CS 162 Spring 2021 Operating Systems and Systems Programming

MIDTERM 3

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 is a question 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 6 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. Please include units and simplify final answers where it makes sense; you may leave unsimplified or symbolic answers in the boxes for work.

(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 three 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. **True/False**

   (a) **(3.0 points)**

   i.

   Direct Memory Access (DMA) is when the CPU communicates with an IO device using built-in instructions that read/write to memory.

   ○ True

   ● False

   ii.

   Explain your answer.

   False: This is memory mapped IO. DMA lets the IO device communicate directly with RAM without needing to go through the CPU.

(b) (3.0 points)

i.

Since the File Allocation Table (FAT) does not actually store file contents, the FAT is not stored on disk.

◯ True

● False

ii.

Explain your answer.

False: We need the FAT to properly access our files, so it must also be stored in persistent storage.

(c) (3.0 points)

i.

The FAT filesystem suffers from more external fragmentation than the Berkeley FFS discussed in class.

○ True

● False

ii.

Explain your answer.

False: Neither suffers from external fragmentation since they rely on blocks.

(d) (3.0 points)

i.

Obi-Wan needs to tell Anakin that he has the high ground.

If Obi-Wan sends a packet from his computer to Anakin's computer over the internet, an intermediate router does not need to access the packet's IP header.

○ True

● False

ii.

Explain your answer.

False. An intermediate router must access the IP header to determine the IP address of the destination host and route the packet accordingly.

(e) (3.0 points)

i.

Emperor Palpatine is making Remote Procedure Calls (RPC) from a client machine to execute Order 66 on a server machine. If the server doesn't need to send a return value to the client, then a server stub is not needed.

○ True

● False

ii.

Explain your answer.

False. The server stub is still required in order to unmarshal the data sent by the client machine.

2. Multiple Select

(a) (3.0 pt)

Which of the following are advantages of magnetic disks over flash memory?

- [x] Lower monetary cost per byte
- [ ] Higher throughput random writes
- [ ] Lower latency random reads
- [x] Hardware durability over many, many writes
- [ ] None of the Above

Magnetic disks have slower random access, but are cheaper and have good hardware durability.

(b) (3.0 pt)

In UNIX, which of the following I/O devices can be accessed using file operations like `open` and `close`?

- [x] Network devices (wifi, bluetooth)
- [x] A mouse
- [x] A keyboard
- [x] The internal hard drive
- [x] An external flash drive
- [ ] None of the Above

In UNIX, all these input/output resources are identified by file descriptors, allowing us to use file operations.

(c) (3.0 pt)

Which of the following will decrease average total latency (per I/O request) for an HDD device?

- [x] Repositioning frequently accessed data from inner tracks to outer tracks
- [x] Increasing the rotational speed of the disk
- [ ] Repositioning frequently accessed data from upper platters to lower platters
- [ ] None of the Above

Total latency is partially determined by rotation time and seek time. Each platter has its own head.

(d) (3.0 pt)

Adding a Flash Translation Layer (FTL):

- [x] Increases DRAM utilization
- [ ] Requires the kernel to translate virtual block numbers to physical page numbers
- [x] Reduces wear out
- [ ] Can decrease the latency of read requests
- [ ] None of the Above

FTLs maintain mapping tables in DRAM to maintain indirection and copy-on-right. They reduce write amplification and wear out by translating logical blocks to low-level flash blocks and pages.

(e) (3.0 pt)

Which of the following protocols are stateful (not stateless)?

- ☐ User Datagram Protocol (UDP)
- ■ Transmission Control Protocol (TCP)
- ☐ Network File System (NFS)
- ■ Two-Phase Commit (2PC)
- ☐ None of the Above

TCP maintains state of open connections. 2P maintains state of which step a coordinator/worker is in.

3. Stardust

The Empire needs a place to store its super-secret Death Star plans, so you've been hired to design a system for them.

(a) (4.0 points)

i.

