CS 162
Fall 2021

Operating Systems and Systems Programming

Midterm 2

INSTRUCTIONS

This is your exam. Complete it either at exam.cs61a.org or, if that doesn't work, by emailing course staff with your solutions before the exam deadline.

This exam is intended for the student with email address `<EMAILADDRESS>`. If this is not your email address, notify course staff immediately, as each exam is different. Do not distribute this exam PDF even after the exam ends, as some students may be taking the exam in a different time zone.

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!

You may start your exam now. Your exam is due at <DEADLINE> Pacific Time. Go to the next page to begin.

Preliminaries

(a) Full name

(b) Student ID number

(c) Autograder login (e.g. `student123`)

1. (27.0 points) True or False

Please pick whether each statement is true or false and explain why. Explanations must be two sentences or less. You may not use semicolons to circumvent this limit. Any answer longer than two sentences will receive a 0.

(a) (3.0 points)

i. An unsafe state means that a deadlock is guaranteed to happen.

○ True

[✓ False]

ii. Explain.

An unsafe state means that there is an order of requests that leads to deadlock, but ther can be orders of requests that don't lead to deadlock.

(b) (3.0 points)

i. Processes may use the same virtual address but will actually access different physical addresses.

[✓ True]

○ False

ii. Explain.

Controlled overlap is the idea that the OS provides an abstraction where each process has the entire computer to itself, but in reality each process only has a portion of physical memory.

(c) (3.0 points)

i. Compulsory cache misses can be fixed by increasing the cache size or associativity.

○ True

[✓ False]

ii. Explain.

Compulsory misses are unavoidable because it's the first access to a page. It can only be solved by pre-fetching.

(d) (3.0 points)

i. If the goal of a preemptive scheduler is to minimize the response time, it should use SJF.

○ True

[✓ False]

ii. Explain.

SJF is optimal for minimizing average response times among non-preemptive schedulers. SRTF should be used for preemptive schedulers.

(e) (3.0 points)

i. The main reason why performance degrades when a process thrashes is due to heavy context-switching.

○ True

[✓ False]

ii. Explain.

The main reason performance degrades when a process trashes is due to frequent page faults, as the process' working set does not fit in the memory.

(f) (3.0 points)

i. In a one-level paging scheme, a single page table and a single TLB are shared by all processes.

True [✓ False]

ii. Explain.

Each process has its own page table. The TLB, however, is still shared across all processes.

(g) (3.0 points)

i. SRTF scheduler can lead to starvation.

[✓ True]

○ False

ii. Explain.

SRTF can starve if many small jobs run, preventing large jobs from running.

(h) (3.0 points)

i. When a system approaches a steady-state 100% utilization, its queue length will approach infinity.

[✓ True]

○ False

ii. Explain.

By Little's Law, the length of the queue is $\lambda \times T_q$ where $T_q \propto \frac{u}{u-1}$. Therefore, as utilization approaches 100%, the queue length approaches infinity.

(i) (3.0 points)

i. When using demand paging, adding more memory to a system will always reduce the number of page faults.

○ True

[✓ False]

ii. Explain.

Bélády's anomaly states that adding more memory can increase page faults for certain page replacement algorithms (e.g. FIFO).

2. (18.0 points) Select All

For the following questions, select all that apply. If none should be selected, don't select any. Selecting an incorrect choice or not selecting a correct choice will lead to point deductions, but the overall score for each question will be a minimum of 0.

(a) (3.0 pt) Which of the following are true about I/O?

☐ Direct memory access (DMA) requires the processor to transfer the data between devices and memory.

■ A PCI bus (not PCI Express bus) can handle a single request at a time.

☐ Polling is always more efficient than interrupts.

■ A blocking I/O operation leads to the process being suspended until the operation completes.

☐ An asynchronous read operation is blocking.

i. DMA programs the device controller to transfer directly to memory via the bus.

ii. PCI Express can handle multiple requests at at time.

iii. Periodically polling for a sparse input is inefficient.

iv. This is the definition of blocking I/O.

v. An asynchronous read operation provides pointer to user's buffer and returns *immediately*. Later, the kernel fills buffer and notifies user.

