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.
This is a **proctored, closed-book exam**. During the exam, you may **not** communicate with other people regarding the exam questions or answers in any capacity.

When answering questions, make no assumptions except as stated in the problems. If there that you believe is open to interpretation, please use the “Clarifications” button to request a clarification. We will issue an announcement if we believe your question merits one.

This exam has 4 parts of varying difficulty and length. Make sure you read through the exam completely before starting to work on the exam. Answering a question instead of leaving it blank will **NOT** lower your score. We will overlook minor syntax errors in grading coding questions.

(a)

Name


(b)

Student ID


(c)

Please read the following honor code: “I understand that this is a closed book exam. I hereby promise that the answers that I give on the following exam are exclusively my own. I understand that I am allowed to use two 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 internet sources in constructing my answers.”

**Type your full name below to acknowledge that you’ve read and agreed to this statement.**


1. (18.0 points) True/False

Please EXPLAIN your answer in TWO SENTENCES OR LESS. Answers without any explanation GET NO CREDIT.

(a) (3.0 points)

i. In a page table-based addressing scheme, every virtual address in a process’ virtual address space always corresponds to a valid page table entry.

☐ True
☐ False

ii. Explain.

[Blank space for explanation]
(b) (3.0 points)

i. In a page table-based addressing scheme, the TLB is a cache for page table entries.
   ○ True
   ○ False

ii. Explain.
(c) (3.0 points)

i.
Page faults always cause a user program to terminate.
○ True
○ False

ii.
Explain.
(d) i. (3.0 points)

In PintOS with priority donation, immediately after a thread releases a lock, the thread’s effective priority may be different from the thread’s base priority.

A. True

B. False

B.

Explain.
(3.0 points)

In PintOS with priority donation, immediately after a thread releases a lock, the thread’s effective priority must be the same as the thread’s base priority.

ii. A) True
   ○ False

B. Explain.
(e) (3.0 points)

i. Assume that Job A always bursts for 10 ticks at a time, and runs forever. Also assume that our system uses a MLFQS (Multi-Level Feedback Queue Scheduler), with the following structure:

(Round-Robin (Q=8), Round-Robin (Q=16), FCFS)

If Job A is initially placed on the top queue when it enters the system, it will eventually be placed on the middle queue; after this point, when the job is not running, it will always be placed on the middle queue.

- True
- False

ii. Explain.
(f) (3.0 points)

i.
Assume that, in our system, there are 10 jobs that will run for 100 ticks total (and never voluntarily yield the CPU or block). If our system uses a round robin scheduler, increasing the quantum from 10 ticks to 20 ticks will always result in higher throughput.

☐ True
☐ False

ii.
Explain.
2. (18.0 points) Multiple Choice

In the following multiple choice questions, please select all options that apply.

(a) (3.0 pt)
Which of the following scheduling algorithms may cause starvation?

- [ ] First Come First Serve (FCFS)
- [ ] Round-Robin (RR)
- [ ] Shortest Remaining Time First (SRTF)
- [ ] Completely Fair Scheduler (CFS)
- [ ] None of the above

(b) (3.0 pt)
Which of the following descriptions of Banker’s Algorithm are correct?

- [ ] Banker’s Algorithm is used in deadlock recovery.
- [ ] Banker’s Algorithm can be used to determine whether or not a system is guaranteed to deadlock.
- [ ] Using Banker’s Algorithm results in an execution that is guaranteed to avoid deadlock.
- [ ] Banker’s Algorithm relies on having known upper limits for resource requests per thread.
- [ ] None of the above.

(c) (3.0 pt)
Which of the following scheduling algorithms are (1) deterministic and (2) will eventually preempt any infinitely looping thread in favor of a thread with finite run-time?

- [ ] First Come First Serve (FCFS)
- [ ] Round-Robin (RR)
- [ ] Multi-Level Feedback Queue Scheduling (MLFQS)
- [ ] Lottery Scheduling
- [ ] Strict Priority Scheduling
- [ ] Shortest Remaining-Time First Scheduling (SRTF)
- [ ] None of the above

(d) (3.0 pt)
Which of the following could be components of a page table entry (PTE)?

