Computer Science 15-410/15-605: Operating Systems
Mid-Term Exam (A), Fall 2024

1. **Please read the entire exam before starting to write. This should help you avoid getting bogged down on one problem.**
2. Be sure to put your name and Andrew ID below.
3. **PLEASE DO NOT WRITE FAINTLY WITH PENCIL. Please write in ink, or, if writing in pencil, please ensure that zero strokes in zero words are faint. Using a mechanical pencil with thin lead is probably unwise.**
4. This is a closed-book in-class exam. You may not use any reference materials during the exam.
5. **If you have a clarification question, please write it down on the card we have provided. Please don't ask us questions of the form "If I answered like this, would it be ok?" or "Are you looking for ...?"**
6. The weight of each question is indicated on the exam. Weights of question *parts* are estimates which may be revised during the grading process and are for your guidance only.
7. Please be concise in your answers. You will receive partial credit for partially correct answers, but truly extraneous remarks may count against your grade.

Andrew Username	
Full Name	

Question	Max	Points	Grader
1.	**10**		
2.	**15**		
3.	**15**		
4.	**20**		
5.	**10**		
	70		

Please note that there are system-call and thread-library "cheat sheets" at the end of the exam.

If we cannot read your writing, we will be unable to assign a high score to your work.

1. $\boxed{\text{10 points}}$ Short answer.

   (a) $\boxed{\text{4 points}}$ In an abstract sense, a mutex is interchangeable with a readers/writers lock if the rwlock is always used in the `RWLOCK_WRITE` mode: either way, the number of threads executing the critical section would never be more than one. However, according to the 15-410 course staff, that abstract-interchangeability thinking is "correct but wrong." Briefly explain how, according to the 15-410 course staff, a programmer *should* decide whether to protect a given critical section with a mutex versus an `RWLOCK_WRITE` rwlock. Briefly justify your explanation. We are expecting an answer that will fit naturally on this page.

(b) $\boxed{\text{6 points}}$ When designing a body of code, at times one finds oneself thinking, "I wonder if I should use Approach A or Approach B?" According to the 15-410 design orthodoxy, you should follow a specific procedure to resolve your question. In accordance with that procedure, please document a design decision *about how/where to allocate memory for some object/purpose* that you made while working on your your Project 2 thread library. For the purposes of this question we will be scoring based on your *documentation* of the decision procedure, not whether we agree or disagree with what you chose. Begin with a brief description of the problem (one to three sentences) and then show us your design decision—answers that correctly use the 15-410 approved data structure will receive higher scores. For full credit, we expect that you will compare *implementable* options, as opposed to non-working ones.

You may use this page as extra space for the P2-design question if you wish.

2. $\boxed{\text{15 points}}$ Faulty Mutex

In this question you will examine some mutex code submitted for your consideration by your project partner. You should assume that `atomic_exchange()` works correctly (as described below), that the `gettid()` system call never "fails," that all queue functions behave in a strictly FIFO fashion, and that they will never fail. You should also assume "traditional x86-32 memory semantics," i.e., *not* "wacky modern memory."

```
/* queue.h */
int  q_init(queue_t *q);            // initializes a queue (to empty)
void q_enqueue(queue_t *q, int i);  // adds integer to queue at the end
int  q_dequeue(queue_t *q);         // removes+returns first integer in queue (else -1)
void q_destroy(queue_t *q);         // frees any resources used by queue
/* For the purposes of this question, assume queue functions cannot fail.  */

/* atomic.h */
/**
 * Atomically:
 * (1) fetches the old value of the memory location pointed to by "target"
 * (2) places the value "source" into the memory location pointed to by "target"
 * Then: returns the old value (that was atomically fetched)
 *
 * Equivalently:
 *  ATOMICALLY {
 *    int previous_target = *target;
 *    *target = source;
 *  }
 *  return previous_target;
 */
extern int atomic_exchange(int *target, int source);
```

