Computer Science 15-410: Operating Systems
Mid-Term Exam (C), Spring 2010

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 *and also* put your Andrew ID at the top of *each* following page.
3. This is a closed-book in-class exam. You may not use any reference materials during the exam.
4. 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 ...?"
5. 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.
6. Please be concise in your answers. You will receive partial credit for partially correct answers, but truly extraneous remarks may count against your grade.
7. **Write legibly even if you must slow down to do so!** If you spend some time to *think clearly* about a problem, you will probably have time to write your answer legibly.

Andrew Username	
Full Name	

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

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

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

We are expecting each part of this question to be answered by three to six sentences. Your goal is to make it clear to your grader that you understand the concept *as it applies to this course* and can apply it when necessary.

(a) $\boxed{\text{5 points}}$ Explain how an "exception" and an "interrupt" are related. Provide an example of each one and discuss how the hardware treats them differently.

(b) $\boxed{\text{5 points}}$ What is a “segment register”? How is one used?

2. 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
 */
#define LSTICK(d) (((d)+(DINERS-1)) % DINERS)
#define RSTICK(d) (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[RSTICK(d)] != -1) {
        cond_wait(&avail[RSTICK(d)], &table);
    }
    stick[RSTICK(d)] = d;

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

    mutex_unlock(&table);
}

void done_eating(int d)
{
    mutex_lock(&table);
    stick[LSTICK(d)] = stick[RSTICK(d)] = -1;
    cond_signal(&avail[RSTICK(d)]);
    cond_signal(&avail[LSTICK(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);
}
```

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 extra space for your dining-philosophers solution if you wish.

3. $\boxed{\text{15 points}}$ SIGSEGV

Consider the following code which might be observed early (hopefully *very very* early!) in the development of a Project 0.

```
#include <stdlib.h>
#include <stdio.h>
#include <signal.h>

/* This code is compiled with -O0, i.e., with the optimizer turned off.
 * Also, we assume sizeof (void*) >= sizeof (long), which is ok for IA32
 * but not portable to other platforms.
 */

static volatile int broken = 0;  /* communicate from handler to checker */

/* This function sets the global "broken" flag to 1 on a seg fault.
 * The function causing the fault can then check the "broken" flag.
 */
void segv(int ignored) {
    broken = 1;
}

/* Check for validity of address. */
int tryit(void *addr) {
    volatile int buffer;
    broken = 0;
    buffer = *(int *)addr;
    return (!broken);
}

int main(int argc, char *argv[]) {
    void *addr;
    struct sigaction sa;

    sa.sa_handler = segv;
    sigfillset(&sa.sa_mask); /* don't want other signals while in segv() */
    sa.sa_flags = 0;
    sigaction(SIGSEGV, &sa, NULL);

    if (argc > 1) {
        addr = (void *) strtoul(argv[1], NULL, 16); /* convert from hex string */

        if (tryit(addr)) {
            printf("Ok: %p\n", addr); return 0;
        } else {
            printf("Not ok: %p\n", addr); return 1;
        }
    } else {
        printf("%s: no address specified\n", argv[0]); return 2;
    }
}
```

The author of the code appears to expect that when the program is run the address specified on the command line will be quickly diagnosed as either valid or invalid. But that's not what happens. For some addresses, the program appears to hang.

```
[de0u@keykos code]$ gcc -O0 -m32 -Wall -o tryit tryit.c
[de0u@keykos code]$ ./tryit 0x8048433
Ok: 0x8048433
[de0u@keykos code]$ ./tryit 0
...time passes...
...user gets angry and presses control-C...
^C
[de0u@keykos code]$
```

Below you may find the assembly code for the relevant C functions.

