Computer Science 15-410/15-605: Operating Systems
Mid-Term Exam (A), Fall 2015

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.	**20**		
4.	**15**		
5.	**10**		
	70		

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

Andrew ID: _____

I have not received advance information on the content of this 15-410 mid-term exam by discussing it with anybody who took part in the main exam session or via any other avenue.

Signature: _____ Date _____

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.

Andrew ID: _____

1. [10 points] Short answer.

   (a) [5 points] Explain what #include files should and should not be used for. We are expecting roughly five bulleted points (e.g., two things that #include files *should* contain and three things they should *not* contain, because those things should be somewhere else). Provide a brief justification for each of your claims.

(b) [5 points] Define "race condition" as the term is applied in this course. What are the necessary ingredients of a race condition?

2. [15 points] Deadlock.

Recently there was some trouble at the Technical Opportunities Conference (TOC). Some booths have a simple procedure involving waiting in line to submit a résumé and chat briefly with a recruiter. But one company decided to add some frills. In particular, they got an extra-large space containing a table full of iPads and a prize machine. Students line up in front of the table to be processed through the interview sequence; recruiters are located on the far side of the table. From the perspective of a student, these are the steps:

1. Wait in line until you are first in line; claim the table.
2. Claim an iPad from the table and fill out a form about yourself.
3. Release the iPad (but continue "owning" the table).
4. Claim a recruiter (wait until one isn't talking to a student who arrived earlier).
5. Release the table and talk to the recruiter you claimed for a random amount of time.
6. Once the recruiter is done talking to you, you must fill out a brief post-chat survey on an iPad to be eligible for a prize. The recruiter who chatted with you must enter a validation code on the iPad, so you will continue "owning" the recruiter while you claim an iPad from the table.
7. Once the recruiter enters the validation code onto the iPad you are holding, you and the recruiter are done, so you release the recruiter. The iPad is now displaying a QR code which is your prize entry ticket. Claim the table, and show the QR code on the iPad screen to the prize machine; at that point you can release the iPad and the table and leave with your prize.

This system is designed to work for varying numbers of students (the students obediently queue up on the outside of the table area until they can allocate the table), varying numbers of recruiters, and varying numbers of iPads. This is shown in the simulation code on the next page: the command-line parameters to `main()` set up varying numbers of the (non-student) recruiting resources and then run some number of student threads. But sometimes this system "gets stuck." A nearby Machine Learning student offers to apply the latest "deep learning with boosting" technology to figure out what is going wrong, but your job will be to apply boring old Computer Science principles to reason through the problem.

The remainder of this page is intentionally blank.

```
sem_t table;
sem_t recruiter;
sem_t ipad;

void *run_student(void *ignored);

int main(int argc, char** argv)
{
    int R, I, S;

    thr_init(4096);
    sgenrand(4096);

    if (argc != 4) {
        printf("Usage ./a.out n_recruiters n_ipads n_students\n");
        exit(1);
    }

    R = (int)strtol(argv[1], NULL, 10);  // convert string to integer
    I = (int)strtol(argv[2], NULL, 10);
    S = (int)strtol(argv[3], NULL, 10);

    sem_init(&recruiter,R);
    sem_init(&ipad,I);
    sem_init(&table,1);

    int *students = malloc(S * sizeof (int));

    int s;
    for (s = 0; s < S; s++) {
        students[s] = thr_create(run_student, (void *) s);
    }

    for (s = 0; s < S; s++) {
        thr_join(students[s], NULL);
    }
    printf("\nDone\n");
    exit(0);
}
```

```
void *run_student(void *ignored)
{
    sem_wait(&table);
    sem_wait(&ipad);
    sleep(genrand()% 300); // fill out form
    sem_signal(&ipad);

    sem_wait(&recruiter); // this could take a while
    sem_signal(&table); // next student can fill out form

    sleep(genrand() % 3600); // chat with recruiter for "a while"

    sem_wait(&ipad);
    sleep(genrand() % 100); // get validation from recruiter
    sem_signal(&recruiter); // recruiter can see next student

    sem_wait(&table);
    sleep(10); // show QR code, get prize
    sem_signal(&ipad);
    sem_signal(&table);

    return (0);
}
```

(a) [10 points] Unfortunately, the code shown above can deadlock. Show *clear, convincing* evidence of deadlock. Begin by *describing the problem* in one or two sentences; then *specify a scenario*, and finally *show a a "tabular execution trace"*. Missing, unclear, or unconvincing traces will result in only partial credit.