The remainder of this page is intentionally blank.

```
typedef struct {
    int turn;
    int locked;
    int queue_locked;
    queue_t waiting;
} mutex_t;

int mutex_init(mutex_t *m) {
    m->turn = -1;
    m->locked = 0;
    m->queue_locked = 0;
    return q_init(&m->waiting);
}

void mutex_lock(mutex_t *m) {
    if (atomic_exchange(&m->locked, 1)) {
        // someone else has the lock
        int tid = gettid();
        while (atomic_exchange(&m->queue_locked, 1))
            continue;
        q_enqueue(&m->waiting, tid);
        m->queue_locked = 0;
        while (m->turn != tid)
            continue;
    }
}

void mutex_unlock(mutex_t *m) {
    int tid;
    while (atomic_exchange(&m->queue_locked, 1))
        continue;
    if ((tid = q_dequeue(&m->waiting)) > 0) {
        m->turn = tid;
    } else {
        m->locked = 0;
    }
    m->queue_locked = 0;
}
/* ... remainder omitted, e.g., mutex_destroy() ... */
```

There is a problem with the mutex code shown above. That is, it does not ensure that all three critical-section algorithm requirements are always met. Identify a requirement which is not met and lay out a scenario which demonstrates your claim. Use the format presented in class, i.e.,

T1	T2
tid = gettid();	
	continue;

...

Be sure that the execution trace you provide us with is easy to read and conclusively demonstrates the claim you are making. You may introduce temporary variables or other obvious notation as necessary to improve the clarity of your answer. *You should report a problem with code that is visible to you rather than assuming a problem in code that you have not been shown.* It is possible to answer this question with a brief, clear trace, so you should do what is necessary to ensure that you do. It is *strongly recommended* that you write down a draft version of any execution trace using the scrap paper provided at the end of the exam, or on the back of some other page, *before* you begin to write your solution on the next page. If we cannot understand the solution you provide, your grade will suffer!

This page is for your faulty-mutex solution.

This page can be used for your faulty-mutex solution.

3. 15 points Dining Philosophers

Consider the code below which implements a small "Dining Philosophers" system using standard concurrency primitives.

```
#include <stdlib.h>
#include <thread.h>
#include <mutex.h>
#include <cond.h>
#include <rand.h>
#include <stdio.h>
#include <syscall.h>

#define DINERS 3

/* "right hand rule" mapping diner numbers to left/right chopsticks
 *
 * diner 0's left chopstick is (0+2)%3==2, right chopstick is 0
 * diner 1's left chopstick is (1+2)%3==0, right chopstick is 1
 * diner 2's left chopstick is (2+2)%3==1, right chopstick is 2
 */
int left(int d)  { return ((d + (DINERS-1) % DINERS)); }
int right(int d) { return (d); }

mutex_t table;
int stick[DINERS];    /* stick -> -1 or owner */
cond_t avail[DINERS]; /* that stick is newly free */

mutex_t prng_lock;
unsigned long genrand_r(void);

void *diner(void *vsd);

int main()
{
    int i;

    sgenrand(get_ticks());
    thr_init(16*1024);
    mutex_init(&table);
    mutex_init(&prng_lock);
    /* "set the table": */
    for (i = 0; i < DINERS; ++i) {
        stick[i] = -1;
        cond_init(&avail[i]);
    }
    /* "Gentlemen, Be Seated!" (Robert A. Heinlein) */
    for (i = 0; i < DINERS; ++i) {
        thr_create(diner, (void *)i);
    }
    thr_exit(0);
    return(99); /* on the exam, we definitely won't get here */
}
```

```
void start_eating(int d)
{
    mutex_lock(&table);

    while (stick[right(d)] != -1) {
        cond_wait(&avail[right(d)], &table);
    }
    stick[right(d)] = d;

    while (stick[left(d)] != -1) {
        cond_wait(&avail[left(d)], &table);
    }
    stick[left(d)] = d;

    mutex_unlock(&table);
}

void done_eating(int d)
{
    mutex_lock(&table);
    stick[left(d)] = stick[right(d)] = -1;
    cond_signal(&avail[right(d)]);
    cond_signal(&avail[left(d)]);
    mutex_unlock(&table);
}

void *diner(void *vsd)
{
    int d = (int) vsd;

    while (1) {
        sleep(genrand_r() % 100);
        start_eating(d);
        sleep(genrand_r() % 10);
        done_eating(d);
    }
}

unsigned long genrand_r(void)
{
    unsigned long ul;
    mutex_lock(&prng_lock);
    ul = genrand();
    mutex_unlock(&prng_lock);
    return (ul);
}
```