```
.globl segv
segv:
        pushl   %ebp
        movl    %esp, %ebp
        movl    $1, broken
        popl    %ebp
        ret
.globl tryit
tryit:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $16, %esp
        movl    $0, broken
        movl    8(%ebp), %eax
        movl    (%eax), %eax
        movl    %eax, -4(%ebp)
        movl    broken, %eax
        testl   %eax, %eax
        sete    %al
        movzbl  %al, %eax
        leave
        ret
```

(a) $\boxed{\text{10 points}}$ In the case where the program doesn't promptly print out a diagnosis for a given address, what is happening, and when? Is it really hung, or should the user wait longer? If so, how long?

Your answer should demonstrate that you understand and can apply the concepts and abstractions relevant to this class; answers which are "correct" but too abstract will receive only partial credit. We are expecting some answers to be roughly two paragraphs long; other answers may show a *well-commented* execution trace.

You may use this page as extra space for your SIGSEGV solution if you wish.

(b) $\boxed{\text{5 points}}$ Explain how to improve the performance of the program in the situation(s) you described in the previous part. Your proposal may include code but that is not mandatory; in either case please clearly demonstrate that you understand how your proposal changes the sequence of events you described above.

4. $\boxed{\text{15 points}}$ “Pair locks”

When you implemented your thread library for Project 2, you implemented several objects which provide locking: mutexes, semaphores, and reader/writer locks. Many other kinds of lock exist, and we will probably discuss some of them later in the semester.

In this question we will discuss a special kind of lock proposed for a special situation. Imagine a system in which many resources come in pairs (perhaps the system has a pair of tape drives and a pair of CD burners), and imagine further that processes, depending on their immediate needs, use either the first element of a pair, the second element, or both.

In an attempt to avoid “something bad” happening if processes used regular locks when trying to acquire some or both devices in a pair, a special lock called a “pair lock” has been devised.

When a pair lock is initialized, both resources managed by the pair lock pair are available. When `pl_lock()` is called, the caller specifies whether to lock the first resource, the second resource, or both—`pl_lock()` returns when all requested elements have been acquired. `pl_unlock()` releases the specified elements of a resource pair, which must currently be held by the caller. When a thread calls `pl_lock()` on a pair lock, it must use `pl_unlock()` to release all elements it holds before it any later acquisition of elements from the pair using `pl_lock()`.

For the purposes of the exam you should assume an error-free environment (invocations of `pl_lock()` and `pl_unlock()` are always legal; memory allocation will always succeed; thread-library primitives will not detect internal inconsistencies or otherwise “fail,” etc.). If you wish, you may assume that the mutexes provided by the underlying thread library provide bounded waiting; you may also assume, if you wish, that the underlying condition variables are “as FIFO as possible.” You may wish to refer to the “cheat sheets” at the end of the exam.

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

typedef struct pl {
    mutex_t m;
    cond_t released;
    int avail[2];
} pl_t;

/* Initialize a pair-lock */
void pl_init(pl_t *pl)
{
    mutex_init(&pl->m);
    cond_init(&pl->released);
    pl->avail[0] = pl->avail[1] = 1;
}
```

```
/* Lock one or both resources controlled by the pair-lock.  */
void pl_lock(pl_t *pl, int want[2])
{
    mutex_lock(&pl->m);

    while ((want[0] && !pl->avail[0]) || (want[1] && !pl->avail[1])) {
        cond_wait(&pl->released, &pl->m);
    }

    /* good to go! */
    if (want[0]) {
        pl->avail[0] = 0;
    }
    if (want[1]) {
        pl->avail[1] = 0;
    }
    mutex_unlock(&pl->m);
    return;
}

/* Unlock resources controlled by a pair-lock.
 * The results are undefined if the resources we are
 * directed to unlock are not currently locked by us.
 */
void pl_unlock(pl_t *pl, int unlock[2])
{
    mutex_lock(&pl->m);
    if (unlock[0]) {
        pl->avail[0] = 1;
    }
    if (unlock[1]) {
        pl->avail[1] = 1;
    }
    cond_signal(&pl->released);
    mutex_unlock(&pl->m);
}
```

Unfortunately, this implementation of pair locks does not ensure progress: an unbounded time may pass during which one or more threads are requesting one or more *available* resources from a pair lock but no requests are granted.

(a) $\boxed{\text{8 points}}$ Provide an execution trace using the format presented in class (as described in the previous problem) which clearly demonstrates this kind of progress failure. Grades will be assigned based on your ability to convince the grader that your execution trace is possible and demonstrative. 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 extra space for your pair-lock solution if you wish.

(b) $\boxed{\text{7 points}}$ Explain how to solve the problem you identified. If you completely understand the problem, your answer to this part can be very short: a sentence explaining a particular code change and a short paragraph explaining why it works. Alternatively, you may present a `struct pl_t` and code for `pl_init()`, `pl_lock()` and `pl_unlock()`; if you take that approach, be sure to briefly summarize how your solution works.

You may use this page as extra space for your pair-lock solution if you wish.

5. 15 points `print_ints()`.

Imagine that in your "spare time" (whatever *that* might be), you begin working on a small embedded operating system for a very small, power-constrained device which has the ability to shut down individual parts of its RAM to save power (the contents are lost). Naturally, you find you need some way to emit debugging output. However, you are shocked to learn that a standard implementation of `printf()` compiles to more than *four kilobytes* of code space; your power budget doesn't allow you to waste that much RAM just for pretty debugging messages. You decide to investigate a simpler, hopefully smaller debugging facility and come up with the following.