You may use this page for your deadlock trace if you wish.

(b) [5 points] Explain how the company can survive the TOC without embarrassing deadlocks. If possible, explain how the company can prevent deadlocks *without changing any of the rules or code shown above*, though you may suggest rule/code changes if you must. Be sure to explain (in a theoretical/conceptual sense) why your solution works. Solutions judged as higher-quality by your grader will receive more points.

3. [20 points] LIFO Condition Variables

In your Project 2 thread library, you implemented various standard synchronization primitives such as mutexes, condition variables, sempahores, and reader/writer locks. Often times, we stressed the importance of ensuring these primitives are "as FIFO as possible" in order to prevent starvation of any particular thread.

While "as FIFO as possible" is a good general policy, it may not be the best for all situations. Consider a situation where a work queue is being served by a pool of worker threads. If the system has been set up with lots of worker threads, there may not be enough work for them. Once a particular worker thread has been blocked for a while awaiting work, its "personal memory" (such as its stack) may fall out of the L1, L2, and maybe even L3 caches. In such a situation it arguably makes sense to give the next available work item to the thread that has been blocked for the *least* amount of time, because that thread will be able to run faster than the longest-blocked thread. Note that no harm is done to a thread that is blocked for a long time—it isn't "starved," it's "on vacation."

In this question you will be asked to forget everything you've been told about the importance of FIFO condition variables and implement a new synchronization primitive, the lifo_cond, which operates in a strictly "last-in, first-out" ("LIFO") manner. That is to say, the first thread to wait on a given lifo_cond should be the last thread to wake up.

Your mission: You will implement `lifo_cond`s with the following interface:

- `int lifo_cond_init(lifo_cond_t* cond)` - Initializes a `lifo_cond`.
- `void lifo_cond_wait(lifo_cond_t* cond, mutex_t* m)` - Waits on the `lifo_cond` until a thread calls `lifo_cond_signal` or `lifo_cond_broadcast` and all other threads that have waited on the `lifo_cond` since this calling thread was put to sleep have already been awakened. Otherwise, this call behaves as `cond_wait`.
- `void lifo_cond_signal(lifo_cond_t* cond)` - Awakens the most recent thread to have waited on the `lifo_cond`—or does nothing if no threads are currently waiting.
- `void lifo_cond_broadcast(lifo_cond_t* cond)` - Wakes all threads currently waiting on the `lifo_cond`—in such a way that the threads waiting the longest will generally run later than the threads waiting for less time.
- `void lifo_cond_destroy(lifo_cond_t* cond)` - Destroys the `lifo_cond`.

A minimal usage example follows on the next page.

The remainder of this page is intentionally blank.

```
#define NTHREADS 100
#define NWORK 4200

mutex_t work_mutex;
lifo_cond_t work_ready;

void threadfn(void *ignored) {
        mutex_lock(&work_mutex);
        while (work_left()) {
                lifo_cond_wait(&work_ready, &work_mutex);
                do_work();
        }
        mutex_unlock(&work_mutex);
}

int main(int argc, char **argv) {

        thr_init(16384);
        mutex_init(&work_mutex);
        lifo_cond_init(&work_ready);

        for (int t = 0; t < NTHREADS; t++) {
                thr_create(threadfn, NULL);
        }

        for (int i = 0; i < NWORK; i++) {
                add_work();
                lifo_cond_signal(&work_ready);
        }
        thr_exit(0);
}
```

Assumptions:

1. You may use regular Project 2 thread-library primitives: mutexes, condition variables, semaphores, reader/writer locks, etc., *but you must assume they operate in **strictly** FIFO order.*
2. You may assume that callers of your lifo_cond routines will obey the rules—for example, nobody will try to destroy a lifo_cond while threads might be blocked on it. **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).