(a) $\boxed{\text{10 points}}$ This system can deadlock. Show an execution trace which demonstrates one way a deadlock can arise. Use the format presented in class; abbreviations which are genuinely obvious are acceptable, e.g.,

D0	D1
	cwait(1)
lock(t)	
s[0]=0	

Grades will be assigned based on your ability to convince the grader that your execution trace is possible and results in a deadlock. Thus it is greatly to your advantage for your answer to be *easy to read, compelling, and concise.* You should almost certainly write your solution down on the back of the previous page (or somewhere else) and check it there before copying it to the space below.

You may use this page as space for your dining-philosophers deadlock if you wish.

(b) $\boxed{\text{5 points}}$ Describe an approach to changing the code above so that deadlock does not occur. We are *not* expecting you to provide replacement code, but the description of your change should be clear enough to convince your grader that your proposal is practical and based on understanding the relevant principles. A small amount of pseudo-code or code is fine if it is a good way to support your position, but pseudo-code or code without an explanation may not receive much credit.

You may use this page as space for your dining-philosophers solution if you wish.

4. $\boxed{\text{20 points}}$ Atomic matching.

In lecture we talked about two fundamental operations in concurrent programming: brief mutual exclusion for atomic sequences (provided in P2 by mutexes) and long-term voluntary descheduling (provided by condition variables). As you know, these can be combined to produce higher-level objects such as semaphores or readers/writers locks.

In this question you will implement a synchronization object called a "binder" that might be used by computational chemists to simulate combining atoms into molecules. Though a binder object could be general in the sense of combining various elements into molecules, for the purposes of this question you will implement a binder object that repeatedly combines two hydrogen atoms and one oxygen atom to create one water molecule. After a binder is initialized, a number of threads which is a multiple of three will invoke the `bind` operation. Each thread will specify whether it represents a hydrogen atom or an oxygen atom. The `bind` operation will suspend threads as necessary until it has control over at least two hydrogen atoms and one oxygen atom; then it will arrange for three threads (two hydrogen threads and one oxygen thread) to return. The return code from the `bind` operation will tell each thread whether it represented the critical third atom to form a molecule, or whether it represented one of the other two atoms. A binder object does not know how many threads will invoke it, though it can depend on the number being a multiple of three and that the ratio of atom types will be appropriate. Though a professional implementation of binder objects would include not only a way to initialize them but also a way to inactivate them, we will skip description and implementation of a `destroy` operation in this exam question.

A small example program using a binder is displayed on the next page.

The remainder of this page is intentionally blank.

```
// We are making H2O, which needs NHYD=2 of type=HYD, plus NOXY of type=OXY
#define HYD 0
#define NHYD 2
#define OXY 1
#define NOXY 1

#define STACK_SIZE 4096
#define MOLECULES 5

binder_t the_binder;

static void *atom(void *v_type)
{
    int type = (int) v_type;
    char *s = (type == OXY) ? "oxygen" : "hydrogen";

    int id = binder_bind(&the_binder, type);

    if (id >= 0)
         printf("%s %d formed molecule #%d!\n", s, thr_getid(), id);
    else
         printf("%s %d helped out\n", s, thr_getid());
    return (NULL);
}

int main(void)
{
    affirm(thr_init(STACK_SIZE) == 0);
    affirm(binder_init(&the_binder) == 0);

    for (int oxy = 0; oxy < MOLECULES; ++oxy)
        affirm(thr_create(atom, (void *)OXY) >= 0);

    for (int hyd = 0; hyd < MOLECULES; ++hyd) {
        affirm(thr_create(atom, (void *)HYD) >= 0);
        affirm(thr_create(atom, (void *)HYD) >= 0);
    }

    thr_exit(NULL);
    panic("I shouldn't be here.  Nobody should be here!");
    return -99;
}
```

