Computer Science 15-410/15-605: Operating Systems
Mid-Term Exam (A), Spring 2025

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.	**10**		
3.	**15**		
4.	**20**		
5.	**10**		
	65		

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. [10 points] Short answer.

(a) [6 points] When designing a body of code, at times one finds oneself thinking, "I wonder if I can assume X?" According to the 15-410 design orthodoxy, immediately upon having such a thought one is required to ask oneself two questions. Please state those questions. It is probably worthwhile to include specific examples and/or to briefly explain why these two replacement questions are important.

You may use this page as extra space for the “assume” question if you wish.

(b) [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 which 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 which could have this effect).
3. Then write a *small* piece of code which 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: 0x00000000, ebx: 0x00000000, ecx: 0xffffeec4,
edx: 0xffffefc4, edi: 0x00000000, esi: 0x00000000,
ebp: 0x01000066, esp: 0x01000066, eip: 0x01000029,
 ss: 0x002b, cs: 0x0023, ds: 0x002b,
 es: 0x002b, fs: 0x002b, gs: 0x002b,
eflags: 0x00000282
```

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

2. [10 points] Deadlock.

Consider a large multi-threaded program running on a multi-core machine — for example, 1,000 threads running on 64 cores. Clearly it would be frustrating to debug deadlocks in such a program. But it might be worse if the program supports "plug-ins," meaning pieces of code that can be added to the program after it is built — perhaps via dynamic linking of shared libraries. That would mean that the program would in effect be written by many authors, so it might be very challenging to use the *deadlock-prevention* approach of banning large classes of allocation behavior, since it would be necessary to make sure that many authors understood on which behavior was banned and never made a mistake.

(a) [4 points] State the four ingredients necessarily present in every deadlock. Provide a brief example (one or two sentences should suffice) that makes it clear you understand each element.

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

Imagine one of your coworkers proposes using *deadlock detection and recovery* for this program.

(b) [3 points] State a *policy* problem/challenge that would plausibly arise when applying deadlock detection and recovery to this class of program.

(c) [3 points] State an *implementation* problem/challenge that would plausibly arise when applying deadlock detection and recovery to this class of program.

3. [15 points] Nemo's Algorithm (Dekker / Dijkstra / Raynal / Eckhardt).

Consider the following critical-section protocol. This protocol is designed for use by multiple threads. In each thread, `i` is the thread number, ranging from zero through `N`, the number of threads in the system. For the purposes of this question we can assume that `N` is a compile-time constant.

```
        volatile int turn = -1;    // initially, it's nobody's turn
        int entering[N] = {0, };   // per-thread flags (initially all zero)

 1.     int n_entering(void) {
 2.         int n_entering = 0;
 3.         for (int t = 0; t < N; t++)
 4.             n_entering += entering[t];
 5.         affirm(n_entering > 0);
 6.         return (n_entering);
 7.     }
 8.
 9.     void lock(void) {
10.         do {
11.             do {
12.                 entering[i] = 0;
13.                 if (turn == -1)
14.                     turn = i;
15.             } while (turn != i);
16.             entering[i] = 1;
17.         } while (n_entering() != 1);
18.     }
19.
20.     void unlock(void) {
21.         turn = -1;
22.         entering[i] = 0;
23.     }
```

There is a problem with this critical-section protocol. Identify a required property which this protocol does not have and then present a trace, using the format presented in class, that supports your claim. You may use more or fewer columns or lines in your trace.

Execution Trace

time	Thread 0	Thread 1
0		
1		
2		

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 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! If you wish, you may abbreviate `entering[]` as `e[]` and `n_entering()` as `n_e()`.

This page is for your Nemo's-Algorithm solution.

This page can be used for your Nemo's-Algorithm solution.

This page can be used for your Nemo's-Algorithm solution if you wish.

4. [20 points] Pausable semaphores.

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.

Something we largely do *not* cover in this class, that is increasingly important to workers in the field of operating systems, is managing heat and power. It is one thing to correctly compute a required answer, but it is often important to perform a computation while remaining within a heat and/or power budget. In this question you will implement a synchronization object called a "pausable semaphore." This object will provide the standard semaphore services allowing a pool of threads to acquire and release logical items, but it will also allow a power-management thread to, from time to time, suspend and resume threads when they are in the act of acquiring and releasing those logical items. While most of the threads are invoking `psem_signal()` and `psem_wait()` on a pausable semaphore, a power-management thread will be invoking `psem_pause()` and `psem_resume()`. While a pausable semaphore is paused, `signal()` and `wait()` operations *both* cause threads to block; when the pausable semaphore is resumed, threads "artifically" blocked by the pause operation resume, and `signal()` and `wait()` operate normally. Like a regular semaphore, a pausable semaphore object does not know how many threads will invoke it.

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

The remainder of this page is intentionally blank.