`print_ints(int count, ...)` - print a list of integer parameters; the number of integers to print is specified as the first parameter to the function, and the other integers are passed as the remaining parameters. The printed integers are separated by spaces and after all the integers are printed a newline character is printed as well.

Here are some sample invocations.

- `print_ints(1,42)` - will print just 42 followed by a newline.
- `print_ints(3,1,2,3)` - will print the smallest three positive integers followed by a newline.
- `print_ints(3,1,2,3,42)` - will also print the smallest three positive integers followed by a newline (note the invocation error).
- `print_ints(1048576,1)` - the results are undefined (not enough ints!).

You decide to prototype this first on a system you are familiar with, namely IA-32 (32-bit Intel x86), using gcc as your compiler, gas as your assembler, and GNU ld as your linker.

Now please write the code for `print_ints()`. You may make use of two simple underlying functions: `print_int(int i)` prints a single integer and `print_char(char c)` prints a single character. You should not need to make calls to any functions or macros other than those, plus ones you write. Most of your code should be in C, though you may use a small amount of assembly code if you wish. Please make sure your code is easy to read, clearly accomplishes its goals, and is at least lightly commented when necessary (don't worry—you don't need to provide any Doxygen comments; a *few* informal but helpful comments should suffice).

You may use this page as extra space for your `print_ints()` solution 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 cas2i_runflag(int tid, int *oldp, int ev1, int nv1, int ev2, int nv2);
int get_ticks();
int sleep(int ticks); /* 100 ticks/sec */

/* 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 ls(int size, char *buf);

/* "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 );
int mutex_destroy( mutex_t *mp );
int mutex_lock( mutex_t *mp );
int mutex_unlock( mutex_t *mp );

int cond_init( cond_t *cv );
int cond_destroy( cond_t *cv );
int cond_wait( cond_t *cv, mutex_t *mp );
int cond_signal( cond_t *cv );
int 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 );
int sem_wait( sem_t *sem );
int sem_signal( sem_t *sem );
int sem_destroy( sem_t *sem );

int rwlock_init( rwlock_t *rwlock );
int rwlock_lock( rwlock_t *rwlock, int type );
int rwlock_unlock( rwlock_t *rwlock );
int rwlock_destroy( 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."

Useful-Equation Cheat-Sheet

$$\cos^2\theta + \sin^2\theta = 1$$

$$\sin(\alpha \pm \beta) = \sin\alpha\cos\beta \pm \cos\alpha\sin\beta$$

$$\cos(\alpha \pm \beta) = \cos\alpha\cos\beta \mp \sin\alpha\sin\beta$$

$$\sin 2\theta = 2\sin\theta\cos\theta$$

$$\cos 2\theta = \cos^2\theta - \sin^2\theta$$

$$e^{ix} = \cos(x) + i\sin(x)$$

$$\cos(x) = \frac{e^{ix} + e^{-ix}}{2}$$

$$\sin(x) = \frac{e^{ix} - e^{-ix}}{2i}$$

$$\int \ln x\, dx = x\ln x - x + C$$

$$\int_0^\infty \sqrt{x}\, e^{-x}\, dx = \frac{1}{2}\sqrt{\pi}$$

$$\int_0^\infty e^{-ax^2}\, dx = \frac{1}{2}\sqrt{\frac{\pi}{a}}$$

$$\int_0^\infty x^2 e^{-ax^2}\, dx = \frac{1}{4}\sqrt{\frac{\pi}{a^3}} \text{ when } a > 0$$

$$\Gamma(z) = \int_0^\infty t^{z-1}e^{-t}\, dt$$

$$i\hbar\frac{\partial}{\partial t}\Psi(\mathbf{r},\, t) = \hat{H}\Psi(\mathbf{r}, t)$$

$$i\hbar\frac{\partial}{\partial t}\Psi(\mathbf{r},\, t) = -\frac{\hbar^2}{2m}\nabla^2\Psi(\mathbf{r},\, t) + V(\mathbf{r})\Psi(\mathbf{r},\, t)$$

$$E = hf = \frac{h}{2\pi}(2\pi f) = \hbar\omega$$

$$p = \frac{h}{\lambda} = \frac{h}{2\pi}\frac{2\pi}{\lambda} = \hbar k$$

$$\nabla \cdot \mathbf{E} = \frac{\rho}{\varepsilon_0}$$

$$\nabla \cdot \mathbf{B} = 0$$

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t}$$

$$\nabla \times \mathbf{B} = \mu_0\mathbf{J} + \mu_0\varepsilon_0\frac{\partial \mathbf{E}}{\partial t}$$