Sample output from one run of the program is found below. Note that the order lines are printed in depends to some extent on the vagaries of the scheduler.

```
hydrogen 1086 formed molecule #0!
hydrogen 1085 helped out
oxygen 1080 helped out
hydrogen 1088 formed molecule #1!
hydrogen 1087 helped out
oxygen 1081 helped out
hydrogen 1090 formed molecule #2!
hydrogen 1089 helped out
oxygen 1082 helped out
hydrogen 1091 helped out
oxygen 1083 helped out
hydrogen 1092 formed molecule #3!
hydrogen 1093 helped out
hydrogen 1094 formed molecule #4!
oxygen 1084 helped out
```

Your task is to implement binders with the following interface:

- `int binder_init(binder_t *bp)` — initializes a binder. In the case of an error, this function will return an error code less than zero.
- `int binder_bind(binder_t *bp, int type)` —
  If a thread (which must be of type `HYD` or `OXY`) arrives before enough other threads of the appropriate types, the thread will block. When a binder has control over least two `HYD` threads and one `OXY` thread, "reasonably promptly" the binder object will simulate the creation of a single water molecule: the three relevant threads will return from from `binder_bind()`. The first two of those threads that had invoked the `bind` operation will receive a negative return code, and the third thread, whose arrival had enabled creation of the water molecule, will receive a non-negative return code which will uniquely identify the water molecule that was created. The molecule identification numbers generated by a single binder object will start from zero and increase in a counter fashion, so they would be suitable for indexing an array.
- `void binder_destroy(binder_t *bp)` — Deactivates a binder object. It is illegal for a program to invoke `binder_destroy()` if any threads are operating on it. Please note: though we are providing a description of the `destroy` operation, we will not be requiring you to implement it.

Assumptions:

1. You may use regular Project 2 thread-library primitives: mutexes, condition variables, semaphores, readers/writers locks, etc.
2. You may assume that callers of your routines will obey the rules. **But you must be careful that you obey the rules as well!**
3. You may *not* use other atomic or thread-synchronization synchronization operations, such as, but not limited to: `deschedule()`/`make_runnable()`, or any atomic instructions (`XCHG`, `LL/SC`).
4. You must comply with the published interfaces of synchronization primitives, i.e., you cannot inspect or modify the internals of any thread-library data objects.
5. You may not use assembly code, inline or otherwise.
6. **For the purposes of the exam, you may assume that library routines and system calls don't "fail"** (unless you indicate in your comments that you have arranged, and are expecting, a particular failure).
7. You may **not** rely on any data-structure libraries such as splay trees, red-black trees, queues, stacks, or skip lists, lock-free or otherwise, that you do not implement as part of your solution.
8. You may use non-synchronization-related thread-library routines in the "`thr_xxx()` family," e.g., `thr_getid()`. You may wish to refer to the "cheat sheets" at the end of the exam. If you wish, you may assume that `thr_getid()` is "very efficient" (for example, it invokes no system calls). You may also assume that condition variables are strictly FIFO if you wish.

It is strongly recommended that you rough out an implementation on the scrap paper provided at the end of the exam, or on the back of some other page, before you write anything on the next page. If we cannot understand the solution you provide, your grade will suffer!

(a) $\boxed{\text{5 points}}$ Please declare your `binder_t` here. If you need one (or more) auxilary structures, you may declare it/them here as well. Then please implement `binder_init()`.

```
typedef struct {











} binder_t;
```

[You may use this page for your binder declaration(s) and `init` if you wish.]

(b) $\boxed{\text{15 points}}$ Now please implement `binder_bind()`. It is not necessary to implement `binder_destroy()`.

. . . space for binder implementation . . .

. . . space for binder implementation . . .

[You may use this page for your binder implementation if you wish.]