(b) (3.0 pt) Which of the following are true about I/O devices?

■ Port mapped I/O interacts with a device via special privileged in/out instructions.

☐ A memory-mapped display controller allows the process to change the image on the screen by using memory store operations.

☐ Seek time is the time it takes for the desired sector to move under the disk head.

■ Magnetic disks have much faster sequential access than random access.

■ Copy-on-write enable SSDs to reduce the write overhead.

i. Port mapped I/O uses in/out instructions to read/write to ports.

ii. With a memory-mapped display controller a process can write directly to the display buffer (also called frame buffer).

iii. This is the rotation time. Seek time is the time it takes to locate the correct track.

iv. Random access is much slower as we need to move the disk head around.

v. Copy-on-write avoids the overhead of the erasure operation which is much more expensive than a write operation, by writing the updates to an empty page.

(c) **(3.0 pt)** Which of the following are true about deadlock?

- ■ It is possible to have a deadlock when there's only one process in the system, and that process is single threaded.
- ☐ A thread in a critical section cannot be preempted by the OS.
- ■ Banker's algorithm ensures some deadlock-free execution order exists at all times
- ☐ A cycle in a resource allocation graph always implies deadlock.
- ☐ Mutual exclusion, hold and wait, no preemption, and circular waiting guarantee that a deadlock will happen.

i. Acquiring a non-reentrant lock twice in a row before releasing will result in deadlock within the same thread.

ii. A thread in a critical section will be context switched when its time quanta expires.

iii. Banker's algorithm prevents deadlock.

iv. Reference lecture 12 slide 30 for an example.

v. All four conditions can be satisfied with multiple instances of a type of a resource.

(d) **(3.0 pt)** Which of the following are true about demand paging?

- ☐ Belady's anomaly states that with certain access patterns, LRU can have a worse hit rate than FIFO.
- ☐ The working set of a process is the set of memory locations that it accesses throughout its lifetime.
- ■ Demand paging does not require a TLB.
- ■ On a page fault, the physical memory traps to the OS.
- ☐ Under the working set model, cache hit rate grows linearly with respect to the cache size.

i. Belady's anomaly is about the effect of increasing cache capacity on the hit rate.

ii. Working set is the subset of the address space the process is currently working through.

iii. TLB is just a cache for address translations.

iv. The memory management unit (MMU) traps to the OS.

v. There are spikes every time a new working set fits.

(e) (3.0 pt) Which of the following are true about synchronization and deadlock?

- [✓] A deadlock implies starvation but the converse is not true.
- ☐ Most modern operating systems use Banker's algorithm to avoid deadlocks.
- [✓] A deadlock can occur with resources such as memory or sockets.
- ☐ A thread can be blocked on more than one condition variable.
- ☐ A preemptive CPU scheduler would eliminate the possibility of deadlock in the system.

i. Deadlock is a form of starvation with a stronger condition where threads fail to make progress due to cyclical waiting.

ii. In most modern operating systems, the number of processes is not fixed and total amount of resources used for each process is unknown, making implementing Banker's algorithm impossible.

iii. Deadlock is a circular waiting of any type of resource not limited to locks.

iv. A thread blocks when waiting on a condition variable.

v. A preemptive CPU scheduler would not release locks from threads that hold them since it only preempts the CPU.

(f) (3.0 pt) Which of the following are true about schedulers?

- [✓] FCFS has throughput at least as good as RR.
- ☐ Gang scheduling is a method to avoid deadlocks by scheduling all threads of a process at once.
- [✓] Lottery scheduling will not result in starvation in expectation.
- ☐ EDF schedules the task with the shortest completion time.
- [✓] RR is the fairest scheduler with regards to wait time for CPU.

i. At the very least, RR incurs the same number of context switches as FCFS. In reality, RR will incur many more context switches, resulting in context switching overhead and suboptimal cache performance.

ii. Gang scheduling minimizes the overhead of spinlocks.

iii. Each job is given at least one ticket, so it should not starve on expectation.

iv. EDF schedules tasks with the earliest deadline per the name.

v. Fairness is well defined here, and each task waits the same amount of time for RR.

