Computer Science 15-410/15-605: Operating Systems
Mid-Term Exam (A), Spring 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.	**20**		
4.	**20**		
5.	**5**		
	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) $\boxed{\text{6 points}}$ According to the 15-410 orthodoxy, there are three "mutex assumptions," meaning conditions that should apply when a mutex is used and thus expectations that a mutex implementor should consider. Note that these three "mutex assumptions" do *not* necessarily apply to all synchronization objects. For each one, briefly state the assumption *and why it matters to mutex implementors.* (As an emergency backup answer, if you can think of three other things about mutexes then you may list them for partial credit.)

(b) $\boxed{\text{4 points}}$ Register dump.

Below is a register dump produced by the "Pathos" P2 reference kernel when it decided to kill a user-space thread. Your job is to carefully consider the register dump and:

1. Determine which "wrong register value(s)" caused the thread to run an instruction that resulted in a fatal exception. You should say why/how the wrong value led to an exception, i.e., merely claiming a register has a "wrong" value will not receive full credit.
2. Briefly state the most plausible way you think that register could have taken on that value (i.e., try to describe a bug that could have this effect).
3. Then write a *small* piece of code that would plausibly cause the thread to die in the fashion indicated by the register dump. *This code does not need to implement exactly the set of steps that you identified as "most plausible" above, or result in the same register values; you should aim to achieve "basically the same effect."* Most answers will probably be in assembly language, but C is acceptable as well. Your code should assume execution begins in `main()`, which has been passed the typical two parameters in the typical fashion.

Please be sure that your description of the fatality and the code, taken together, clearly support your diagnosis.

```
Registers:
eax: 0xffffd000, ebx: 0x0100707c, ecx: 0x00000001,
edx: 0x01000154, edi: 0x00000000, esi: 0x00000000,
ebp: 0xffffefa4, esp: 0xffffefa8, eip: 0xffffefc8,
ss:     0x002b,  cs:     0x0023,  ds:     0x002b,
es:     0x002b,  fs:     0x002b,  gs:     0x002b,
eflags: 0x00000246
```

You may use this page for the register-dump question.

2. $\boxed{\text{15 points}}$ Going out of business.

In the "Paradise Lost" lecture we discussed a multi-threaded application in which worker threads were searching for a solution but sometimes, due to a specific kind of error, some threads suddenly vanished before the answer was found. We discussed a problematic implementation of a function called `find_work()` and improved its operation. The lecture slides encouraged students to find and fix another bug in `find_work()`. The code shown below applies a fix to `find_work()` and also shows code for `perform_work()`, which you have likely not seen before.

```
queue_t queue, *q = &queue;
mutex m;
cond_t new_work;
int going_out_of_business = 0;
```

```
void worker(void *ignored)
{
  workitem *work;
  while (work = find_work())
      perform(work);
  thr_exit((void *) 0);
}
```

```
workitem *find_work(void)
{
  workitem *w = (workitem *) NULL;
  mutex_lock(&m);
  while (!going_out_of_business && !(w = (workitem *) dequeue(q)))
      cond_wait(&new_work, &m);
  mutex_unlock(&m);
  return (w);
}

void perform(workitem *w)
{
  workitem *new;
  switch (evaluate(w, &new)) {
  case SOLUTION_NODE:
    going_out_of_business = 1;
    cond_signal(&new_work);
    printf("The answer is %s!!!!!\n", w->string);
    free(w);
    return;
  case TERMINAL_NODE:
    free(w);
    return;
  case REPLACEMENT_NODE:
    mutex_lock(&m);
    enqueue(q, new);
    mutex_unlock(&m);
    cond_signal(&new_work);
    free(w);
    return;
  default:
    panic("bad evaluation!");
  }
}
```

The code shown above has (at least) two concurrency bugs that can result in one or more threads getting "stuck." You will be asked to show a trace for one bug, then describe and fix that bug, then describe and fix a second bug (a trace will not be required). In each case we believe a fix is very straightforward, i.e., changing one to three lines of code. **Please note: though we are not showing the code for `evaluate()`, the two bugs are not in `evaluate()` and are not caused by or related to `evaluate()`.**

(a) $\boxed{\text{7 points}}$ Present a tabular execution trace that demonstrates *clearly* one way that the above code can plausibly result in one or more threads getting "stuck."

