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. At a minimum, explain: what causes a process to move up or down in the queue hierarchy, what the relative sizes of the time quanta are, and what performance metric is optimized for each process type.

SOLUTION

2 points - Three queues (or multiple queues)
3 points - Processes move down in queues by not relenquishing cpu voluntarily
3 points - Processes move up by relenquishing cpu voluntarily or waiting too long
3 points - 2nd level queue time quantum is same or double 1st level queue
3 points - 3rd level queue time quantum is 10 times 2nd level queue or infinite
3 points - Processes run from given queue only if all higher level queues empty
3 points - CPU-bound processes spend less time context switching
3 points - IO-bound processes spend less time waiting

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?

SOLUTION

2 points - CPU-bound processes spend too much time context switching

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

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

SOLUTION

2 points - IO-bound processes spend too much time waiting

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

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

SOLUTION

3 points - 80% of CPU bursts expire before the timer interrupt fires

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: setjmp and longjmp are used when a thread starts, when a thread context switches, and when a thread exits).

SOLUTION

2 points - Start: setjmp to fill buffer
2 points - Start: change frame/stack pointers
2 points - Start: longjmp to new buffer
2 points - Start: run function or local variables inaccessible
2 points - Context switch: setjmp in kt_sched to fill buffer
2 points - Context switch: longjmp in kt_sched to someone else's buffer
2 points - Context switch: longjmp later in kt_sched back to my buffer
2 points - Exit: setjmp when starting to fill exit buffer
2 points - Exit: longjmp when exiting to exit buffer

2.B. Preemption (3 points)

Are KThreads preemptive?

SOLUTION

3 points - No

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?

SOLUTION

3 points - The function gets put on the ready queue at start time
3 points - When the scheduler decides it is that job's turn, it gets the cpu

3. Deadlock

3.A. Conditions (5 points)

What are the 4 conditions necessary for deadlock.

SOLUTION

1.25 points - Hold and wait
1.25 points - Circular wait
1.25 points - Mutual exclusion
1.25 points - No preemption

3.B. Which (5 points)

Which ones can be broken in code?

SOLUTION

2.5 points - Hold and wait
2.5 points - Circular wait

1 person said all 4 conditions can be broken in code (book answer - full credit)

3.C. How (5 points)

Describe in general terms, how one would accomplish this?

SOLUTION

2.5 points - Break hold and wait by only locking resources if all resources can be locked
2.5 points - Break circular wait by locking resources in order

-1 point if you said resources must be unlocked in order
+1 point if you mentioned safe state

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;
}
```

SOLUTION

3 points - No
3 points - Circular wait was broken
3 points - Resources locked in order

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);
}
```

SOLUTION

3 points - No
3 points - Circular wait was broken
3 points - Resources locked in order

4. Semaphores (7 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).

SOLUTION

3 points - Call make_kt_sem(2)
2 points - Begin critical section with P_kt_sem()
2 points - End critical section with V_kt_sem()