Suppose you choose to store the plans on an inode-based file system where blocks are 32 bytes and file block pointers are 32 bits. You are given an inode of exactly 2 indirect pointers and 1 doubly indirect pointer. How many direct pointers should you add to support a maximum file size of *exactly* 3200 bytes?

20

ii.

Optionally, show your work below for partial credit.

Number of pointers per block: 32 bytes for 1 block / 4 bytes per file block pointer = 8 pointers per block. Thus, one indirect pointer references 8 direct pointers, and 1 doubly indirect pointer references 8*8 = 64 direct pointers.
block size * (data blocks from doubly indirect + data blocks from indirect + data blocks from direct) = maximum file size
32 B * (64 data blocks + 2*8 data blocks + X) = 3200 B
X = 20 direct pointers

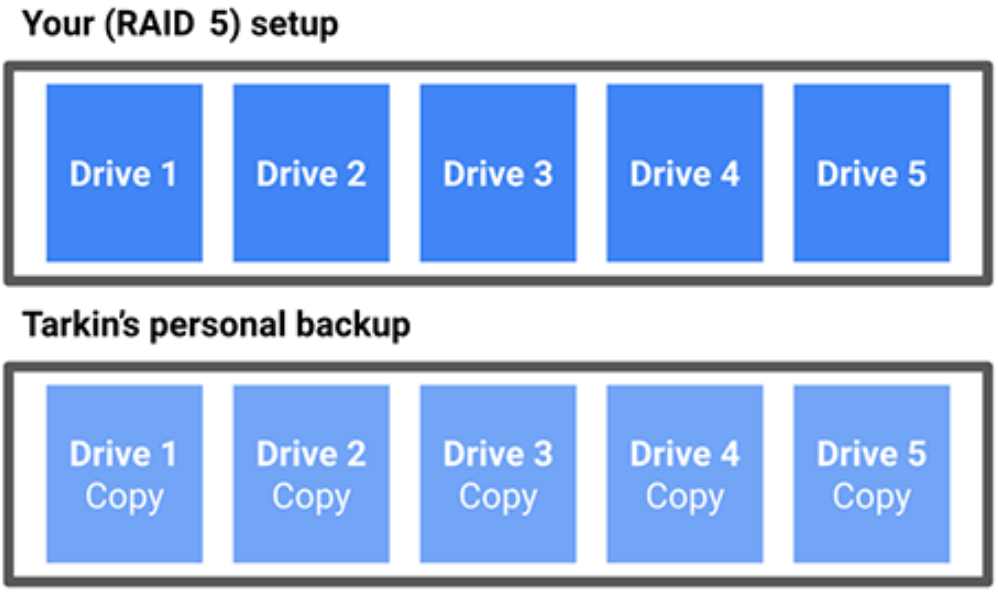
(b) (4.0 pt)

Darth Vader decides to splurge and purchase SSDs for the new storage system, so you consider using a log-structured file system instead. In what ways is using a log-structured filesystem better than using an inode-based file system on an SSD? Assume that the SSD device does not support a Flash Translation Layer (FTL).

In a log-structured file system, all new writes are written to a continual log, rather than erasing and rewriting existing blocks, which are slower operations on an SSD, and wears the SSD out faster.

(c) (4.0 pt)

