Midterm Review

Computer Operating Systems, Spring 2024

Instructor: Travis McGaha

Head TAs: Nate Hoaglund & Seungmin Han

TAs:

Adam Gorka	Haoyun Qin	Kyrie Dowling	Ryoma Harris
Andy Jiang	Jeff Yang	Oliver Hendrych	Shyam Mehta
Charis Gao	Jerry Wang	Maxi Liu	Tom Holland
Daniel Da	Jinghao Zhang	Rohan Verma	Tina Kokoshvili
Emily Shen	Julius Snipes	Ryan Boyle	Zhiyan Lu

Administrivia

- Midterm is coming soon (2 days from now!)
  - Meyerson B1 5:15 pm to 7:15 pm Thursday 2/29
  - 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 final
- Lecture today will be exam review
  - Thurs will be the exam
  - Travis will move office hours to be earlier on Thursday

Midterm Philosophy / Advice (pt. 1)

- I do not like midterms that ask you to memorize things
  - You will still have to memorize some critical things.
  - I will hint at some things, provide documentation or a summary of some things. (for example: I will provide parts of the man pages for various system calls)

- I am more interested in questions that ask you to:
  - Apply concepts to solve new problems
  - Analyze situations to see how concepts from lecture apply

- Will there be multiple choice?
  - If there is, you will still have to justify your choices

Midterm Philosophy / Advice (pt. 2)

- I am still trying to keep the exam fair to you, you must remember some things
  - High level concepts or fundamentals. I do not expect you to remember every minute detail.
    - E.g. how a file descriptor table works should be known, but not the exact details of what is in each entry of the open file table
    - (I know this boundary is blurry, but hopefully this statement helps)

- I am NOT trying to "trick" you (like I sometimes do in poll everywhere questions)

Midterm Philosophy / Advice (pt. 3)

- I am trying to make sure you have adequate time to stop and think about the questions.
  - You should still be wary of how much time you have
  - But also, remember that sometimes you can stop and take a deep breath.
- Remember that you can move on to another problem.
- Remember that you can still move on to the next part even if you haven't finished the current part

Midterm Philosophy / Advice (pt. 4)

- On the midterm you will have to explain things
- Your explanations should be more than just stating a topic name.
- Don't just say something like (for example) "because of threads" or just state some facts like "threads are parallel and lightweight processes".
- State how the topic(s) relate to the exam problem and answer the question being asked.

Disclaimer

- **THIS REVIEW IS NOT EXHAUSTIVE**

- **Topics not in this review are still testable**

Topics

- Fork() & Interleavings
- Signals
- Processes & IPC (Tcsetpgrp, pipe, exec, etc)
- File System
- Caches

fork

- Consider the following C code that uses fork()
  - Which of these outputs are possible? Please justify your answer
  - 380380
  - 338008
  - 380803

```c
int main() {
  pid_t pid = fork();
  pid = fork();

  if (pid == 0) {
    printf("3");
  } else {
    printf("8");
    int status;
    waitpid(pid, &status, 0);
    printf("0");
  }
}
```

Signals: Critical Sections

- A vector is data structure that represents a resizable array. For those used to Java, think of it like an ArrayList.
- Consider the following C snippet that outlines what a vector of floats is and how we would push a value to the end of it. Is there a critical section in the vec_push function? If so, what line(s)?

```
typedef struct vec_st {
  size_t length, capacity;
  float* eles;
} Vector;

void vec_push(Vector* this, float to_push) {
  // assume that we don't have to resize for simplicity
  assert(this->length < this->capacity);
  this->length += 1;  // increment length to include it
  this->eles[this->length - 1] = to_push;  // add the ele to the end
}
```

Signals Continued

- Signals can happen at any time and thus there are issues with making signal handlers safe to avoid any critical sections. In general, it is advised to keep signal handlers as short as possible or just avoid them at all costs.

- In each of these scenarios, tell us whether it is necessary to use signals and register a signal handler. If it is necessary, how safe is it?

- We want to have our program acknowledge when a user presses CTRL + Z or CTRL + C and print a message before exiting/stopping

Signals Continued

- The user needs to type floating point numbers to stdin, but there are some special floating point numbers like NaN, infinity, and –infinity. To avoid this, we have the user hit CTRL + C for NaN, CTRL + Z for infinity and other key combinations for other special values.

Processes

- We want to write a in C program that will compile and evaluate some other program. The program we are grading is similar to penn-shredder. For this program we write, lets assume we are running penn-shredder once and evaluating it. We need to be able to:
  - Specify the input and get output of the shredder
  - Set a time limit so that penn-shredder doesn't go infinite
  - Setup penn-shredder to receive signals from the keyboard (e.g. CTRL + C and CTRL + Z)

- Roughly how many times do we need to call each of these system calls? Briefly explain any system call you specify non-zero for

Processes Cont.

- Roughly how many times do we need to call each of these system calls? Briefly explain your answer for every system call.

System Call	Number	Justification
fork()		
execvp()		
pipe()		
waitpid()		
kill()		
signal()		
tcsetpgrp()		

File System Block Allocation

- Consider that we want to read the 5th block of the file **`/home/me/.bashrc`**, what is the worst-case number of disk blocks that must be read in for each of the following:
  - You can assume a block is 4096 bytes
  - assume that directory entries we are looking for are in the first block of each directory we search
- Linked List Allocation
  - Assume we know the block number of the first block in root dir
- Linked List Allocation via FAT
  - Assume we know where the root directory starts in the FAT.
  - You can also assume a FAT entry is 2 bytes.
- I-nodes
  - assume we know where the I Node for the root directory is

File System Block Allocation

- How does the numbers change if we instead wanted to write to the 5th block of the file?

- Despite not having the best numbers, I nodes are still chosen over FAT. Why is this the case?

Caches Q1

- Let's say we are making a program that simulates various particles interacting with each other. To do this we have the following structs to represent a color and a point

```
struct color {
  int red, green, blue;
};
```

```
struct point {
  double x, y;
  struct color c;
};
```

- If we were to store 100 point structs in an array, and iterate over all of them, accessing them in order, roughly how many cache hits and cache misses would we have?
  - Assume:
    - a cache line is 64 bytes
    - the cache starts empty
    - `sizeof(point)` is 32 bytes, `sizeof(color)` is 16 bytes

Caches Q2

- Consider the previous problem with point and color structs.

- In our simulator, it turns out a VERY common operation is to iterate over all points and do calculations with their X and Y values.

- How else can we store/represent the point objects to make this operation faster while still maintaining the same data? Roughly how many cache hits would we get from this updated code?