3. **(32.0 points) Short Answer**

Answer each question in three sentences or less unless otherwise specified. You may not use semicolons to circumvent this limit. Any answer longer than three sentences will receive a 0.

(a) **(4.0 pt)** Consider a SSD without a flash translation layer (FTL) having 4 KiB pages, 512 KiB erasure blocks, 3 ms erasure times, and 50 μs read and write times. Assuming no queueing and controller times, how long in ms does it take to write to an existing page? Explain.

15.8 ms. Every time a page is written to, the block containing that page needs to be read in, erased, and written back. This gives us number of pages in erasure block $\times$ (read time per page + write time per page) + erasure time $= (512$ KiB/erasure block$\div 4$ KiB/page$) \times (50$ s$+50$ s$)+3$ ms $= 15.8$ ms

(b) **(4.0 pt)** What is thrashing? How do you prevent it?

Thrashing occurs when a cache is too small to hold the working set resulting in constant page faults and cache misses. Thrashing can be prevented by increasing the cache size/associativity, optimizing memory accesses for better spatial/temporal locality, or using a better replacement algorithm.

(c) **(4.0 pt)** In terms of speed or memory efficiency, what is one advantage and one disadvantage of using a single level page table compared to a multilevel page table?

Single level page tables require less memory lookups, resulting in faster translation times. However, a sparse virtual address space can result in wasted memory space when using single level paging due to internal fragmentation.

(d) **(4.0 pt)** In a system with $N > 1$ jobs of equal length T that arrive at the same time, will FCFS always have a lower average response time than RR with quantum $< T$? Assume there is no I/O operation for any of the jobs. Explain.

Yes. RR will finish all jobs in the last N time slices since the job lengths are greater than the quantum.

(e) **(4.0 pt)** Does a direct-mapped cache always have a lower hit rate than a fully associative cache for the same access pattern? Explain.

No. A bad page replacement algorithm for fully associative cache may result in a worse hit rate than a direct-mapped cache. Moreover, if there are no conflict misses, the direct-mapped cache will work just as well as fully associative.

(f) **(4.0 pt)** Explain priority inversion and how to correct a priority scheduler to avoid priority inversion.

Priority inversion occurs when a high priority thread blocks on a resource held by a low priority thread. Priority inversion can be avoided with priority donation from high priority thread to low priority thread.

(g) **(4.0 pt)** Under what scenario is LRU a poor approximation for MIN?

LRU assumes an entry that has not been used recently will not be used in the future. Access patterns that lack temporal locality will result in poor performance.

(h) **(4.0 pt)** Can a user program unfairly manipulate priority to take advantage of the MLFQ scheduler? Explain.

A program can add meaningless I/O (e.g. `printf`) to unfairly increase the priority. By "yielding" early, the user program will remain at high priority queue levels.

4. (18.0 points) Design Doc Cram

Jieun just realized that her Project 2 design doc is due in 1 hour! She has yet to start on it as she's been busy releasing her new album. She is panicking, but you've reassured you can help her.

Fill in the blank for each question with a word or phrase that is the most appropriate based on your design review. Each blank should consist of no more than 10 words. Any answers longer than 10 words will receive a 0

(a) (3.0 pt) Any method that unblocks thread(s) needs to check if the unblocked thread(s) ___________ for strict priority scheduler.

have greater priority than the current thread

(b) (3.0 pt) To accomplish strict priority scheduling, all scheduling decisions should be made using the ___________ of each thread.

effective priority

(c) (3.0 pt) The data structure for user level locks and semaphores is similar to the ___________ from Project 1.

file descriptor table

(d) (3.0 pt) With multiple process-level synchronization primitives, deadlock should be avoided by ___________.

defining a global acquisition order

(e) (3.0 pt) When the main thread calls `pthread_exit`, it must ___________ threads.

join on all unjoined

(f) (3.0 pt) `process_wait` and `pthread_join` need to ___________ before blocking on the semaphore.

drop all locks

5. (36.0 points) Replacement O'Clock

Inspired by the scheduling homework, Michael and Nathan decide to implement the clock algorithm. They've started to write the skeleton, but they need your help to finish the rest of it!