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) [5 points] Please declare your `lifo_cond_t` here. If you need one (or more) auxilary structures, you may declare it/them here as well.

```
typedef struct {

} lifo_cond_t;
```

(b) [15 points] Now please implement `lifo_cond_init()`, `lifo_cond_wait()`, `lifo_cond_signal()`, `lifo_cond_broadcast()`, and `lifo_cond_destroy()`.

. . . space for lifo_cond implementation . . .

. . . space for lifo_cond implementation . . .

4. [15 points] "Go Ahead" continued...

Consider the following critical-section protocol, which is an attempt to remedy a problem uncovered in the Homework 1 "Go Ahead" critical-section protocol, which in turn was based on the "registering interest" approach presented in our "Synchronization #1" lecture.

```
       int think[2] = { 0, 0 };
       int want[2] = { 0, 0 };
       int goahead[2] = { 0, 0 };

 1.    do {
 2.       ...remainder section...
 3.       think[i] = 1;
 4.       want[i] = 1;
 5.       if (want[j]) {
 6.         goahead[j] = 1;
 7.       }
 8.       think[i] = 0;
 9.       while (think[j])
10.         continue;
11.       if (i == 0) // tie breaker
12.         goahead[i] = 1 - goahead[j];
13.       while (want[j] && !goahead[i])
14.         continue;
15.       ...begin critical section...
16.       ...end critical section...
17.       want[i] = 0;
18.       goahead[i] = 0;
19.   } while (1);
```

(This protocol is presented in "standard form," i.e., if thread 0 is running this code, $i == 0$ and $j == 1$; if thread 1 is running this code, $i == 1$ and $j == 0$.)

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.,

T0	T1
ga[0]=0;	
	ga[1]=0;

...

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!

You may use this page for your "Go Ahead" solution if you wish.

5. [10 points] `Nuts & Bolts`.

As you and your partner are getting started on your kernel project, you are examining each other's Project 1 driver implementations. Below is the code for your partner's assembly-language wrapper for the timer interrupt.

```
.globl timer_handler_c // extern void timer_handler_c(unsigned int ticks);

.text
.globl timer_handler
timer_handler:
  // stack discipline
  pushl %ebp
  movl %esp,%ebp
  // we must save callee-save registers
  // save caller-save also just to be safe
  pushl %ebx
  pushl %esi
  pushl %edi
  pushl %eax
  pushl %ecx
  // now do the work
  movl ticks,%eax
  pushl %eax
  call timer_handler_c
  popl %eax
  incl ticks
  // restore what we saved
  popl %ecx
  popl %eax
  popl %edi
  popl %esi
  popl %ebx
  popl %ebp
  iret

.data
// unsigned long ticks = 0;
ticks:
  .align 4
  .long 0x00000000
```

(a) [4 points] When you see your partner's `timer_handler` code, you realize that something is horribly wrong with it. What is that code doing wrong? Be as specific as you can.

(b) [6 points] Your partner angrily responds that the code must be fine—despite lots of testing the game never crashed. You explain that the `timer_handler` code, while absolutely wrong, will in practice cause trouble only very rarely. Why is that? Note that more-convincing answers will receive more credit.

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 );

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."

Andrew ID: _____

Typing Rules Cheat-Sheet

$$\begin{array}{rcl} \tau & ::= & \alpha \mid \tau \to \tau \mid \mu\alpha.\tau \mid \forall\alpha.\tau \\ e & ::= & x \mid \lambda x{:}\tau.e \mid e\,e \mid \mathsf{fix}(x{:}\tau.e) \mid \mathsf{fold}_{\alpha.\tau}(e) \mid \mathsf{unfold}(e) \mid \Lambda\alpha.e \mid e[\tau] \end{array}$$

$$\frac{}{\Gamma,\, \alpha\,\mathbf{type} \vdash \alpha\,\mathbf{type}}\ \text{istyp-var} \qquad \frac{\Gamma \vdash \tau_1\,\mathbf{type} \quad \Gamma \vdash \tau_2\,\mathbf{type}}{\Gamma \vdash t_1 \to t_2\,\mathbf{type}}\ \text{istyp-arrow}$$

