Computer Science 15-410: Operating Systems
Mid-Term Exam (A), Spring 2004

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. If you have a question about the exam, please write it down on the card you were provided with and then raise your hand.
4. This is a closed-book in-class exam. You may not use any reference materials during the exam.
5. You must complete the exam by the end of the class period.
6. Answer all questions. 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.
8. **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.	**15**		
	75		

1. [10 points] Give a *brief* definition of each of the following terms as they apply to this course. You may add a sentence providing an example or a clarification if you wish, but there is no need for long answers.

   (a) [2 points] XCHG

   (b) [2 points] Kernel stack

   (c) [2 points] Atomic instruction sequence

(d) [2 points] Exception

(e) [2 points] `yield()`

2. [15 points] Interrupt handling.

Consider the following attempt at a timer interrupt handler, perhaps written by a 410 student enamored of inline assembly language as opposed to stand-alone assembly language files.

```
static void
timer_handler() {
  static unsigned int ticks_since_boot = 0;

  asm("pusha");

  tick(++ticks_since_boot);

  if ((ticks_since_boot % (100*100)) == 0)
    /* debug printf every 100 seconds */
    printf("Timer interrupt %d\n", ticks_since_boot);

  outb(INT_CTL_REG, INT_CTL_DONE);

  asm("popa");
  asm("iret");
}
```

Please explain *in detail* the horrible thing that will happen when this code is run. Your explanation may include a clearly-drawn picture, but one is not required.

3. [20 points] Debugging a memory dump.

Given a dump of the stack memory and a symbol table, you will generate a stack trace as in Project 0. For each function in the trace, please write the name of the function and the names and values of the arguments. You may assume all strings are null terminated, and pointer arrays end with a NULL pointer. As was the case in Project 0, the first function you display will be the function which called the currently-running function (not the currently-running function itself). Your trace will be complete once you have displayed the invocation of the `main` function.

Stack Dump

Here is a dump of memory, starting just before %ebp(`0xbffff788`) at the time of the breakpoint. For convenience, we have provided both hex and characters forms. Note, however, that the hex is presented in big-endian form by word, for readability, where the characters are in Intel-native little-endian, so the individual bytes may not line up between the two dumps.

```
              3 2 1 0    7 6 5 4    b a 9 8    f e d c  0123456788abcdef
0xbffff780: 0x400162c0 0x00000002 0xbffff798 0x08048440 |.b.@........@...|
0xbffff790: 0x696e7500 0x000f3478 0xbffff7e8 0x08048379 |.unix4......y...|
0xbffff7a0: 0x0000000f 0x0000019a 0x40016000 0x00000003 |.........`.@....|
0xbffff7b0: 0x61616161 0x61616161 0x61616161 0x61616161 |aaaaaaaaaaaaaaaa|
0xbffff7c0: 0xbfff0061 0x40007b31 0x0804818d 0x400129b0 |a...1{.@.....).@|
0xbffff7d0: 0x40006a8b 0x40024b43 0x002a8982 0x0005007f |.j.@CK.@..*.....|
0xbffff7e0: 0xbffff83c 0x400125c0 0xbffff898 0x0804835b |<....%.@....[...|
0xbffff7f0: 0xbffff870 0xbffff830 0x40027b7d 0x080481c7 |p...0...}{.@....|
0xbffff800: 0x276e6f64 0x6f662074 0x74656772 0x65687420 |don't forget the|
0xbffff810: 0x6f6f6220 0x6572206b 0x74726f70 0x28205000 | book report.P (|
0xbffff820: 0x29100415 0x00000000 0xbffff8cc 0x01016000 |...).........`..|
0xbffff830: 0x6920534f 0x77612073 0x6d6f7365 0x40002165 |OS is awesome!.@|
0xbffff840: 0x00000003 0x40016280 0x00313370 0xdeadbeef |.....b.@p31.....|
0xbffff850: 0x72617473 0x20646574 0x20656874 0x6e72656b |started the kern|
0xbffff860: 0x79206c65 0x003f7465 0x080481a6 0x0177ff8e |el yet?.......w.|
0xbffff870: 0x7464696d 0x736d7265 0x65726120 0x6e756620 |midterms are fun|
0xbffff880: 0x40016200 0x00000000 0x00000001 0xbffff8dc |.b.@............|
0xbffff890: 0x40009eb1 0x400069bf 0xbffff8e8 0x0804850b |...@.i.@........|
0xbffff8a0: 0xbffff8b0 0x00000000 0xbffff8d0 0x00000000 |................|
0xbffff8b0: 0xbffff8d8 0xbffff8dc 0xbffff8d4 0xbffff8d6 |................|
0xbffff8c0: 0xbffff8da 0xbffff8de 0x00000000 0x4001d9dc |..............@|
0xbffff8d0: 0x40021e6c 0x0062006f 0x00610066 0x0072006f |l..@o.b.f.a.o.r.|
0xbffff8e0: 0x40009da0 0x4001037f 0xbffff958 0x4002c4ed |...@...@X.......|
0xbffff8f0: 0x00000003 0xbffff900 0x400ed705 0xbffffae3 |...........@....|
0xbffff900: 0xbffff930 0xbffff920 0xbffff910 0x00000000 |0... ...........|
0xbffff910: 0x6d5f534f 0x65746469 0x535f6d72 0x40003430 |OS_midterm_S04.@|
0xbffff920: 0xbf003234 0x40009eb1 0x40012934 0x400162b0 |42.....@4).@.b.@|
0xbffff930: 0x74732f2e 0x746b6361 0x00747365 0x4003ffd2 |./stacktest....@|
0xbffff940: 0xbffff9b0 0x40000d48 0x400125c0 0x00000003 |....H..@.%.@....|
0xbffff950: 0x00000000 0x4013dd30 0x00000000 0x08048271 |....0..@.......q|
```