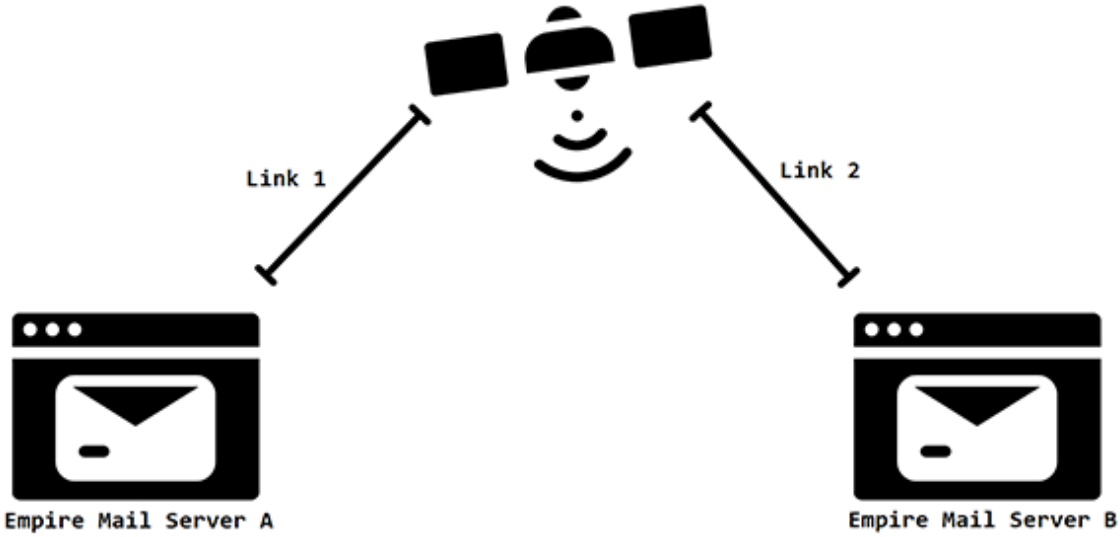
You realize that the blueprints are too large to fit on a single disk drive, so instead you store the data across 5 drives using RAID 5. Grand Moff Tarkin also decides to maintain a backup copy of your RAID 5 setup, mirroring it on 5 of his personal drives. If there are two simultaneous drive failures across all 10 drives, can the Empire recover all the data? Why or why not? Be sure to consider all possible scenarios.



Yes. If the two failed drives are in the same set of 5, they can be replaced from the other set of 5. If the two failed drives are in different sets of 5, they can each be reconstructed via RAID 5's redundancy.

(d) (4.0 points)

Now, Grand Moff Tarkin wants to send the blueprints over interstellar satellite connections, from Empire Mail Server A to Empire Mail Server B. Empire Mail Server A's connection to the satellite is denoted Link 1, while Empire Mail Server B's connection to the satellite is denoted Link 2.



Assume the following information:

- 120 ms latency from either server to the satellite, and vice versa (both Link 1 & Link 2)
- 10 ms latency for the router machinery within the satellite; assume that the router can process packets at full line rate (i.e. as fast as needed by the network)
- Maximum Transfer Unit of 1,000 B (both Link 1 and Link 2)
- Bandwidth over Link 1: 200 GiB/s
- Bandwidth over Link 2: 250 GiB/s
- TCP/IP Header size: 50 B

Note that the bandwidth is given in GiB/s for convenience, instead of Gb/s, which would typically be used.

i. **A.**

Assuming that Empire Mail Server A and Empire Mail Server B are communicating via a mail-transfer protocol carried over TCP/IP, what is the maximum sustained communications bandwidth (excludes overhead) that the two servers could achieve over the satellite connection?

190 GiB/s

B.

Optionally, show your work below for partial credit.

(1000B - 50B)/1000B * min(200GiB/s, 250GiB/s) = 190GiB/s

ii. A.

Assuming that Empire Mail Server A and Empire Mail Server B are communicating via a mail-transfer protocol carried over TCP/IP. Based on the given assumptions, what is the Round Trip Time (RTT) for a 512 B packet?

500ms

B.

Optionally, show your work below for partial credit.

2 * (120ms + 10ms + 120ms) = 500ms

4. Got Milk?

Alina, William, and Allan are native residents of Tatooine, and they all drink the same type of milk everyday. At the end of each day, the trio runs Two-Phase Commit (2PC) to ensure that they will always drink the same type of milk.

This means that, for each day, they will either choose to:

(a) Agree to change beverages OR
(b) Drink the same beverage as the previous day

If no action has been determined, they will choose to drink the same type of milk as the previous day by default (gross).

Alina is the coordinator and William/Allan are followers. Today, they are drinking **white** milk and Alina wants to drink **blue** milk tomorrow.

The termination protocol for each worker is to wait for the coordinator to recover.

