Name (Last, First): _____

This exam consists of 5 questions on 8 pages. Be sure you have the entire exam before starting. The point value of each question is indicated at its beginning; the entire exam is worth 100 points. Individual parts of a multi-part question are generally assigned approximately the same point value; exceptions are noted. This exam is open text and notes. However, you may NOT share material with another student during the exam.

Be concise and clearly indicate your answer. Presentation and simplicity of your answers may affect your grade. Answer each question in the space following the question. If you find it necessary to continue an answer elsewhere, clearly indicate the location of its continuation and label its continuation with the question number and subpart if appropriate.

You should read through all the questions first, then pace yourself.

The questions begin on the next page.

Problem	Possible	Score
1	20	
2	20	
3	20	
4	20	
5	20	
Total	100	

1. (_____/20 points)

Short answer questions

(a) Consider the following C struct and definition of `mynode`:

```
struct node {
    int x;
    char a[8];
    int y;
    struct list_elem elem;
    int z;
}

struct node mynode;
```

On a 32 bit machine, what is the value of `((unsigned int) &((struct node *) 0)->y)`?

(b) What is the relationship between a stack and a thread? Can multiple threads use the same stack? Explain you answers.

(c) What are the differences between the priority scheduler and the 4.4 BSD Advanced Scheduler (mlfqs)?

(d) On entry to the kernel from a system call, why is it necessary to check that the user stack pointer (esp) is valid? What do need to check to ensure the esp is valid?

2. (_____/20 points)

User-level Memory Allocation

Recall your implementation of the user-level memory allocator API from Project 0. Also recall the following declarations from memalloc.h:

```
struct free_block
  {
    size_t              length;         /* length of block, including header */
    struct list_elem    elem;           /* list element for free list */
  };

struct used_block
  {
    size_t              length;         /* length of block, including header */
    uint8_t             data[0];        /* memory_block.data points at the
                                           memory behind the length . */
  };
```

Also assume that your implementation has the following global variables:

```
static struct list free_list;
```

Write a function called `int find_largest_free_block_size()` that returns the size largest free block on the list. I've provided the source code for list.c.c

3. (_____/20 points)

Multiprocessor Locking with Blocking

Recall the uniprocessor implementation of `lock_acquire()` with blocking:

```
lock_acquire(struct lock *lk) {
    int acquired = 0;
    while (!acquired) {
        disable_preemption();
        if (lk->value == UNLOCKED) {
            acquired = 1; lk->value = LOCKED;
        } else {
            listadd(lk->list, cur); thread_block();
        }
        enable_preemption();
    }
}
```

Now recall the multiprocessor implementation of `lock_acquire()` with `test_and_set()`:

```
lock_acquire(int *lock) {
    while (test_and_set(lock)) {
        while (*lock); /* loop */
    }
}
```

Show how to implement a multiprocessor version of `lock_acquire()` that blocks the thread after 3 attempts of trying to get the lock. Hint: you can use the struct lock in you new version of `lock_acquire()`.

4. (_____/20 points)

Double Locking

Consider the following code from synch.h:

```
bool
sema_try_down (struct semaphore *sema)
{
  enum intr_level old_level;
  bool success;

  ASSERT (sema != NULL);

  old_level = intr_disable ();
  if (sema->value > 0)
    {
      sema->value--;
      success = true;
    }
  else
    success = false;
  intr_set_level (old_level);

  return success;
}

bool
lock_try_acquire (struct lock *lock)
{
  bool success;

  ASSERT (lock != NULL);
  ASSERT (!lock_held_by_current_thread (lock));

  success = sema_try_down (&lock->semaphore);
  if (success)
    lock->holder = thread_current ();
  return success;
}
```

Write a new function called `lock_try_acquire2(struct lock *lock1, struct lock *lock2)` that tries to acquire both lock1 and lock2. If they can both be acquired then acquire the locks and return true, if they cannot both be acquired then return false. Your solution needs to be free of any race conditions. Think carefully.

You can write you solution on the next page.

Continue problem 4 here.

5. (_____/20 points)

System Calls: Adding Fork to Pintos

In Project 2 you implemented the `exec()` system call that starts a new process and loads an executable into the processes address space (allocating pages as necessary). Describe how you would implement the `fork()` system call in Pintos. You do not need to provide code (pseudo code is fine), but you need to provide enough detail such that someone could implement `fork()` from you description. I've provided the source code for process.c.

Continue your answers here if necessary.