ECE598: Advanced Operating Systems – Practice Homework 10

Spring 2016

Scheduling, Locking and Context Switches

Due: Not due, practice for midterm

Answer the following multiple choice questions.

1. Scheduling

   (a) What is the difference between a plain round-robin scheduling algorithm and a priority-based one?

   (b) You design a scheduling algorithm for a multi-core system that tests every possible combination of processes before choosing one (factorial time, `O(N!)`). Why might this be a bad idea?

2. Threading

   (a) What is a benefit of userspace threads?

   (b) What is a benefit of kernel threads?

3. Multiprocessors / Locking

   Look at the memory allocation code below and answer the following questions.

   (a) Would this code need locking on a single-processor system?

   (b) Would this code need locking on a multi-processor system?

   (c) What is the latest point you could take the lock (A-F)?

   (d) What is the earliest point you could release the lock (A-F)?

```c
void *memory_allocate(int size) {

    /* A */
    int first_chunk,num_chunks,i;

    /* B */
    if (size==0) size=1;
    num_chunks = ((size-1)/CHUNK_SIZE)+1;

    /* C */
    first_chunk=find_free(num_chunks);

    /* D */
    if (first_chunk<0) {
        printk("Error!  Could not allocate %d of memory!\n",size);
        return NULL;
    }

    /* E */
    for(i=0;i<num_chunks;i++) memory_mark_used(first_chunk+i);

    /* F */
    return (void *)(first_chunk*CHUNK_SIZE);
}
```

Solutions

1. Scheduling

   (a) Round-robin scheduling simply means switching in each process in turn, one at a time. No special consideration is given to any process.
   With weighted scheduling the processes can be assigned different weights (priorities) and the more important ones will be scheduled more often.

   (b) Schedulers (since they run at context-switch time) need to be very fast. An O(N!) scheduler is likely not to be very fast, especially once you have more than a few processes running.

2. Threading

   (a) Userspace threads have fast context switches as they do not have to call into the kernel.

   (b) Kernel threads can run across multiple CPUs (userspace threads live inside of one process so only can run on one CPU at a time).

3. Locking

   (a) Usually you do not need locking on a single-processor system, but it is possible to end up with re-entrant code. For example, if an interrupt handler was doing memory allocations (this is usually considered a bad idea) and it happened to interrupt the memory allocation code.

   (b) Yes, you need to lock as you can have a race condition if two CPUs are trying to read the free list and then update it at the same time.

   (c) You would want to lock at C as that's the point where the free list (the critical section) is first accessed.

   (d) You can release the lock at F as that's after the free list has been updated so it is in an up-to-date state and other CPUs can now read the list and can a valid result.