5. $\boxed{\text{10 points}}$ Your high school friend Kelly comes to visit you near the beginning of 15-410 and attends a few lectures. Not having found the lecturer's style compelling, Kelly decides to get some revenge when the Project 1 tarball is issued. Here is Kelly's test kernel.

```
/** @file game.c
 *
 * @brief Try *this* "shortest legal fairy tale"!!!!!!!
 */
#include <p1kern.h>
#include <stdio.h>
#include <simics.h>
#include <multiboot.h>
#include <x86/asm.h>
#include <x86/interrupt_defines.h>

volatile static int __kernel_all_done = 0;

void tick(unsigned int numTicks)
{
    outb(INT_CTL_PORT,INT_ACK_CURRENT);
    kernel_main(NULL, 0, (char **)0, (char **)0);
}

int kernel_main(mbinfo_t *mbinfo, int argc, char **argv, char **envp)
{
    /*
     * Initialize device-driver library.
     */
    handler_install(tick);
    enable_interrupts();

    while (!__kernel_all_done) {
        continue;
    }

    return 0;
}
```

What happens when Kelly's test kernel is run, assuming a reasonable implementation of the Project 1 drivers? If you feel that multiple scenarios are possible, please pick one and focus on its details. You may wish to break your answer into several parts, such as "Assuming the ... is ...", "At first, ...", "After some time..." and/or "Eventually...".

Your answer doesn't have to *exactly* match the particular quirks of x86 hardware as long as you justify your response in terms of plausible hardware. That is, we're looking for a plausible understanding of the parts of the problem and how they might well fit together.

...Space for the Kelly's-kernel question...

You may use this page as extra space for the Kelly's-kernel question if you wish.

System-Call Cheat-Sheet

```
/* Life cycle */
int fork(void);
int exec(char *execname, char *argvec[]);
void set_status(int status);
void vanish(void) NORETURN;
int wait(int *status_ptr);
void task_vanish(int status) NORETURN;

/* Thread management */
int thread_fork(void); /* Prototype for exam reference, not for C calling!!! */
int gettid(void);
int yield(int pid);
int deschedule(int *flag);
int make_runnable(int pid);
int get_ticks();
int sleep(int ticks); /* 100 ticks/sec */
typedef void (*swexn_handler_t)(void *arg, ureg_t *ureg);
int swexn(void *esp3, swexn_handler_t eip, void *arg, ureg_t *newureg):

/* Memory management */
int new_pages(void * addr, int len);
int remove_pages(void * addr);

/* Console I/O */
char getchar(void);
int readline(int size, char *buf);
int print(int size, char *buf);
int set_term_color(int color);
int set_cursor_pos(int row, int col);
int get_cursor_pos(int *row, int *col);

/* Miscellaneous */
void halt();
int readfile(char *filename, char *buf, int count, int offset);

/* "Special" */
void misbehave(int mode);
```

If a particular exam question forbids the use of a system call or class of system calls, the presence of a particular call on this list does not mean it is “always ok to use.”

Thread-Library Cheat-Sheet

```
int mutex_init( mutex_t *mp );
void mutex_destroy( mutex_t *mp );
void mutex_lock( mutex_t *mp );
void mutex_unlock( mutex_t *mp );

int cond_init( cond_t *cv );
void cond_destroy( cond_t *cv );
void cond_wait( cond_t *cv, mutex_t *mp );
void cond_signal( cond_t *cv );
void cond_broadcast( cond_t *cv );

int thr_init( unsigned int size );
int thr_create( void *(*func)(void *), void *arg );
int thr_join( int tid, void **statusp );
void thr_exit( void *status );
int thr_getid( void );
int thr_yield( int tid );

int sem_init( sem_t *sem, int count );
void sem_wait( sem_t *sem );
void sem_signal( sem_t *sem );
void sem_destroy( sem_t *sem );

#define RWLOCK_READ  0
#define RWLOCK_WRITE 1
int rwlock_init( rwlock_t *rwlock );
void rwlock_lock( rwlock_t *rwlock, int type );
void rwlock_unlock( rwlock_t *rwlock );
void rwlock_destroy( rwlock_t *rwlock );
void rwlock_downgrade( rwlock_t *rwlock );
```

If a particular exam question forbids the use of a library routine or class of library routines, the presence of a particular routine on this list does not mean it is “always ok to use.”

