CS560 Midterm
Spring 2008

1. Scheduling Algorithms

1.A. Multilevel feedback queues (23 points)

Tell me in as much detail as possible, everything you know about multilevel feedback queues.

1.B. Time quantum too small (2 points)

In the case of a round-robin scheduling algorithm, what is the disadvantage if the time quantum too small?

1.C. Time quantum too big (2 points)

What is the disadvantage if the time quantum is too big.

1.D. Time quantum just right (3 points)

What is the rule of thumb for sizing the time quantum correctly?

2. KThreads

2.A. setjmp, longjmp (18 points)

Tell me in as much detail as possible, everything you know about how the setjmp and longjmp calls are used in the implementation of the KThreads system (Hint: There are 3 places where setjmp and longjmp are used).

2.B. Preemption (3 points)

Are KThreads preemptive?

2.C. kt_fork (6 points)

When exactly is it that the function passed as an argument to the kt_fork call gets a piece of the cpu?

3. Deadlock

3.A. Conditions (5 points)

What are the 4 conditions necessary for deadlock.

3.B. Which (5 points)

Which ones can be broken in code?

3.C. How (5 points)

Describe in general terms, how one would accomplish this?

3.D. Coding example 1 (9 points)

Can the following code deadlock? Why or why not? (After a process calls **lock_multiple()** it will not call **lock_multiple()** again until it has called **unlock_multiple()** with the same list with which it called **lock_multiple()**. Processes only lock resources using **lock_multiple()**).

```
typedef struct {
  int instance;
  char *name;
  Semaphore *s;
} Resource;

void lock_multiple(Dllist resources)
{
  JRB t, tmp;
  Dllist dtmp;
  Resource *r;
  char *key;

  t = make_jrb();
  dll_traverse(dtmp, resources) {
    r = (Resource *) dtmp->val.v;
    key = (char *) malloc(sizeof(char) * (strlen(r->name+20));
    sprintf(key, "%10d %s", r->instance, r->name);
    jrb_insert_str(t, key, new_jval_v((void *) r));
  }
  while (!jrb_empty(t)) {
    r = (Resource *) t->flink->val.v;
    key = t->flink->key.s;
    P(r->s);
    jrb_delete_node(t->flink);
    free(key);
  }
  jrb_free_tree(t);
}

void unlock_multiple(Dllist resources)
{
```

```
  Dllist tmp;
  Resource *r;

  dll_traverse(tmp, resources) {
    r = (Resource *) tmp->val.v;
    V(r->s);
  }
  return;
}
```

3.E. Coding example 2 (9 points)

Can the following code deadlock? Why or why not?

```
#define NTHREADS (10)

pthread_mutex_t *locks[NTHREADS];

void *thread(void *arg)
{
  int id, *ip;

  ip = (int *) arg;
  id = *ip;

  while(1) {
    if (id == 0) {
      pthread_mutex_lock(locks[0]);
      pthread_mutex_lock(locks[1]);
      pthread_mutex_lock(locks[NTHREADS-1]);
    } else if (id == NTHREADS - 1) {
      pthread_mutex_lock(locks[0]);
      pthread_mutex_lock(locks[NTHREADS-2]);
      pthread_mutex_lock(locks[NTHREADS-1]);
    } else {
      pthread_mutex_lock(locks[id-1]);
      pthread_mutex_lock(locks[id]);
      pthread_mutex_lock(locks[id+1]);
    }
    sleep(random%10+1);
    pthread_mutex_unlock(locks[(id+NTHREADS-1)%NTHREADS]);
    pthread_mutex_unlock(locks[id]);
    pthread_mutex_unlock(locks[(id+1)%NTHREADS]);
  }
}

main()
{
```

```
  int i;
  pthread_t t[NTHREADS];
  pthread_attr_t attrs[NTHREADS];
  int ids[NTHREADS];

  for (i = 0; i < NTHREADS; i++) {
    locks[i] = (pthread_mutex_t *) malloc(sizeof(pthread_mutex_t));
    pthread_mutex_init(locks[i]);
  }

  for (i = 0; i < NTHREADS; i++) {
    ids[i] = i;
    pthread_attr_init(attr+i);
    pthread_attr_setscope(attr+i, PTHREAD_SCOPE_SYSTEM);
    pthread_create(t+i, attr+i, thread, id+i);
  }
  pthread_exit(NULL);
}
```

4. Semaphores (9 points)

Describe in the broadest possible terms how one might allow no more than two threads at a time into a critical section of code using a semaphore (Hint: Your answer should, at a minimum, include the use the following calls: make_kt_sem, P_kt_sem, V_kt_sem).