- [ ] Physical page number
- [ ] Virtual page number
- [ ] Valid bit
- [ ] Offset
- [ ] None of the above
(e) (3.0 pt)
Assume that the goal for our page replacement policy is to minimize total page replacements. Which of the following statements are accurate?

- FIFO can have the same performance as MIN.
- Clock Algorithm always has the same performance as Second-Chance List Algorithm.
- LRU performs at least as good as any Nth-Chance Clock Algorithm.
- An optimal page replacement policy prevents thrashing.
- None of the above.

(f) (3.0 pt)
Assume we make a single memory access on a processor which has no caches and no TLB. All else being equal, which of the following are true?

- Base-and-bound will on average be faster than a single-level page table.
- An inverted page table will on average be faster than a single-level page table.
- A single-level page table will on average be faster than a multi-level page table.
- A multi-level page table will on average be faster than an inverted page table.
- None of the above.
3. (38.0 points) Short Answer
   (a) (11.0 points) Virtual Memory Bits Pieces

   Suppose we have virtual memory mapped with a two-level page table scheme, where each page table fits exactly inside a page, and each page table entry is exactly 4 \(2^2\) bytes in size. We also have a physical memory size of 16 MiB \(2^{24}\) B, and a page size of 4 KiB \(2^{12}\) B. Additionally, we have a fully-associative tagged TLB, where each entry includes 1 byte of metadata containing information including a valid bit, dirty bit, and a process ID.

   i. (1.0 pt)
      How many bits are in the offset of an address in this scheme?

   ii. (1.0 pt)
      How many bits are in the first-level page table index in this scheme?

   iii. (1.0 pt)
      How many bits are in the second-level page table index in this scheme?

   iv. (1.0 pt)
      How many bits are in the physical page number in this scheme?

   v. (1.0 pt)
      How large of a virtual address space does this scheme address, in bytes? You may leave your answer as a power of 2.

   vi. (2.0 pt)
      If our TLB had 8 entries, how big is it in bytes?
vii. (2.0 pt)

Assume we start with an empty TLB and make the following memory accesses:
- Process 1 accesses 0x10000000
- Process 1 accesses 0x10001000
- Process 2 accesses 0x10000000

How many entries of our TLB will now be filled?

viii. (2.0 pt)

How many different pages will now be in memory, assuming there were 0 valid pages in physical memory before these accesses?
(b) (9.0 points) Rocky Wrench’s Replacements

Monty Mole and Rocky Wrench disagree about the best page replacement policies for demand paging – each is trying to throw a wrench into the other’s arguments! In the following problems, assume that the system has 3 pages of physical memory (1-3), and the program they run uses 4 pages of virtual memory (A-D). Provide lists as a comma-separated list of virtual memory accesses (e.g. A,B,C,D,A).

i. (3.0 pt)

Rocky Wrench is convinced that the best page replacement policy for demand paging is FIFO (First In First Out). Demonstrate the problems with FIFO by providing a list of 6 additional accesses (after the first two accesses that are given) that would cause every access to page fault.

<table>
<thead>
<tr>
<th>Page \ Access</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ii. (3.0 pt)

Monty Mole is convinced that the best page replacement policy for demand paging is MRU (Most Recently Used). Demonstrate the problems with MRU by providing a list of 6 additional accesses (after the first two accesses that are given) that would cause every access to page fault.

<table>
<thead>
<tr>
<th>Page \ Access</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

iii. (3.0 pt)

After putting their disagreements aside, they come up with an optimal page replacement policy. Monty Mole and Rocky Wrench use that replacement policy with a memory access pattern that contains 3 A’s, 2 B’s, 2 C’s, and 1 D (not necessarily in that order; you should not assume it begins with A and B). What is the maximum total number of page faults that could occur when using that optimal page replacement policy on such a memory access pattern?