```
// Assumes that printf() of a short single-line message is atomic.
// Test code:  not required to exhibit reasonable synchronization behavior.

#define WIDTH 8 // number of coprocessors
#define THREADS 16 // number of clients

#define RUN_TICKS 10
#define SLEEP_TICKS 5
#define TOTAL_TICKS 15410

static psem_t limiter;

void *threadbody(void *vid) {
  int id = (int) vid;
  int base;

  while (1) {
    psem_wait(&limiter);
    printf("Thread %d running...fans will be ON\n", id);
    base = get_ticks();
    while (get_ticks() < base + RUN_TICKS)
      continue;
    printf("Thread %d done for now\n", id);
    psem_signal(&limiter);
    sleep(SLEEP_TICKS);
  }
  return (0);
}

int main(void) {
  thr_init(8192); // exam: cannot fail
  psem_init(&limiter, WIDTH); // exam: cannot fail

  printf("Starting %d threads\n", THREADS);

  for (int n = 0; n < THREADS; ++n)
    affirm(thr_create(threadbody, (void *)n) > 0); // exam: cannot fail

  int base, ticks;

  base = get_ticks();
  while ((ticks = get_ticks()) < base + TOTAL_TICKS) {
    sleep(ticks % 67); // heating is likely now
    psem_pause(&limiter);
    sleep(ticks % 23); // cool down a bit
    psem_resume(&limiter);
  }

  printf("All done!\n");
  task_vanish(0);
  panic("task_vanish() didn't???");
}
```

Your task is to implement pausable semaphores with the following interface:

- `int psem_init(psem_t *psp, int count)` — initializes a pausable semaphore (which begins unpaused). In the case of an error, this function will return an error code less than zero.
- `void psem_wait(psem_t *psp)` — acquires a logical object from the pool, blocking if necessary, and also blocking while the semaphore is paused.
- `void psem_signal(psem_t *psp)` — places a logical item into the pool, also blocking while the semaphore is paused.
- `void psem_pause(psem_t *psp)` — places a pausable semaphore into the "paused" state. It is expected that this operation is "fairly prompt."
- `void psem_resume(psem_t *psp)` – removes a pausable semaphore from the "paused" state, returning it to normal semaphore operation. It is expected that this operation is "more or less prompt in accordance with circumstances."
- `void psem_destroy(psem_t *psp)` – deactivates a pausable semaphore. It is illegal for a program to invoke `psem_destroy()` if any threads are operating on it.

The remainder of this page is intentionally blank.

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!

Note that this problem involves subtleties. It is possible to write code that "runs ok," but important to avoid code that is "correct but wrong," "arguably less wrong," or otherwise behaves in ways that are unwarranted.

(a) [5 points] Please declare your `psem_t` here. If you need one (or more) auxilary structures, you may declare it/them here as well. Then please implement `psem_init()`.

```
typedef struct {

} psem_t;
```

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

(b) [15 points] Now please implement `psem_wait()`, `psem_signal()`, `psem_pause()`, `psem_resume()`, and `psem_destroy()`.

. . . space for pausable-semaphore implementation . . .

. . . space for pausable-semaphore implementation . . .

[You may use this page for your pausable-semaphore implementation if you wish.]

5. [10 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 (there is no "stack switch" as happens during a user-to-kernel transition in Project 2 or Project 3).

List the three elements of the Project 1 (kernel-only) trap frame. Briefly say why each of those elements is saved onto the stack by the CPU (for example, 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 */
```

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.

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.