBR containing the value 0xC0000. Each access either reads or writes one byte.

If any operation is unsuccessful while doing the walk, fill in the rest of the row with N/A and write the failure in the Result column. If we notice an access violation, we do **NOT** translate the address. Possible access failures include **"KERNEL_ONLY"**, **"NOT_WRITABLE"**, and **"INVALID"**, and any addresses not in present in the table have failure code **"UNKNOWN"**. Be careful, since it is possible that two PTEs point to the same page with different metadata. Otherwise, write the byte loaded or **"OK"** if the store was successful.

Virtual Address	Operation	PTE 1 Address	PTE 2 Address	Physical Address	Result
0x731355	Load	0xC00E6	0x20026	0xA0055	0xDB
0x52203F	Store	**0xC00A4**	**0x60040**	**0xF003F**	**OK**
0x52127F	Store	**0xC00A4**	**0x60024**	**A007F**	**OK**
0x760063	Load	**0xC00EC**	**N/A**	**N/A**	**INVALID**
0x603208	Load	**0xC00C0**	**0x20064**	**0xF0008**	**0x19**
0x62226E	Load	**0xC00C4**	**0x20044**	**F006E**	**0xAB**
0x012001	Store	**0xC0002**	**0x20040**	**N/A**	**KERNEL_ONLY**
0x472317	Store	**0xC008E**	**0x60046**	**N/A**	**NOT_WRITABLE**

Addr	**+F**	**+E**	**+D**	**+C**	**+B**	**+A**	**+9**	**+8**	**+7**	**+6**	**+5**	**+4**	**+3**	**+2**	**+1**	**+0**
0x20000	A6	2C	94	4A	77	62	CF	7A	C6	E3	83	B7	AD	1F	A0	03
0x20010	C5	BB	2F	09	65	06	0C	06	48	5C	70	F0	00	03	ED	69
0x20020	E0	3D	26	D6	38	0A	95	04	A0	01	F0	01	A0	01	A0	03
0x20030	EF	AC	66	40	0E	F3	96	20	37	78	57	47	A0	03	A8	65
0x20040	64	14	39	E1	D7	91	47	48	A0	03	F0	01	42	92	A0	0F
0x20050	07	78	7C	27	EF	67	13	33	E2	86	D0	3F	5F	37	96	5C
0x20060	92	B5	5D	54	FC	CC	22	0A	93	74	F0	03	4A	6E	F4	18
0x20070	17	08	74	7D	FA	48	82	D3	88	22	1E	F6	04	F2	7E	7E
...																
0x60000	8A	71	6B	6C	DC	E3	54	3C	48	5E	CD	8B	2C	34	F3	9B
0x60010	2E	4D	F9	88	9E	C4	5D	7C	3B	C2	0F	F0	07	00	CE	FF
0x60020	B9	EF	30	07	A6	2E	19	88	F0	02	A0	03	A0	04	85	94
0x60030	F5	C9	F6	6B	B2	80	D0	7F	78	D9	1F	CA	A0	08	10	CC
0x60040	C1	28	09	80	56	3B	08	53	F0	05	A0	01	CE	07	F0	03
0x60050	B7	79	7C	68	3E	A3	8D	CC	DD	89	93	B8	42	A8	B7	DA
0x60060	E1	DD	FB	64	7F	68	C7	D7	B4	74	A0	03	8C	D7	60	AA
0x60070	87	3D	4E	D7	1A	EF	F9	EA	B0	43	5D	48	21	83	01	1A
...																
0xA0000	BE	A7	CF	47	E2	16	C0	B1	4B	5D	9B	48	60	91	05	38
0xA0010	5C	1F	AB	E6	20	AD	64	AA	E9	C5	4F	2B	ED	97	6A	58
0xA0020	91	35	18	1D	C0	9C	76	BE	9F	D2	9A	EC	F9	D2	D6	65
0xA0030	DF	DE	2E	9C	51	A0	FB	ED	F6	41	62	CB	C6	93	00	9E
0xA0040	C3	EC	31	3C	B1	8C	80	D2	10	8F	65	70	39	C7	09	DA
0xA0050	A8	E5	DB	62	6F	E3	7B	CB	AD	AF	DB	60	A0	1A	62	3A
0xA0060	A9	A9	1D	9D	21	78	69	20	6B	26	06	BD	B3	A4	61	47
0xA0070	4C	5D	24	FF	D3	0E	4D	F7	20	01	57	62	8B	30	92	A4
...																
0xC0000	2F	50	9D	F0	E8	20	A1	96	9E	B4	F4	E1	20	03	00	60
0xC0010	47	42	14	05	46	45	C1	52	93	EC	C3	6F	D7	30	CC	DE
0xC0020	6D	69	5B	4F	83	DC	33	91	FF	91	C9	A1	19	CC	A0	01
0xC0030	BC	D5	B0	15	E9	B5	4A	24	01	28	7E	C1	5C	8A	CC	48
0xC0040	2F	2B	F5	A5	A6	A6	12	20	04	C9	93	DD	A5	DB	53	6D
0xC0050	1A	A4	D6	41	A6	DD	FF	EE	6E	93	53	18	20	07	D0	D6
0xC0060	FA	F3	25	C5	83	90	E5	41	2C	76	08	A0	60	01	60	08
0xC0070	F9	B3	66	46	ED	82	DD	E3	3C	13	2A	60	0E	33	26	BC
0xC0080	60	03	F2	03	0E	48	1B	D5	95	13	B0	B8	2A	91	2A	AA
0xC0090	69	2C	E4	A9	42	21	B3	F6	C0	C8	96	C4	65	FD	CE	84
0xC00A0	82	BA	BA	E6	65	5B	F6	97	16	06	60	03	A6	DC	FD	60
0xC00B0	C0	05	DA	F6	66	8D	4F	26	BA	74	67	D7	30	DB	AC	14
0xC00C0	21	EE	CD	59	1A	DE	4E	4D	E4	DB	20	01	51	30	20	03
0xC00D0	E8	48	2B	8E	1B	E6	FE	6D	31	48	ED	92	4A	A2	64	12
0xC00E0	ED	86	20	02	5D	F3	99	90	20	07	D7	D4	89	46	0C	D2
0xC00F0	71	EE	86	DB	7C	0A	DE	46	5E	90	92	CF	7E	20	18	EB
...																
0xF0000	A0	B6	91	71	A1	E9	58	19	53	A8	0C	42	DB	0A	BC	21
0xF0010	81	D5	B9	F9	5D	30	5C	90	1A	3F	B4	FF	7B	E6	94	94
0xF0020	EA	D1	F3	97	1F	D8	F6	FC	CB	61	38	DB	D9	FF	8A	FB
0xF0030	55	18	F2	52	7B	CE	CA	B1	F5	79	95	51	83	4B	1B	07
0xF0040	3A	84	37	C0	31	51	76	0C	FC	FC	BF	53	C3	2E	64	E8
0xF0050	87	3F	5C	EC	C5	F9	A9	5A	0A	FA	BF	26	6C	CC	53	5A
0xF0060	B2	AB	1F	1D	06	D7	51	D6	1B	1B	C5	71	2D	1D	E9	53
0xF0070	8D	BB	D9	16	66	89	21	BA	64	CD	6D	B0	B6	ED	ED	0C