You are given the following data structures and methods.

```
struct clock {
  struct list entries;        /* Pintos list of blocks in this clock */
  struct entry* curr_entry;   /* Current entry the clock hand is pointing to. */
};

struct entry {
  int page;                   /* Page this entry holds, -1 if empty or invalid. */
  int use;
  struct list_elem elem;
};

/* Evicts CLOCK->CURR_ENTRY. If entry is empty or invalid, does nothing. */
void evict_entry(struct clock* clock);

/* Brings in PAGE but does not update CLOCK->CURR_ENTRY. */
void admit_page(struct clock* clock, int page);

/* Advances CLOCK->CURR_ENTRY to next entry, wrapping around to the first entry if necessary.*/
void advance_clockhand(struct clock* clock);

/* Runs an iteration of the clock algorithm.
   Return 1 on a cache hit and 0 on a miss. */
int clock_iteration(struct clock* clock, int target_page) {
  struct list_elem* e;
  for (__________[A]__________) {
    struct entry* entry = __________[B]__________;
    if (__________[C]__________) {
      __________[D]__________;
      __________[E]__________;
    }
  }

  while (1) {
    __________[F]__________;
    if (__________[G]__________) {
      __________[H]__________;
    } else {
      if (__________[I]__________) {
        __________[J]__________;
      }
      __________[K]__________;
      __________[L]__________;
      __________[L]__________;
      return 0;
    }
  }

  return -1;
}
```

Your job is to implement `clock_iteration`. Assume `clock` and each entry in it are already initialized. `clock`'s capacity must stay fixed (i.e. not allowed to change the number of entries).

You are only allowed to write one line of code per line (fill in **two** lines for [L]). You may not use semicolons to write multiple lines (except for filling in the blank of a `for` loop). You are only allowed to use methods and data structures provided to you in this problem and the reference sheet.

(a) (2.0 pt) [A]

```
e = list_begin(&clock->entries); e != list_end(&clock->entries); e =
list_next(e)
```

(b) (2.0 pt) [B]

```
list_entry(e, struct entry, elem)
```

(c) (3.0 pt) [C]

```
entry->page == target_page
```

(d) (3.0 pt) [D]

```
entry->use = 1
```

(e) (2.0 pt) [E]

```
return 1
```

(f) (3.0 pt) [F]

```
advance_clockhand(clock)
```

(g) (3.0 pt) [G]

```
clock->curr_entry->page != -1 && clock->curr_entry->use == 1
```

(h) (3.0 pt) [H]

```
clock->curr_entry->use = 0
```

(i) (3.0 pt) [I]

```
clock->curr_entry->page != -1
```

(j) (3.0 pt) [J]

```
evict_entry(clock)
```

(k) (3.0 pt) [K]

```
admit_page(clock, target_page)
```

(l) (6.0 pt) [L]

```
clock->curr_entry->page = target_page
clock->curr_entry->use = 1
```

6. (28.0 points) MarcOS

Marcus is building a virtual memory scheme for his new operating system MarcOS. He designs a virtual memory paging scheme using single level page tables with 24-bit virtual addresses, 32-bit physical addresses, 64 KiB page size, and a page table entry of size 4 bytes. Assume each page table fits in a single page and memory addresses are byte-addressed.

(a) (6.0 points)

For the following questions, write a single decimal integer. Any answer not matching this format (e.g. using mathematical expressions) will receive a 0.

i. (2.0 pt) What is the least number of entries necessary per page table?