Ureg Cheat-Sheet

```
#define SWEXN_CAUSE_DIVIDE       0x00  /* Very clever, Intel */
#define SWEXN_CAUSE_DEBUG        0x01
#define SWEXN_CAUSE_BREAKPOINT   0x03
#define SWEXN_CAUSE_OVERFLOW     0x04
#define SWEXN_CAUSE_BOUNDCHECK   0x05
#define SWEXN_CAUSE_OPCODE       0x06  /* SIGILL */
#define SWEXN_CAUSE_NOFPU        0x07  /* FPU missing/disabled/busy */
#define SWEXN_CAUSE_SEGFAULT     0x0B  /* segment not present */
#define SWEXN_CAUSE_STACKFAULT   0x0C  /* ouch */
#define SWEXN_CAUSE_PROTFAULT    0x0D  /* aka GPF */
#define SWEXN_CAUSE_PAGEFAULT    0x0E  /* cr2 is valid! */
#define SWEXN_CAUSE_FPUFAULT     0x10  /* old x87 FPU is angry */
#define SWEXN_CAUSE_ALIGNFAULT   0x11
#define SWEXN_CAUSE_SIMDFAULT    0x13  /* SSE/SSE2 FPU is angry */

#ifndef ASSEMBLER

typedef struct ureg_t {
    unsigned int cause;
    unsigned int cr2;   /* Or else zero. */

    unsigned int ds;
    unsigned int es;
    unsigned int fs;
    unsigned int gs;

    unsigned int edi;
    unsigned int esi;
    unsigned int ebp;
    unsigned int zero;  /* Dummy %esp, set to zero */
    unsigned int ebx;
    unsigned int edx;
    unsigned int ecx;
    unsigned int eax;

    unsigned int error_code;
    unsigned int eip;
    unsigned int cs;
    unsigned int eflags;
    unsigned int esp;
    unsigned int ss;
} ureg_t;

#endif /* ASSEMBLER */
```

Useful-Equation Cheat-Sheet

$$\cos^2\theta + \sin^2\theta = 1$$
$$\sin(\alpha \pm \beta) = \sin\alpha\cos\beta \pm \cos\alpha\sin\beta$$
$$\cos(\alpha \pm \beta) = \cos\alpha\cos\beta \mp \sin\alpha\sin\beta$$
$$\sin 2\theta = 2\sin\theta\cos\theta$$
$$\cos 2\theta = \cos^2\theta - \sin^2\theta$$
$$e^{ix} = \cos(x) + i\sin(x)$$
$$\cos(x) = \frac{e^{ix} + e^{-ix}}{2}$$
$$\sin(x) = \frac{e^{ix} - e^{-ix}}{2i}$$

$$\int \ln x\, dx = x\ln x - x + C$$
$$\int_0^\infty \sqrt{x}\, e^{-x}\, dx = \frac{1}{2}\sqrt{\pi}$$
$$\int_0^\infty e^{-ax^2}\, dx = \frac{1}{2}\sqrt{\frac{\pi}{a}}$$
$$\int_0^\infty x^2 e^{-ax^2}\, dx = \frac{1}{4}\sqrt{\frac{\pi}{a^3}} \text{ when } a > 0$$
$$\Gamma(z) = \int_0^\infty t^{z-1}e^{-t}\, dt$$

$$i\hbar\frac{\partial}{\partial t}\Psi(\mathbf{r},\, t) = \hat{H}\Psi(\mathbf{r}, t)$$
$$i\hbar\frac{\partial}{\partial t}\Psi(\mathbf{r},\, t) = -\frac{\hbar^2}{2m}\nabla^2\Psi(\mathbf{r},\, t) + V(\mathbf{r})\Psi(\mathbf{r},\, t)$$
$$E = hf = \frac{h}{2\pi}(2\pi f) = \hbar\omega$$
$$p = \frac{h}{\lambda} = \frac{h}{2\pi}\frac{2\pi}{\lambda} = \hbar k$$
$$\nabla \cdot \mathbf{E} = \frac{\rho}{\varepsilon_0}$$
$$\nabla \cdot \mathbf{B} = 0$$
$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t}$$
$$\nabla \times \mathbf{B} = \mu_0\mathbf{J} + \mu_0\varepsilon_0\frac{\partial \mathbf{E}}{\partial t}$$

If you wish, you may tear this page off and use it for scrap paper. But be sure not to write anything on this page which you want us to grade.

If you wish, you may tear this page off and use it for scrap paper. But be sure not to write anything on this page which you want us to grade.