5. (14.0 points) EEVDF

Suppose we are implementing EEVDF (Earliest Eligible Virtual Deadline First) as our scheduling algorithm.

Our system has the following constraints:

(a) We represent each request length in real physical time, which is measured in terms of *ticks*.

(b) We allocate CPU time for each thread in increments of a fixed time quanta q=1 tick.

(c) Break ties for scheduling in favor of the highest numbered thread.

(d) Thread 1 becomes active at $t = 0$, and has a request length $r_1 = 3$, with weight $w_1 = 1$.

(e) Thread 2 becomes active at $t = 1$, and has a request length $r_2 = 1$, with weight w_2=1.

(f) Thread 3 becomes active at $t = 2$, and has a request length $r_3 = 3$, with weight $w_3 = 3$.

Assume each thread takes up exactly its requested service time for each request, and leaves the system after completing 3 requests of its specified request length.

Write virtual eligible time and virtual deadline for each thread, as well as which thread runs at the given physical and virtual time. **T{i}** represents **thread i** in the below table. The **scheduled thread** column refers to the currently running thread at the corresponding time.

(a) **(1.0 pt)** How does the rate of virtual time ($\frac{dV}{dt}$) change at $t = 0$, $t = 1$, and $t = 2$? Show your calculations.

From $t = 0$ to $t = 1$, $\frac{dV}{dt} = \frac{1}{w_1} = 1$.
From $t = 1$ to $t = 2$, $\frac{dV}{dt} = \frac{1}{w_1+w_2} = 0.5$.
From $t = 2$ to $t = 3$, $\frac{dV}{dt} = \frac{1}{w_1+w_2+w_3} = 0.2$.

(b) **(13.0 pt)** Fill in this table based on the above specifications.

Real time (ticks)	**Virtual time**	**T1**		**T2**		**T3**		**Scheduled thread**
t	**V(t)**	v_e	v_d	v_e	v_d	v_e	v_d	–
0	0	0	3	–	–	–	–	T1
1	1	0	3	1	2	–	–	**T2**
2	**1.5**	**0**	**3**	**2**	**3**	**1.5**	**2.5**	**T3**
3	**1.7**	**0**	**3**	**2**	**3**	**1.5**	**2.5**	**T3**
4	**1.9**	**0**	**3**	**2**	**3**	**1.5**	**2.5**	**T3**
5	**2.1**	**0**	**3**	**2**	**3**	**2.5**	**3.5**	**T2**
6	**2.3**	**0**	**3**	**3**	**4**	**2.5**	**3.5**	**T1**