CS 111: Operating System Principles

Final Exam Summer '21

Instructor: Jon Eyolfson

1. Page Replacement (30 points total)
Assume the following accesses to physical page numbers:

1, 2, 3, 4, 5, 2, 3, 1, 4, 2

or in table format:

Access	1	2	3	4	5	6	7	8	9	10
Physical Page	1	2	3	4	5	2	3	1	4	2

Assume that all pages are initially on disk. For each access you'll have to answer which page gets evicted, and which page gets brought in. You have 4 physical pages in memory.

Unless otherwise stated, you'll be using the clock algorithm to replace pages. Recall on a page hit, you'll set the reference bit to 1.

1a. (22 points) After each access, if there's a page fault, state time, the page number that caused the fault, and the page it replaces.

1b. (2 points) How many total page faults are there when using the clock algorithm? **8**

1c. (6 points) How many total page faults are there when using the OPTIMAL algorithm? **6**

2. Page Tables (10 points)

Consider a system with 39-bit virtual addresses, 44-bit physical addresses, 4096 byte pages, and the PTE size is 4 bytes. This system is impossible to use in practice. What aspect of the system needs to change for it to be usable? Explain why your change now makes it usable.

3. Threads (20 points total)
Consider the following code:

```
#include <pthread.h>

#define NUM_THREADS 4

void *run(void *arg) {
  fork();
  printf("PID %d Thread %d\n", getpid(), gettid());
  return NULL;
}

int main() {
  for (int i=0; i < NUM_THREADS; ++i) {
    pthread_t thread;
    pthread_create(&thread, NULL, &run, NULL);
    pthread_detach(thread);
  }
  return 0;
}
```

Assume that no functions can return errors.

3a. (8 points) The parent process should print 4 times (once per thread), but as it's written it doesn't(!). Why do I not always see the parent process print 4 times? (Hint: you can ignore the `fork` for this question). What is the fix to prevent this problem?

3b. (12 points) Assume that gettid returns an integer between 1-4 inclusive that's unique for each thread and that it persists even after a fork. When we run our fixed code we get this output from the original parent process:

```
PID 100 Thread 1
PID 100 Thread 2
PID 100 Thread 3
PID 100 Thread 4
```

Assuming that the parent process has a PID of 100 and all the children receive PIDs greater than 100 (sequentially), write out ONLY the output from the child processes. There are many different possibilities, yours just has to be consistent.

4. Locks (20 points)
Consider the following code:

```
pthread_rwlock_t *rwlock;
int count = 0;

void *increment(void *arg) {
  pthread_rwlock_rdlock(&rwlock);
  int x = *count;
  pthread_rwlock_unlock(&rwlock);

  ++x;

  pthread_rwlock_wrlock(&rwlock);
  *count = x;
  pthread_rwlock_unlock(&rwlock);
}
```

Assume that the read-write lock gets initialized properly. Recall that rdlock locks for readers only, and wrlock locks locks for a single writer. The unlock call unlocks both types of lock calls properly.

4a. (8 points) Does this code have a data race? If not, explain why. If so, show an incorrect outcome using only 2 threads (each thread executing increment once).

4b. (12 points) Without changing the lock to a mutex (keeping it as a rwlock), explain what you would have to change to prevent all data races. You can just describe your answer, or copy/paste the code and apply your changes.

5. Locking (20 points)
You want to write a program that has one thread that prints status updates every 10 times processed, you initially come up with the following code:

```
#define NUM_THREADS 4

pthread_mutex_t mutex;
/* (1) */
int count;

void *regular(void *) {
  for (int i = 0; i < 25000; ++i) {
    /* Do something neat that takes a long time */
    pthread_mutex_lock(&mutex);
    ++count;
    /* (2) */
    pthread_mutex_unlock(&mutex);
  }
  return NULL;
}

void *every_10(void *) {
  for (int i = 0; i < 10000; ++i) {
    /* (3) */
    printf("Processed %d times\n", count);
    /* (4) */
  }
  return NULL;
}

int main() {
  /* create 4 threads that run regular */
  /* create 1 thread that runs every_10 */
  return 0;
}
```

Assume that you don't have to error check, and threads are handled properly. Explain the changes you would have make to have the `every_10` function print only after processing 10 items (e.g. it should only print 10, 20, 30, etc.). There are program points (1), (2), (3), and (4) to help you with your explanation. You can explain your changes at these points, or copy/paste the skeleton and make your modifications.

(Hint: you should only have to have modifications at (1), (2), (3), (4), and this question was designed for an online test, you may want to look at Lecture 12).

6. Memory Allocation (20 points total)
Assume you have a buddy allocator that initially has a single 512 byte free block. You cannot allocate more memory.

6a. (4 points) The first allocation is for 200 bytes. Describe how the allocation occurs with the buddy algorithm. State what's in the free list after the allocation.

6b. (2 points) How many bytes are lost to internal fragmentation after this first allocation?

6c. (4 points) Next, we deallocate the 200 byte allocation. Describe what happens. State what the free list looks like now.

6d. (4 points) Now, we get 4 different 100 byte allocations. You don't have to describe the process. However, state the free list at the end of all 4 allocations.

6e. (2 points) How many bytes are lost due to internal fragmentation after the 4 allocations?

6f. (4 points) Describe an approach you could take to fit another 100 byte allocation within your original 512 bytes. You can assume you can use your approach to allocate all 5 100 byte allocations from scratch.

7. Advanced Scheduling (5 points) What is priority inversion with respect to scheduling, and how would you mitigate it?

8. Disks (5 points) Assume you have NAND Flash (SSD) with 4 KiB pages and 128 pages per block. If a block is completely full (all 128 pages have data), explain deleting a single page would work. You can assume there's already a newly erased block that's writable.

9. Distributed Systems (10 points)

You're running a server that runs test cases using a RPC call. Clients request a git commit and a test case and receive a boolean response whether the test passes or not. Describe one pro and one con of using a "at most once" RPC scheme over a "best effort" scheme.

10. Virtual Machines (10 points)

Why do cloud providers heavily use virtual machines, as opposed to giving customers access to bare metal hardware? To answer the question: name two benefits of using virtual machines.