Midterm Review

Computer Operating Systems, Fall 2023

Instructor: Travis McGaha

Head TAs: Nate Hoaglund & Seungmin Han

TAs:

Andy Jiang	Haoyun Qin	Kevin Bernat	Ryoma Harris
Audrey Yang	Jason hom	Leon Hertzberg	Shyam Mehta
August Fu	Jeff Yang	Maxi Liu	Tina Kokoshvili
Daniel Da	Jerry Wang	Ria Sharma	Zhiyan Lu
Ernest Ng	Jinghao Zhang	Rohan Verma	

Administrivia

- Midterm is coming soon (**TODAY!**)
  - Meyerson B1 7:00 pm to 9:00pm Thursday 10/19
  - If you can't make the time, please send me an email **ASAP**

- Midterm Policies posted on the course website. Please read through them.
  - You are allowed 1 page of notes 8.5 x 11 double sided notes
  - Clobber policy: can show growth by doing better on the second midterm

Lecture Format

- I've written 1 new question, a thread question that is more "conceptual"

- Rest of lecture is your chance to ask last minute questions, anything that you want me to go over before the exam.

Disclaimer

- **THIS REVIEW IS NOT EXHAUSTIVE**
- **Topics not in this review are still testable**

Disclaimer

- **NOT ALL QUESTIONS WILL BE LIKE THE ONES IN LECTURE REVIEW**
  - **Most will, and other review materials don't give practice on this**

New Threads Question Pt.1

- I want to calculate all the prime numbers between 0 and 500, I've written some code to distribute the work across 5 worker threads.
  - This is pseudo code, it is not legal C

```
#define START 0
#define END   500
#define NUM_WORKERS 5
#define INCREMENT ((START - END) / NUM_WORKERS)

void calculate_primes(int start, int end, list<int>* results) {
  for (int i = start; i < end; i++) {
    if (isprime(i)) {
      (*results).add(i);
    }
  }
}
```

New Threads Question Pt.1

```
#define START 0
#define END   500
#define NUM_WORKERS 5
#define INCREMENT ((START - END) / NUM_WORKERS)

void calculate_primes(int start, int end, list<int>* results) {
  for (int i = start; i < end; i++) {
    if (isprime(i)) {
      (*results).add(i);
    }
  }
}

int main() {
  // pseudo code, this doesn't compile
  list<int> results;
  pthread_t threads[NUM_WORKERS];

  for (int i = 0; i < NUM_WORKERS; i++) {
    // create a thread to calculate between i * INCREMENT and (i + 1) * INCREMENT
    pthread_create(calculate_primes(i * INCREMENT, (i + 1) * INCREMENT, &results));
  }

  for (thd in threads) {
    pthread_join(thd, NULL);
  }

  // do something with all the prime numbers

  return EXIT_SUCCESS;
}
```

- This is pseudo code, it is not legal C

New Threads Question Pt.1

```
14 int main() {
15   // pseudo code, this doesn't compile
16   list<int> results;
17   ucontext_t contexts[NUM_WORKERS];
18
19   for (int i = 0; i < NUM_WORKERS; i++) {
20     // create a context to calculate between i * INCREMENT and (i + 1) * INCREMENT
21     getcontext(&(contexts[i]));
22     makecontext(&(contexts[i]), calculate_primes(i * INCREMENT, (i + 1) * INCREMENT, &results));
23   }
24
25   for (context in contexts) {
26     ucontext_t parent_context;
27
28     context.stack = allocate_new_stack;
29     context.link = &parent_context;
30     swapcontext(&parent_context, &context);
31
32   }
33
34   // do something with all the prime numbers
35
36   return EXIT_SUCCESS;
37 }
```

- This is pseudo code, it is not legal C

New Threads Question Pt.1

- I want to calculate all the prime numbers between 0 and 500, I've written some code to distribute the work across 5 worker threads.

- Which implementation would likely be faster, the one using ucontext or the one using pthreads?
  - What if we assume we have multiple CPU cores?
  - What if we assume that we have a single cpu core and no other running programs?

New Threads Question Pt.2

- What if instead of finding primes. I had a list of 50 files and wanted to count the number of times the word “tacoma” showed up in the files. I split the work up across 5 threads again.

- Which implementation would likely be faster, the one using ucontext or the one using pthreads?
  - Assume that we have a single cpu core and no other running programs.

New Threads Question Pt.3

- In both of these problems, I used threads and gave each thread a pointer to a list they could store their results onto.

- Why doesn't this work if I had instead fork()'d processes?