256 entries. The page size is 64 KiB, meaning $\log_2 64 = \log_2(2^8 \cdot 2^{10}) = 18$ **bits of the virtual address are used for the offset. This leaves 8 bits for the virtual page number, meaning there are** $2^8 = 256$ **virtual pages. Page tables are indexed by virtual page numbers, so the number of page table entries is the number of virtual pages.**

ii. (2.0 pt) How many levels of page tables would be required to map the entire virtual address space to the physical addresses?

1 level. From (i), there are 2^8 **virtual pages. Since each page table entry is 4 bytes, the page table would be of size** 2^{10} **bytes or 1 KiB, which fits in one page of size 64 KiB.**

The original intent of this question was that the question did not give the assumption of a single level paging scheme.

iii. (2.0 pt) How many bits of metadata can you fit in each page table entry?

16 bits. From (i), we calculated there were 16 bits for the offset. The offset bits are the same in virtual and physical addresses, so this leaves 16 bits for the physical page number. Page table entries are composed of the physical page number and metadata bits. Since each page table entry is 4 bytes = 32 bits, this leaves 16 bits for the metadata bits.

(b) (10.0 points)

Suppose the first six page table entries are as follows: `0x123442C3`, `0x4320E8C4`, `0xAD007DA2`, `0x20200D6A`, `0x7A511594`,`0x6000CCE3`. Assume all six are valid and the rest of the page table entries are invalid.

For the following questions, write the *virtual address that maps to the given physical address* as a hexadecimal integer with the `0x` prefix (`x` lowercase) without any leading zeros (e.g. `0x1` not `0x01`). Any hex letters must be written in capital letters (e.g. `0xC`). If no such virtual address exists, write NONE in capital letters. Any answers not matching this format will receive a 0.

i. (2.0 pt) `0x20001234`

NONE. From (a.i), we calculated there were 16 bits bits for the offset, which is 4 hex digits. The hex digits left of the right four are the physical page number in both the physical address and page table entry. None of the entries contains a physical page number of `0x2000`.

ii. (2.0 pt) `0x20207A51`

`0x37A51`**. From (a.i), we calculated there were 16 bits bits for the offset, which is 4 hex digits. The hex digits left of the right four are the physical page number in both the physical address and page table entry. The physical page number `0x2020` is mapped by the virtual page number `3`, giving a virtual address of `0x37A51` by concatenating the virtual physical page number and offset.**

iii. (2.0 pt) `0x1234AD00`

`0xAD00`**. From (a.i), we calculated there were 16 bits bits for the offset, which is 4 hex digits. The hex digits left of the right four are the physical page number in both the physical address and page table entry. The physical page number `0x1234` is mapped by the virtual page number `0`, giving a virtual address of `0xAD00` by concatenating the virtual physical page number and offset.**

iv. (2.0 pt) `0x2020AD00`

`0x3AD00`**. From (a.i), we calculated there were 16 bits bits for the offset, which is 4 hex digits. The hex digits left of the right four are the physical page number in both the physical address and page table entry. The physical page number `0x2020` is mapped by the virtual page number `3`, giving a virtual address of `0x3AD00` by concatenating the virtual physical page number and offset.**

v. (2.0 pt) `0xAD002000`

`0x22000`**. From (a.i), we calculated there were 16 bits bits for the offset, which is 4 hex digits. The hex digits left of the right four are the physical page number in both the physical address and page table entry. The physical page number `0xAD00` is mapped by the virtual page number `2`, giving a virtual address of `0x22000` by concatenating the virtual physical page number and offset.**

(c) (12.0 points)

For the following questions, give an explanation in three sentences or less. Any answer longer than three sentences will receive a 0.

i. (4.0 pt) Marcus wants to implement demand paging with the Nth chance page replacement policy, using the metadata bits in the page table entry to store any necessary information. How many metadata bits for each page table entry are required by this policy if $N = 100$? Explain. You should only count metadata bits that are specifically needed to implement the Nth chance page replacement policy.