You may use this page as extra space for the trace if you wish.

(b) $\boxed{\text{4 points}}$ Briefly describe what's wrong with the code that results in the problem you traced above, and briefly describe how to fix that bug.

(c) $\boxed{\text{4 points}}$ Briefly describe what's wrong with the code that results in a *different* "stuck" scenario and briefly describe how to fix that bug. Showing a trace is not necessary if your description is clear.

3. $\boxed{\text{20 points}}$ Kitchen robots.

You are hired to oversee the kitchen at a brand-new restaurant. However, the chefs at this restaurant are a little unique... they are all robots. The basic functioning of the kitchen is as follows:

- There are N shared stations in the kitchen; at each station, one step of a recipe for some dish can take place. For example, a step might be slicing, frying, simmering, warming, etc. So there would be a slicing station, a frying station, a simmering station, a warming station, etc.
- Each station can be occupied by only one robot at a time.
- There are M recipes, one for each different dish that a customer can order.
- A recipe is represented as an ordered list of the stations used to prepare the dish. For example, `[1, 0, 2]` would be a recipe where the robot would first need to use station 1, then use station 0, then use station 2, at which point the dish would be fully prepared and the robot would serve it to the customer.
- All recipes have positive (non-zero) length.
- All recipes will be free from consecutive duplicates, so `[2, 2, 1]` would not be a valid recipe since there is a consecutive duplicate 2 (ruling out consecutive duplicates is fine since the robot would be staying at the same station anyway).
- The protocol that the robots use to avoid races when using the shared stations is shown immediately below, for recipe `r[]` of length n:

```
void protocol(int r[], int n) {
    acquire station r[0];
    perform all needed tasks needed at station r[0];
    for (int i = 1; i < n; i++) {
      acquire station r[i];
      move all materials from station r[i-1] to r[i];
      release station r[i-1]
      perform all needed tasks at station r[i];
    }
    release station r[n-1];
}
```

The remainder of this page is intentionally blank.

(a) $\boxed{\text{1 point}}$ Before your first night of work, the owner of the restaurant tells you how the previous kitchen supervisor handled cases of suspected deadlock. If the supervisor observed that some group of robots in the kitchen were done performing the tasks at their current stations, but not moving from one station to another "for a while," the supervisor would power down those robots, throw away their partially-cooked dishes, and then power the robots back on. At this point each robot would restart its current recipe from the start, and the supervisor would fervently hope that robots wouldn't "get stuck for a while" again. What is the name for this approach to handling deadlock?

(b) $\boxed{\text{4 points}}$ After hearing of the previous way of handling suspected deadlocks, you want to first show that deadlocks can actually occur. For the most powerful demonstration, provide a concrete example using just *one single recipe*. Provide a convincing trace of the station acquisition protocol show above that demonstrates the proposed deadlock. (If you can't provide a single-recipe trace, you can receive partial credit by providing a trace using multiple recipes.)

[Use this page for your deadlock trace if you wish.]

(c) $\boxed{\text{2 points}}$ Now that you have proof that deadlocks are possible, you try to figure out a *deadlock prevention* approach to solve our deadlock problems. Describe a “law” banning some recipes, which could be declared before the restaurant opens, that will ensure that no deadlocks can occur. Which standard deadlock ingredient would be outlawed? Does this law forbid the single recipe you described in the previous part? Briefly explain.

(d) $\boxed{\text{6 points}}$ Now consider a *deadlock avoidance* approach in which each robot requests permission to start a specific recipe before beginning the protocol, and announces the completion of the recipe when it's done. Describe how the kitchen supervisor should react to each recipe-start request and each recipe-done announcement. More points will be assigned to approaches which allow some situations in which multiple robots are working in the kitchen at the same time despite their recipes using some stations in common.

[Use this page for your avoidance answer if you wish.]

(e) $\boxed{\text{5 points}}$ Draw a design matrix to compare your deadlock prevention approach to your deadlock avoidance approach. Make sure your analysis pertains to the kitchen scenario instead of merely generic deadlock prevention and avoidance. Make sure your analysis includes at least three characteristics.