Assuming no crashes and timeouts, here is the correct sequence of 2PC logs. Use it as a reference for the questions below.

```
1) Alina: decides she wants everyone to drink blue milk tomorrow
2) Alina: writes "START-2PC: Drink blue milk tomorrow" to her log

3) Alina transmits to William: "VOTE-REQ: Let's drink blue milk tomorrow"
4) William: writes "VOTE-COMMIT: I'm ready to change beverages" to his log
5) William transmits to Alina: "VOTE-COMMIT: I'm ready to change beverages"

6) Alina transmits to Allan: "VOTE-REQ: Let's drink blue milk tomorrow"
7) Allan: writes "VOTE-COMMIT: I'm ready to change beverages" to his log
8) Allan transmits to Alina: "VOTE-COMMIT: I'm ready to change beverages"

9) Alina: writes "GLOBAL-COMMIT: Okay, drink blue milk tomorrow" to her log

10) Alina transmits to William: "GLOBAL-COMMIT: Okay, drink blue milk tomorrow"
11) William: writes "GLOBAL-COMMIT: Drink blue milk tomorrow" to his log
12) William: Milk = blue
13) William transmits to Alina: "Changed my beverage successfully!"

14) Alina transmits to Allan: "GLOBAL-COMMIT: Okay, drink blue milk tomorrow"
15) Allan: writes "GLOBAL-COMMIT: Drink blue milk tomorrow" to his log
16) Allan: Milk = blue
17) Allan transmits to Alina: "Changed my beverage successfully!"

18) Alina: Milk = blue
19) Alina: writes "END 2PC" to her log
```

Only assume the crashes specified in each question. You can assume that no other crashes will occur. Assume that all transmissions over the network are instantaneous and guaranteed to succeed.

(a) (4.0 points)

i.

Assume Alina crashes after step 8 and no one else fails. Assuming that Alina recovers before the end of the day, is everyone guaranteed to drink blue milk tomorrow?

○ Yes

● No

ii.

Why or why not?

No. On reboot, Alina will see she hasn't logged a global action but has started a vote request. Thus, she will send out a GLOBAL-ABORT and everyone will drink the same white milk.

(b) (4.0 points)

i.

Now, assume Alina crashes after step 2 and never recovers, but no one else fails. Are William and Allan guaranteed to drink white milk tomorrow?

- (●) Yes
- ( ) No

ii.

Why or why not?

Yes. Because has the coordinator never trasmitted the VOTE-REQ messages. As a result, the workers will timeout, abort, and halt.

(c) (4.0 points)

i.

Now, assume that Alina crashes after step 13 and recovers before the end of the day. Is everyone guaranteed to drink blue milk tomorrow?

- (●) Yes
- ( ) No

ii.

Why or why not?

Yes. Because a GLOBAL-COMMIT has been recorded in the log, the log guarantees that the trio must change beverages. Thus, the color of their milk will be blue the next day.

5. Millennium Falcon Mods

Chewie is working on upgrading his new spaceship to support faster-than-light travel for hyperspace jumps. To do so, he needs to replace the disk with a new disk that has higher bandwidth and capacity (apparently, the hyperdrive produces a lot of data very quickly!).

He finds a spaceship HDD with the following parameters:

- 100 TiB in size
- 10 GiB/s data transfer rate
- 3000 RPM
- 512 MiB block size
- 8 ms average seek time
- 1 ms controller delay

Note that the data transfer rate is given in GiB/s for convenience, rather than Gbytes/s, which would typically be used.

(a) (4.0 points)

Assuming no queueing at the controller, what is the average time to write a single random block to the disk?

i.

69 ms

ii.

Optionally, show your work below for partial credit.

T
= controller + queueing + seek + rotational + transfer
= 1 ms + 0 + 8 ms + (1/2)(1 / (3*10^4 RPM))(60 sec / min)(10^3 ms / sec) ms
+ ((512 MiB) / (10 GiB/s))(10^3 ms / sec) ms
= 1 + 0 + 8 + 10 + 50
= 69 ms

(b) (4.0 points)

Though the disk itself is quite fast, Chewie observes that disk operations sometimes take a while to be serviced. Chewie reasons that, since the spaceship HDD handles disk operations from many other parts of the ship besides the hyperdrive, there is non-negligible queueing time.