7 bits. He needs $\lceil \log_2 100 \rceil = 7$ bits for the counter. There are other bits such as the dirty, writable, and valid bits that need to be kept track, but they are technically not unique to the Nth chance algorithm. However, full credit was given regardless of if you mentioned these bits.

ii. (4.0 pt) Marcus observes that writing dirty pages back to disk is taking a substantially long time. What is ONE change he can make to the Nth chance page replacement policy to favor eviction of clean pages over dirty pages? Explain.

He can give dirty pages an extra chance by evicting dirty pages when the counter reaches $N + 1 = 101$ instead of N. Only evicting clean pages is not an option since the cache may become full with dirty pages.

iii. (4.0 pt) Marcus adds a hardware TLB in hopes of speeding up his lookup times. How many physical memory operations are required to read a single 32-bit word in the worst-case? Explain.

2 operations. We need to lookup the page table and read the actual memory in. A TLB lookup does not count as a physical memory operation since the TLB is a separate hardware component. No cache operations are counted since no such assumption of an existence of a cache was made.

7. (40.0 points) Hallowbean Party

After a tiring first half of the semester, Sean and Edward decide to throw a large bean potluck party with their CS 162 students in the Wozniak patio.

(a) Edward is organizing the food. However, there are only 3 slots on the table to serve all the bean dishes that CS 162 students have cooked. If a student wants eat a bean dish different than the three currently on the table, they need to ask a TA to make space on the table by paging a dish out to the refrigerator.

Luckily, we studied demand paging during the semester. Therefore, whenever a student would like to eat a dish that is not currently ready to serve, a dish fault (i.e page fault) will occur. The TAs will evict a dish from the table and place it back in the refrigerator. Then, the TA will bring the requested dish from the refrigerator onto the newly freed table space.

To ensure the food is served quickly, Edward asks you to examine different page replacement policies including LRU, MIN, and clock.

Say we have an access pattern of

Pinto, Black, Kidney, Lima, Black, Edamame, Pinto, Kidney, Lima, Pinto

with 3 slots on the table. A chart is given for your convenience using the first letter of each bean.

	P	B	K	L	B	E	P	K	L	P
1										
2										
3										

For each replacement policy, fill in the slot number each dish requested is placed on (i.e. answer 1, 2, or 3). If a dish is already on the table answer 0.

Then calculate the effective access time (EAT) in seconds to get served a dish assuming it takes 10 seconds to serve a dish on a table and 60 seconds to replace a dish on the table with a dish from the refrigerator. Show your work.

i. (8.0 points) LRU

A. (0.5 pt) Pinto

1

B. (0.5 pt) Black

2

C. (0.5 pt) Kidney

3

D. (0.5 pt) Lima

1

E. (0.5 pt) Black

0

F. (0.5 pt) Edamame

3

G. (0.5 pt) Pinto

1

H. (0.5 pt) Kidney

2

I. (0.5 pt) Lima

3

J. (0.5 pt) Pinto

0

K. (3.0 pt) EAT

The replacement chart from the results above is

	P	B	K	L	B	E	P	K	L	P
1	P			L			P			+
2		B			+			K		
3			K			E			L	

58 seconds. EAT = Hit Rate × Hit Time + Miss Rate × Miss Time $= 0.2 \times 10 + 0.8 \times (10 + 60) = 58$**.**

ii. (8.0 points) **MIN**

A. (0.5 pt) Pinto

1

B. (0.5 pt) Black

2

C. (0.5 pt) Kidney

3

D. (0.5 pt) Lima

3

E. (0.5 pt) Black

0

F. (0.5 pt) Edamame

2

G. (0.5 pt) Pinto

0

H. (0.5 pt) Kidney

2

I. (0.5 pt) Lima

0

J. (0.5 pt) Pinto

0

K. (3.0 pt) EAT

The replacement chart from the results above is

	P	B	K	L	B	E	P	K	L	P
1	P						+			+
2		B			+	E		K		
3			K	L					+	

46 seconds. EAT $=$ **Hit Rate** $\times$ **Hit Time** $+$ **Miss Rate** $\times$ **Miss Time** $= 0.4 \times 10 + 0.6 \times (10 + 60) = 46$**.**

iii. (8.0 points) Clock

Assume the clock hand that starts at 3 and moves in the direction of increasing page number.