Symbols

For each function, you are given the starting address, and for each argument to that function, you are given the offset of that parameter from %ebp.

address	*name*	*offset*
0804830e	char * sillyfunc(void)	
08048322	int funfunc(...)	
	char ** strs	00000008
08048374	char mystery(...)	
	char * s2	00000008
	char * s1	0000000c
080483ac	char strange(...)	
	int arg	00000008
	int * parg	0000000c
	char ch	00000010
0804841a	int crazy(...)	
	int x	00000008
	int y	0000000c
0804848d	int main(...)	
	int argc	00000008
	char ** argv	0000000c

Please fill in the trace below with names and values. Note that you may not need all of the lines provided. Please use the following format:

Function funname (i=42, str="string", strp=["foo", "bar"])

Function _____ (_____

_____) in

Function _____ (_____

_____) in

Function _____ (_____

_____) in

Function _____ (_____

_____) in

Function _____ (_____

_____) in

4. [15 points] Deadlock

It is time to build a house, and your task is to write an instruction manual for the carpenters. The instruction manual will take the form of a set of C functions. The carpenters must complete various jobs, each of which requires certain tools. These carpenters run a low-budget operation, so they have brought with them only one instance of each type of tool. The five tools are the `screwdriver`, the `power_drill`, the `hand_drill`, the `drill_chuck`, and the `clamp`. You may assume that there are plenty of nails, screws, and drill bits. In order to use a tool, a carpenter must first acquire the tool. In order to complete the house on time it is very important that the carpenters do not find themselves in a deadlocked state!

And now for some basic carpentry... In order to attach the right size drill bit to the `power_drill` or to the `hand_drill` you must also acquire the `drill_chuck`. You must also use the `drill_chuck` to detach the drill bit when a job is complete.

- In order to build a window frame, you must acquire the `screwdriver`, the `power_saw`, and the `clamp`.
- In order to install a doorknob, you need either the `hand_drill` or the `power_drill` and the `screwdriver`. You must acquire the drill and attach the drill bit before acquiring the `screwdriver`. Don't forget you must use the `drill_chuck` to attach or detach a drill bit.
- In order to repair a hinge, you need the following tools: the `clamp`, the `screwdriver`, and either the `hand_drill` or the `power_drill`. For this job, you must acquire the `screwdriver` before acquiring a drill.

(a) [7 points] Fill in the sequence of `request()`, `release()`, and `perform_work()` statements for each of the following jobs (e.g., `request(power_drill);`, `perform_work();`). Both `request()` and `release()` take a single resource parameter and block until they can complete. Use only these three statement types. Your solution *must* guarantee that the carpenters will never find themselves deadlocked and *should* avoid holding tools when they are not in use.

```
window_frame *build_window_frame(void){




}
```

```
doorknob *install_knob(void){









}

hinge *repair_hinge(void){









}
```

(b) [3 points] Explain why deadlock is not possible.

5. [15 points] Concurrency

For each of the following functions decide whether or not it is thread-safe.

If it is thread-safe, give a brief but strong argument explaining why it is thread safe. If it is not, either give a brief but strong argument that it is not or an execution trace demonstrating the problem.

For each of the problems, you should assume the semantics used for Project 2.

(a)
```
#define QUEUE_LEN 32

typedef struct{
  void *buf[QUEUE_LEN];
  int start;
  int end;
  mutex_t lock;
} queue_t;

/* This function adds data to the queue
 *
 * returns 0 on success, -1 on failure
 */
int enqueue(queue_t *queue, void *data) {

  /* If the queue is full, return failure */
  if (queue->start == (queue->end + 1) % QUEUE_LEN)
    return -1;

  /* lock the queue while we set the data and increment the length */
  mutex_lock(&queue->lock);
  queue->buf[queue->end] = data;
  queue->end = (queue->end + 1) % QUEUE_LEN;
  mutex_unlock(&queue->lock);

  return 0;
}
```

(b)
```
/* thread IDs */
int tid[] = {0, 0};

/* hello world message to be printed out */
char *greeting = "Hello world!\n";

/* the size of the stack for created threads */
#define STACK_SIZE 0x1000

void *foo(void *arg) {
  int my_tid, your_tid;
  char *msg;

  /* get my tid */
  my_tid = tid[(int)arg];

  /* get the tid of the other thread */
  your_tid = tid[((int)arg + 1) % 2];

  if (my_tid < your_tid) {
    /* send hello world message to other thread */
    thr_exit((void *)greeting);
  } else {
    /* receive the message and print it */
    thr_join(your_tid, NULL, (void **)&msg);
    print("%s", msg);
  }

  return NULL;
}

/* This program prints "Hello world!\n" */
int main() {
  thr_init(STACK_SIZE);

  /* create two threads */
  tid[0] = thr_create(foo, 0);
  tid[1] = thr_create(foo, 1);

  thr_exit(NULL);
}
```