A block operation takes `Ts ms` on average to service, which you calculated above. You can use the approximation `Ts = 200/3 ms = 66.67 ms` to make calculations easier.

i.

Assume the distribution of service time has `C=2` (not memoryless), and these requests for blocks arrive via an exponential (memoryless) process. If we observe an average queue length of `L=2`, what is the average arrival rate of requests `lambda` in requests per ms?

Hint: Use Little's Law (`L = lambda * Tq`) and the quadratic formula:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

0.01 requests per ms

ii.

Optionally, show your work below for partial credit.

L = lambda * Tq, where Lq = 2
Tq = Ts * 0.5(1+C) * u/(1-u), where C = 2, u = lambda * Ts
2 / lambda = Ts * 0.5(1+2) * u/(1-u)
2 = lambda*Ts* (3/2) * u/(1-u)
2 = u * (3/2) * u/(1-u)
2 - 2*u* = *(3/2)* (u^2)
3u^2 + 4u - 4 = 0
Solve for u: u = 2/3
Solve for lambda: lambda = (2/3)/Ts = (2/3)/(200/3) = 1/100

(c) (4.0 pt)

Chewie continues to be plagued by excessive latency while using his new HDD, finding that the same HDD is used by both the hyperdrive computer and the hologram computer. Explain why implementing the buffer cache from Project 3 (with no modifications) in the hyperdrive computer's operating system may cause issues.

The buffer cache is write-back, which will cause data consistency issues since multiple systems are using the HDD simultaneously.

6. (30.0 points) PintOS watch

The Linux `epoll` and `inotify` syscalls allow a process to watch multiple file descriptors, blocking until some specified I/O event takes place on any of the file descriptors. One usage of these syscalls is to watch for modifications among a set of files (e.g. perhaps you'd like to write a program that automatically re-runs `make` whenever it detects changes to C code files in your current directory). **In this problem, you'll implement a very simple version of this functionality in PintOS.**

Specifically you'll implement a syscall which we'll call `watch` in PintOS which takes in a list of file descriptors, and blocks until one of them has been modified (i.e. written to). `watch` should return the file descriptor corresponding to the file that was modified. You need not implement any special handling for the standard input/output/error file-descriptors (i.e. 0, 1, 2). Assume that header files have been set up such that you can access the below functions and `struct` definitions from any file. Fill in the blanks in the below skeleton code to complete the implementation of `watch`. The number of blank lines provided serves as a recommendation, not as a minimum nor a maximum. Please note that this is a **coding** question: all solutions must be given in the C programming language. No credit will be awarded for pseudocode, the contents of comments, code which calls "helper functions" which do not exist, etc.

Hint: anything included in the skeleton (e.g. `struct` fields, helper functions, etc.) is there for a good reason. We'd recommend reading over the entire skeleton before starting to answer the question.

userprog/syscall.c

```
struct file_descriptor {
    // Other fields omitted
    bool is_dir;
    struct file* file;
    struct dir* dir;
};

// Returns the `struct file_descriptor*` corresponding to `fd`
struct file_descriptor* lookup_fd(int fd) {
    // Implementation omitted, you can assume all calls to this function succeed
    // so long as `fd` is a valid file descriptor, and that the current thread
    // is terminated otherwise.
}

struct watch_data {
    int fd;
    int ref_count; // Reference count
    struct lock lock;
    struct condition cv;
};

void release_watch_data(struct watch_data* data) {
    ASSERT(lock_held_by_current_thread(&data->lock));
    _______________[A]_________________
    lock_release(&data->lock);
    if (____[B]_____)
        free(data);
}

// Returns once one of `fds` has been written to
// Assume we have already verified `fds`, and `num_fds` is valid (i.e. >0)
static int sys_watch(const int* fds, unsigned num_fds) {
    struct watch_data* data = __________[C]__________;
    // Initialize `data`
```

```
    _____________[D]_________________
    _____________[D]_________________
    _____________[D]_________________
    _____________[D]_________________
    for (unsigned i = 0; i < num_fds; i++) {
        struct file_descriptor* fd = lookup_fd(fds[i]);
        struct inode* inode = fd->is_dir ? dir_get_inode(fd->dir)
                                         : file_get_inode(fd->file);
        _____________[E]_________________
    }
    _________[F]_______________
    while (____[G]_____) {
        ________[H]________
    }
    int modified = data->fd;
    release_watch_data(data);
    return modified;
}
```