<table>
<thead>
<tr>
<th>Page \ Access</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
(c) (4.0 points) Rosalina’s Special Scheduler

Rosalina’s Round-Robin (RR) scheduler is a special implementation of RR that considers thread priorities. More specifically, the quanta of a thread (in ticks) is equal to its priority. Say there is an IO-bound thread, Thread A, with a priority of 64 that always bursts for 1 tick in between I/O. Another CPU-bound thread, Thread B, has a priority of 2 and always bursts for 100 ticks in between I/O. Assume each I/O operation takes 1 tick.

i. (2.0 pt)

Assuming the two threads only yield for I/O, for what fraction of time will Thread A run on the CPU on average?

ii. (2.0 pt)

Why doesn’t the higher priority thread (Thread A), with a higher quanta, run for a majority of the time?
(d) (14.0 points) Mario's Moves

Mario wrote a program that synchronizes three of his special moves as threads.

```c
sem_t semaphore;
pthread_mutex_t lock;

void threadA() {
    sem_wait(&semaphore); // line 1
    printf("threadA!");
    sem_post(&semaphore)
}

void threadB() {
    sem_wait(&semaphore); // line 1
    pthread_mutex_lock(&lock); // line 2
    printf("threadB!");
    pthread_mutex_unlock(&lock);
    sem_post(&semaphore);
}

void threadC() {
    sem_wait(&semaphore); // line 1
    pthread_mutex_lock(&lock); // line 2
    sem_wait(&semaphore); // line 3
    printf("threadC!");
    sem_post(&semaphore);
    pthread_mutex_unlock(&lock);
    sem_post(&semaphore);
}

int main() {
    sem_init(&semaphore, 0, /*initial value*/ 2);
    pthread_mutex_init(&lock, NULL);
    pthread_t threads[3];
    pthread_create(&threads[0], NULL, threadA, NULL);
    pthread_create(&threads[1], NULL, threadB, NULL);
    pthread_create(&threads[2], NULL, threadC, NULL);
}```
i. (4.0 pt)
What order of execution will cause this program to print nothing? Provide your answer as a list of lines of thread:N, specifying the order of execution of the lines of code of each thread.
NOTE: include any lines/expressions that start executing but may not finish executing.

Example:
threadA:1
threadC:1
threadC:2
threadC:3
threadB:1

ii. (2.0 pt)
What is name of the condition that causes the program to print nothing?

iii. (4.0 points)
Mario realizes his mistake, and decides to implement a new function to replace pthread_mutex_lock to prevent this from happening. Complete the function below to ensure that all three print statements are executed regardless of how the threads are scheduled (assume that every pthread_mutex_lock(&lock) line is replaced with pthread_mutex_lock_safe()). You may only use one statement/expression in each of the provided blanks.