(f) $\boxed{\text{1 point}}$ Give a scenario where your deadlock prevention approach would be better than your deadlock avoidance approach. Support why deadlock prevention would be better using your design matrix.

(g) $\boxed{\text{1 point}}$ Give a scenario where your deadlock avoidance approach would be better than your deadlock prevention approach. Support why deadlock avoidance would be better using your design matrix.

4. $\boxed{\text{20 points}}$ Trio 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 "trio matcher." The idea is that some parallel tasks must be worked on by groups of three threads, and the threads need some (dynamic) way to pick partners to work with. After a trio matcher is initialized, a number of threads which is a multiple of three will invoke the `match` operation. The `match` operation involves some amount of thread synchronization, potentially including blocking, and then returns to each thread, in a reasonably timely fashion, the thread identification numbers of the threads it has been matched with. A match object does not know how many threads will invoke it, though it can depend on the number being a multiple of three. Once a program is sure that no more threads will invoke the `match` operation on a particular object, the `destroy` operation can and should be invoked.

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

The remainder of this page is intentionally blank.

```
#define NTHREADS 33
int tids[NTHREADS];
tmatch_t matcher;
void *threadbody(void *ignored);

int main(int argc, char** argv)
{
    thr_init(4096); // exam: no failures
    tmatch_init(&matcher); // exam: no failures

    for (int t = 0; t < NTHREADS; t++) {
        tids[t] = thr_create(threadbody, (void *) t); // exam: no failures
    }
    for (int t = 0; t < NTHREADS; t++) {
        thr_join(tids[t], NULL);
    }
    printf("Done\n");
    tmatch_destroy(&matcher);
    thr_exit(0);
}

void *threadbody(void *ignored)
{
    int me = thr_getid();
    int partner1, partner2;
    int done = 0, rounds = 0, coolpartners;

    while (!done) {
        tmatch_match(&matcher, &partner1, &partner2);

        printf("I am %d, my partners are %d and %d.\n", me, partner1, partner2);

        if (((me + partner1 + partner2) % 3) == 2)
            coolpartners = 1;
        else
            coolpartners = 0;

        if (coolpartners) {
            printf("Whee!  That was so much fun I might do it again.\n");
        }
        if ((++rounds == 10) || !coolpartners) {
            done = 1;
        }
    }
    return 0;
}
```

Your task is to implement trio matchers with the following interface:

- `int tmatch_init(tmatch_t *tmp)` — initializes a trio matcher.
- `void tmatch_match(tmatch_t *tmp, int *p1p, int *p2p)` — "Reasonably promptly" returns the thread i.d.'s of two other threads invoking `tmatch_match()` on the same trio-matcher object. Note that `tmatch_match()`, as specified for this exam, does not return error codes.
- `void tmatch_destroy(tmatch_t *tmp)` — Deactivates a trio-matcher object. It is illegal for a program to invoke `tmatch_destroy()` if any threads are operating on 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 `tmatch_t` here. If you need one (or more) auxilary structures, you may declare it/them here as well.

```
typedef struct {










} tmatch_t;
```

(b) $\boxed{\text{15 points}}$ Now please implement `int tmatch_init()`, `void tmatch_match()`, and `void tmatch_destroy()`.

. . . space for trio-matcher implementation . . .

. . . space for trio-matcher implementation . . .

[You may use this page for your trio-matcher implementation if you wish.]

5. $\boxed{\text{5 points}}$ `Nuts & Bolts`.

In the Project 1 environment, where all code runs in kernel mode, when an interrupt, exception, or other “surprise” happens, the CPU pushes a “trap frame” onto the existing stack (in Project 1 there is no “stack switch” as happens during a user-to-kernel transition in Project 2 or Project 3).

Assuming an *interrupt* surprise occurs, list the three elements of the Project 1 (kernel-only) trap frame (which might more accurately be called an “interrupt frame”). Briefly say why each of those elements is saved onto the stack by the CPU (ideally, you should probably be able to suggest something that would go wrong if that element were *not* saved and later restored).

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.