filesys/inode.c

```
struct listener {
    int fd;
    struct watch_data* data;
    struct list_elem elem;
};

struct inode {
    // Other fields omitted
    struct lock listeners_lock; // You can assume this has been properly initialized
    struct list listeners; // You can assume this has been properly initialized
};

off_t inode_write_at(struct inode* inode, /* Other arguments omitted */) {
    // Code that actually writes to the inode omitted

    lock_acquire(&inode->listeners_lock);
    while (!list_empty(&inode->listeners)) {
        struct list_elem* elem = list_pop_back(&inode->listeners);
        struct listener* listener = list_entry(elem, struct listener, elem);
        ___________________[I]_______________________
        ___________________[I]_______________________
        ___________________[I]_______________________
        ___________________[I]_______________________
        ___________________[I]_______________________
        ___________________[I]_______________________
    }
    lock_release(&inode->listeners_lock);

    // Cleanup code omitted
}

void inode_add_listener(struct inode* inode, int fd, struct watch_data* data) {
    struct listener* listener = _______[J]_______
    // Initialize `listener`
    __________[K]_____________
```

```
    __________[K]_____________
    lock_acquire(&inode->listeners_lock);
    list_push_back(&inode->listeners, &listener->elem);
    lock_release(&inode->listeners_lock);
}
```

(a)

[A]

```
int new_ref_cnt = --data->ref_count;
```

(b)

[B]

```
new_ref_cnt == 0
```

(c)

[C]

```
malloc(sizeof(struct watch_data));
```

(d)

[D]

```
data->fd = -1;
data->ref_count = 1 + num_fds;
lock_init(&data->lock);
cond_init(&data->cv);
```

(e)

[E]

```
inode_add_listener(inode, fds[i], data);
```

(f)

[F]

```
lock_acquire(&data->lock);
```

(g)

[G]

```
data->fd == -1
```

(h)

[H]

```
cond_wait(&data->cv, &data->lock);
```

(i)

[I]

```
lock_acquire(&listener->data->lock);
if (listener->data->fd == -1)
    listener->data->fd = listener->fd;
cond_signal(&listener->data->cv, &listener->data->lock);
release_watch_data(listener->data);
free(listener);
```

(j)

[J]

```
malloc(sizeof(struct listener));
```

(k)

[K]

```
listener->data = data;
listener->fd = fd;
```

7. Reference Sheet

```
/********************************* Threads **********************************/
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);

/******************************** Processes *********************************/
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[]);

/****************************** High-Level I/O ******************************/
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);

/******************************* Low-Level I/O ******************************/
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);

/******************************* PintOS Lists *******************************/
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);

/****************************** PintOS Threads ******************************/
void sema_init(struct semaphore *sema, unsigned value);
```

```
void sema_down(struct semaphore *sema);
void sema_up(struct semaphore *sema);
void lock_init(struct lock *lock);
void lock_acquire(struct lock *lock);
void lock_release(struct lock *lock);
void cond_init(struct condition *cond);
void cond_wait(struct condition *cond, struct lock *lock);
void cond_signal(struct condition *cond, struct lock *lock);
void cond_broadcast(struct condition *cond, struct lock *lock);
enum intr_level intr_get_level(void);
enum intr_level intr_set_level(enum intr_level);
enum intr_level intr_enable(void);
enum intr_level intr_disable(void);
bool intr_context(void);
void intr_yield_on_return(void);
```

No more questions.