A. (0.5 pt) Pinto

1

B. (0.5 pt) Black

2

C. (0.5 pt) Kidney

3

D. (0.5 pt) Lima

1

E. (0.5 pt) Black

0

F. (0.5 pt) Edamame

3

G. (0.5 pt) Pinto

2

H. (0.5 pt) Kidney

1

I. (0.5 pt) Lima

3

J. (0.5 pt) Pinto

0

K. (3.0 pt) EAT

The replacement chart from the results above is

	P	B	K	L	B	E	P	K	L	P
1	P			L				K		
2		B			+		P			+
3			K			E			L	

58 seconds. EAT = Hit Rate $\times$ Hit Time + Miss Rate $\times$ Miss Time $= 0.2 \times 10 + 0.8 \times (10 + 60) = 58$**.**

(b) After all the students have enjoyed their beans, Sean decides to host an axe throwing competition. All CS 162 students gather to compete, and the student who hits the target the most number of times will win. To ensure the game is safe, Sean adds several locks to prevent students from throwing the axe until they hold the lock for both the target and the axe. Until then, students remain blocked as they wait until it's safe to throw.

Complete the methods `compete` and `tidy_axes` to ensure a safe and deadlock free throwing environment. You are only allowed to write one line of code per line (fill in **two** lines for [B], [C], [E], [F]). You may not use semicolons to write multiple lines. You are only allowed to use methods and data structures provided to you in this problem and the reference sheet.

```
typedef struct {
  pthread_mutex_t axe_lock;
  pthread_mutex_t target_lock;
} game_data_t;

typedef struct {
  int id;
  int num_targets_hit;
  pthread_t thread;
  game_data_t* game_data;
} thread_data_t;

/* Throws axe and returns 1 if target is hit, 0 otherwise. */
int throw_axe();

/* Sharpens an axe and cleans up the target. */ */
void cleanup();

void* compete(void* arg) {
  __________[A]__________;

  __________[B]__________;
  __________[B]__________;

  thread_data->num_targets_hit += throw_axe();
  thread_data->num_targets_hit += throw_axe();
  thread_data->num_targets_hit += throw_axe();

  __________[C]__________;
  __________[C]__________;

  return NULL;
}

void* tidy_axes(void* arg) {
  __________[D]__________;

  __________[E]__________;
  __________[E]__________;

  cleanup();

  __________[F]__________;
  __________[F]__________;
  return NULL;
}

int main(int argc, char* argv[]) {
  int N = atoi(argv[1]);

  game_data_t game_data;
  pthread_mutex_init(&game_data.axe_lock, NULL);
  pthread_mutex_init(&game_data.target_lock, NULL);

  thread_data_t thread_data[N];
  for (int i = 0; i < N; i++) {
    thread_data[i].id = i;
    thread_data[i].num_targets_hit = 0;
    thread_data[i].game_data = &game_data;
    pthread_create(&thread_data[i].thread, NULL, compete, &thread_data[i]);
  }

  pthread_t tidy_threads[N / 10];
  for (int i = 0; i < N; i++) {
    pthread_join(thread_data[i].thread, NULL);
    printf("Competitor %d hit %d targets\n", i, thread_data[i].num_targets_hit);

    if (i % 10 == 0) {
      pthread_create(&tidy_threads[i / 10], NULL, tidy_axes, &game_data);
      pthread_join(tidy_threads[i / 10], NULL);
    }
  }

  return 0;
}
```

i. (2.0 pt) [A]

```
thread_data_t* thread_data = (thread_data_t*) arg
```

ii. (3.0 pt) [B]

```
pthread_mutex_lock(&thread_data->game_data->axe_lock)
pthread_mutex_lock(&thread_data->game_data->target_lock)
```