```c
void pthread_mutex_lock_safe() {
    ___[A]___;
    pthread_mutex_lock(&lock);
    ___[B]___;
}
```

A. 

[A]
B.

[B]
iv. (4.0 pt)

Explain how your solution above changes the ordering of resource allocation to prevent the condition you noted in 4.2.
4. (26.0 points) Monty Mole’s Memory Management

In order to reduce the latency of L1 cache hits, many modern CPU architectures employ a type known as Virtually-Indexed, Physically-Tagged (VIPT) caches. In this system, rather than having to perform address translation to be able to look up an entry in the cache, the CPU instead selects the cache set based on the virtual address in parallel with address translation, only afterwards storing or comparing bits from the now-translated physical address for the tag. You have been contracted to help Monty Mole implement such a design for the L1 data cache of an application-specific operating system.

After much deliberation, you and your team of Koopalings have decided on a 256 byte, 2-way set associative cache with LRU replacement and 16-byte cache lines. To test the effectiveness of this layout, you decide to simulate a series of cache accesses on your new design.

Given the list of memory addresses, the initial state of the cache, and the layout of physical memory below, fill out the the final state of the cache once all memory accesses are complete. An LRU bit value of 0 indicates that that set’s Entry 0 is up for eviction, while an LRU bit value of 1 indicates the same for Entry 1.

Assume the following details hold for the system throughout the problem:

- The operating system uses single-level paging for address translation.
- Pages are 256 bytes in size.
- The Page Table Base Register contains the value 0x10000000.
- All relevant virtual addresses are present within the system’s TLB before execution begins, and remain there throughout.
- All memory contents are displayed in big-endian order.
- Each Page Table Entry is 4 bytes long: bits [31:8] contain the PPN, bit 0 contains the valid bit, and bits [7:1] are unspecified.
- All pages are fully accessible by the running process.

You may find it helpful to use the emailed exam PDF as a reference in this problem. The physical memory table below is also available as a separate PDF. Some answers may be related, so use shortcuts wherever possible!

<table>
<thead>
<tr>
<th>Address</th>
<th>0x0</th>
<th>0x4</th>
<th>0x8</th>
<th>0xC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x10000000</td>
<td>0x10840111</td>
<td>0x330CD003</td>
<td>0x00000000</td>
<td>0x10010000</td>
</tr>
<tr>
<td>0x10000010</td>
<td>0xB33A0001</td>
<td>0xA55A0200</td>
<td>0x01A05203</td>
<td>0x00000000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0x10000000</td>
<td>0xA55A0201</td>
<td>0x5500BE47</td>
<td>0xA55A0001</td>
<td>0xA55A0101</td>
</tr>
<tr>
<td>0x10000010</td>
<td>0x3300C81</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td>0xA55A0200</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0x12100440</td>
<td>0x3300BE47</td>
<td>0x33000000</td>
<td>0x3300C001</td>
<td>0xB55A0201</td>
</tr>
<tr>
<td>0x12100450</td>
<td>0x730F1101</td>
<td>0x730F1201</td>
<td>0x730F1301</td>
<td>0xB55A0200</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0x0300C000</td>
<td>0xF000000</td>
<td>0x0F000000</td>
<td>0x0F000000</td>
<td>0x0F000000</td>
</tr>
<tr>
<td>0x0300C010</td>
<td>0x40000000</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x0300C020</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x0300C030</td>
<td>0x0F000000</td>
<td>0x0F000000</td>
<td>0x0F000000</td>
<td>0x0F000000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0x0300C0C0</td>
<td>0x5EAF000D</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td>0x0001FFFF</td>
</tr>
<tr>
<td>0x0300C0D0</td>
<td>0x0EFE000E</td>
<td>0x0F000000</td>
<td>0x0F000000</td>
<td>0x0F000000</td>
</tr>
<tr>
<td>0x0300C0E0</td>
<td>0x0F000000</td>
<td>0x0EFE000E</td>
<td>0x0EFE000E</td>
<td>0x0EFE000E</td>
</tr>
</tbody>
</table>
### Hex to Binary Conversion Table

<table>
<thead>
<tr>
<th>Hex</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>0001</td>
<td>0001</td>
</tr>
<tr>
<td>0010</td>
<td>0010</td>
</tr>
<tr>
<td>0011</td>
<td>0011</td>
</tr>
<tr>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>0101</td>
<td>0101</td>
</tr>
<tr>
<td>0110</td>