$$\frac{\Gamma,\, \alpha\,\mathbf{type} \vdash \tau\,\mathbf{type}}{\Gamma \vdash \mu\alpha.\tau\,\mathbf{type}}\ \text{istyp-rec} \qquad \frac{\Gamma,\, \alpha\,\mathbf{type} \vdash \tau\,\mathbf{type}}{\Gamma \vdash \forall\alpha.\tau\,\mathbf{type}}\ \text{istyp-forall}$$

$$\frac{}{\Gamma,\, x : \tau \vdash x : \tau}\ \text{typ-var} \qquad \frac{\Gamma,\, x : \tau_1 \vdash e : \tau_2 \quad \Gamma \vdash \tau_1\,\mathbf{type}}{\Gamma \vdash \lambda x{:}\tau_1.e : \tau_1 \to \tau_2}\ \text{typ-lam} \qquad \frac{\Gamma \vdash e_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1\,e_2 : \tau_2}\ \text{typ-app}$$

$$\frac{\Gamma,\, x : \tau \vdash e : \tau \quad \Gamma \vdash \tau\,\mathbf{type}}{\Gamma \vdash \mathsf{fix}(x{:}\tau.e) : \tau}\ \text{typ-fix}$$

$$\frac{\Gamma \vdash e : [\mu\alpha.\tau/\alpha]\tau \quad \Gamma,\, \alpha\,\mathbf{type} \vdash \tau\,\mathbf{type}}{\Gamma \vdash \mathsf{fold}_{\alpha.\tau}(e) : \mu\alpha.\tau}\ \text{typ-fold} \qquad \frac{\Gamma \vdash e : \mu\alpha.\tau}{\Gamma \vdash \mathsf{unfold}(e) : [\mu\alpha.\tau/\alpha]\tau}\ \text{typ-unfold}$$

$$\frac{\Gamma,\, \alpha\,\mathbf{type} \vdash e : \tau}{\Gamma \vdash \Lambda\alpha.e : \forall\alpha.\tau}\ \text{typ-tlam} \qquad \frac{\Gamma \vdash e : \forall\alpha.\tau \quad \Gamma \vdash \tau'\,\mathbf{type}}{\Gamma \vdash e[\tau'] : [\tau'/\alpha]\tau}\ \text{typ-tapp}$$

$$\frac{}{\lambda x{:}\tau.e\,\mathbf{value}}\ \text{val-lam} \qquad \frac{}{\mathsf{fold}_{\alpha.\tau}(e)\,\mathbf{value}}\ \text{val-fold} \qquad \frac{}{\Lambda\alpha.\tau\,\mathbf{value}}\ \text{val-tlam}$$

$$\frac{e_1 \;\mapsto\; e_1'}{e_1\,e_2 \;\mapsto\; e_1'\,e_2}\ \text{steps-app}_1 \qquad \frac{e_1\,\mathbf{value} \quad e_2 \;\mapsto\; e_2'}{e_1\,e_2 \;\mapsto\; e_1\,e_2'}\ \text{steps-app}_2$$

$$\frac{e_2\,\mathbf{value}}{(\lambda x{:}\tau.e_1)\,e_2 \;\mapsto\; [e_2/x]e_1}\ \text{steps-app-}\beta$$

$$\frac{}{\mathsf{fix}(x{:}\tau.e) \;\mapsto\; [\mathsf{fix}(x{:}\tau.e)/x]e}\ \text{steps-fix}$$

$$\frac{e \;\mapsto\; e'}{\mathsf{unfold}(e) \;\mapsto\; \mathsf{unfold}(e')}\ \text{steps-unfold}_1 \qquad \frac{}{\mathsf{unfold}(\mathsf{fold}_{\alpha.\tau}(e)) \;\mapsto\; e}\ \text{steps-unfold}_2$$

$$\frac{e \;\mapsto\; e'}{e[\tau] \;\mapsto\; e'[\tau]}\ \text{steps-tapp}_1 \qquad \frac{}{(\Lambda\alpha.e)[\tau] \;\mapsto\; [\tau/\alpha]e}\ \text{steps-tapp}_1$$

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.