The lock acquisition order must be the same as [E] to prevent deadlock.

iii. (3.0 pt) [C]

```
pthread_mutex_unlock(&thread_data->game_data->target_lock)
pthread_mutex_unlock(&thread_data->game_data->axe_lock)
```

The lock release order does not matter.

iv. (2.0 pt) [D]

```
game_data_t* game_data = (game_data_t*) arg
```

v. (3.0 pt) [E]

```
pthread_mutex_lock(&game_data->axe_lock)
pthread_mutex_lock(&game_data->target_lock)
```

The lock acquisition order must be the same as [B] to prevent deadlock.

vi. (3.0 pt) [F]

```
pthread_mutex_unlock(&game_data->target_lock)
pthread_mutex_unlock(&game_data->axe_lock)
```

The lock release order does not matter.

8. References

(a) C API

```
/* Semaphore */
int sem_init(sem_t *sem, int pshared, unsigned int value);
int sem_wait(sem_t *sem);
int sem_post(sem_t *sem);

/* Lock */
int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *attr);
int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);

/* Condition Variable */
int pthread_cond_init(pthread_cond_t *cond, const pthread_condattr_t *attr);
int pthread_cond_signal(pthread_cond_t *cond);
int pthread_cond_broadcast(pthread_cond_t *cond);
int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex);

/* Process */
pid_t fork(void);
pid_t wait(int *status);
pid_t waitpid(pid_t pid, int *status, int options);
int execv(const char *path, char *const argv[]);

/* Thread */
int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
                    void *(*start_routine) (void *), void *arg);
int pthread_join(pthread_t thread, void **retval);
void pthread_exit(void *retval);

/* High-Level IO */
FILE *fopen(const char *pathname, const char *mode);
int fclose(FILE *stream);
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);
fprintf(FILE * restrict stream, const char * restrict format, ...);

/* Low-Level IO */
int open(const char *pathname, int flags);
int close(int fd);
ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);
off_t lseek(int fd, off_t offset, int whence);
int dup(int oldfd);
int dup2(int oldfd, int newfd);
int pipe(int pipefd[2]);
int close(int fd);

/* Socket */
int socket(int domain, int type, int protocol);
int bind(int sockfd, struct sockaddr *addr, socklen_t addrlen);
int listen(int sockfd, int backlog);
int accept(int sockfd, structure sockaddr *addr, socklen_t *addrlen);
int connect(int sockfd, struct sockaddr *addr, socklen_t addrlen);
```

```
ssize_t send(int sockfd, const void *buf, size_t len, int flags);

/* Pintos List */
void list_init(struct list *list);
struct list_elem *list_head(struct list *list);
struct list_elem *list_tail(struct list *list);
struct list_elem *list_begin(struct list *list);
struct list_elem *list_next(struct list_elem *elem);
struct list_elem *list_end(struct list *list);
struct list_elem *list_remove(struct list_elem *elem);
bool list_empty(struct list *list);
#define list_entry(LIST_ELEM, STRUCT, MEMBER) ...
void list_insert(struct list_elem *before, struct list_elem *elem);
void list_push_front(struct list *list, struct list_elem *elem);
void list_push_back(struct list *list, struct list_elem *elem);
```

(b) Unit Conversions

Prefix	Factor
Gibi (Gi)	2^{30}
Mebi (Mi)	2^{20}
Kibi (Ki)	2^{10}
milli (m)	10^{-3}
micro (μ)	10^{-6}
nano (n)	10^{-9}

(c) Policy Acronyms

Acronym	Term
SJF	Shortest Job First
SRTF	Shortest Remaining Time First
FIFO	First In First Out
FCFS	First Come First Serve
RR	Round Robin
EDF	Earliest Deadline First
MLFQ	Multi-Level Feedback Queue
LRU	Least Recently Used
MIN	Minimum

No more questions.