<td>0110</td>
</tr>
<tr>
<td>0111</td>
<td>0111</td>
</tr>
<tr>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>1001</td>
<td>1001</td>
</tr>
<tr>
<td>1010</td>
<td>1010</td>
</tr>
<tr>
<td>1011</td>
<td>1011</td>
</tr>
<tr>
<td>1100</td>
<td>1100</td>
</tr>
<tr>
<td>1101</td>
<td>1101</td>
</tr>
<tr>
<td>1110</td>
<td>1110</td>
</tr>
<tr>
<td>1111</td>
<td>1111</td>
</tr>
</tbody>
</table>

### LRU Bits Table

<table>
<thead>
<tr>
<th>Set</th>
<th>LRU Bit</th>
<th>Tag for Entry 0</th>
<th>Tag for Entry 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0x3300C0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0xA55A01</td>
<td>0x800230</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0xB55A03</td>
<td>0xB55A04</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0xA55A02</td>
<td>0x800113</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0x800010</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0x3300C0</td>
<td>0x840112</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>0x70F3F0</td>
<td></td>
</tr>
</tbody>
</table>

You may also find this table of hex to binary conversions helpful:
(a) (15.0 points)

<table>
<thead>
<tr>
<th>Virtual Address</th>
<th>Physical Address</th>
<th>Set (0.5 pt ea.)</th>
<th>Hit? (0.5 pt ea.)</th>
<th>Value (1 pt ea.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x840110C0</td>
<td>(a)</td>
<td>6</td>
<td>Hit</td>
<td>0xBEEFBEEF</td>
</tr>
<tr>
<td>0x00F003DC</td>
<td>(b)</td>
<td>6</td>
<td>Miss</td>
<td>0x0005A1AD</td>
</tr>
<tr>
<td>0x00F00324</td>
<td>(c)</td>
<td>(d)</td>
<td>(e)</td>
<td>(f)</td>
</tr>
<tr>
<td>0x840110C0</td>
<td>(g)</td>
<td>6</td>
<td>Hit</td>
<td>0xBEEFBEEF</td>
</tr>
<tr>
<td>0x84011208 0x3300C008</td>
<td>(h)</td>
<td>(i)</td>
<td>(j)</td>
<td></td>
</tr>
<tr>
<td>0x840112D0 0x3300C0D0</td>
<td>(k)</td>
<td>(l)</td>
<td>(m)</td>
<td></td>
</tr>
<tr>
<td>0x00F00040 0xA55A0240</td>
<td>(n)</td>
<td>(o)</td>
<td>(p)</td>
<td></td>
</tr>
<tr>
<td>0x00F00070 0xA55A0270</td>
<td>(q)</td>
<td>(r)</td>
<td>(s)</td>
<td></td>
</tr>
<tr>
<td>0x00000164</td>
<td>(t)</td>
<td>3</td>
<td>Miss</td>
<td>0xC001C142</td>
</tr>
</tbody>
</table>

i.

(a)

ii.

(b)

iii.

(c)

iv.

(d)

v.

(e)
- Hit
- Miss
vi. (f)

vii. (g)

viii. (h)

ix. (i)
  ○ Hit
  ○ Miss

x. (j)

xi. (k)

xii. (l)
  ○ Hit
  ○ Miss
xiii.  
(m)  

xiv.  
(n)  

xv.  
(o)  
⊙ Hit  
⊙ Miss  

xvi.  
(p)  

xvii.  
(q)  

xviii.  
(r)  
⊙ Hit  
⊙ Miss  

xix.  
(s)  

xx.

(t)
(b) (5.0 points)

Suppose we were to entirely remove the TLB from the system.

i. (2.0 pt)

Would you expect the total number (not the rate) of cache misses to increase, decrease, or be unaffected by the change?

- Increase
- Decrease
- No Change

ii. (3.0 pt)

Why so?
(c) (6.0 points)

Imagine if you were to scale the capacity of the cache by a factor of 4, from 256 bytes to 1 KiB, while leaving all other parameters (associativity, replacement policy, etc.) unchanged. Suppose two processes (different virtual -> physical mappings) execute on the CPU one after another, resulting in the following two virtual/physical address combinations being present in the cache simultaneously:

<table>
<thead>
<tr>
<th>Process</th>
<th>Virtual Address</th>
<th>Physical Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0x400000F4</td>
<td>0x700000F4</td>
</tr>
<tr>
<td>B</td>
<td>0x5594FFF0</td>
<td>0x700000F0</td>
</tr>
</tbody>
</table>

i. (3.0 pt)

What problem(s) could this cause?

ii. (3.0 pt)

Could this problem also occur in the 256-byte VIPT cache described above? Why so or why not?
5. Reference Sheet

```
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);
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);
int sem_init(sem_t *sem, int pshared, unsigned int value);
int sem_post(sem_t *sem); // up
int sem_wait(sem_t *sem); // down
int pthread_cond_init(pthread_cond_t *cond, pthread_condattr_t *cond_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);
```

```
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[]);
```

```
FILE *fopen(const char *path, const char *mode);
FILE *fdopen(int fd, const char *mode);
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);
int fclose(FILE *stream);
```

```
int open(const char *pathname, int flags);
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);
```
No more questions.