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");
  }
}
```

fork

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

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

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

  - 380380
    - Possible, we have four processes from calling fork() twice. Pid is set from the second call to fork which is called from two different processe resulting in two new children.
    - To get 380380 we can imagine that there are two parent-child relationships created from the second call to fork. We can run one of those children and then its parent in that order. We can repeat this process for the second parent-child pair.

fork

- Consider the following C code that uses fork()
  - Which of these outputs are possible? Please justify your answer
  - 338008
    - Impossible, we can't have an 8 last. Within a process it still executes in a specific order, a process cannot print 0 and then 8.

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

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

fork

- Consider the following C code that uses fork()
  - Which of these outputs are possible? Please justify your answer
  - 380803
    - Impossible, we can't have a 3 as last the parent waits on the child before it prints "0". This means the child waited on must finish before the parent prints "0", so "3" must come before an "0".

```
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: Critical Sections

- The last two lines could have an issue. Since they modify a piece of memory that could be shared, there may be an issue with signal handlers pushing onto the vector at the same time. Consider the case where we increment length and then a signal handler runs to push something onto the end of the vector.

```c
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

- We want to have our program acknowledge when a user presses CTRL + Z or CTRL + C and print a message before exiting/stopping
  - **Probably need this to be done with signals to catch CTRL + Z and CTRL + C. Can't catch them otherwise**
  - **The printing may not be safe since we are modifying global state when we go through the file system to print**

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.

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.
  - **Do not do this. We can have the user input the floating point values in other ways without us having to resort to using signals to communicate this.**

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()		

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()	2	Need to fork compiler and penn-shredder
execvp()	2	To exec compiler and penn-shredder
pipe()		
waitpid()		
kill()		
signal()		
tcsetpgrp()		

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()	2	Need to fork compiler and penn-shredder
execvp()	2	To exec compiler and penn-shredder
pipe()	2	To send input and get output from penn-shredder
waitpid()		
kill()		
signal()		
tcsetpgrp()		

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()	2	Need to fork compiler and penn-shredder
execvp()	2	To exec compiler and penn-shredder
pipe()	2	To send input and get output from penn-shredder
waitpid()	2	To wait for compiler and penn-shredder to terminate
kill()		
signal()		
tcsetpgrp()		

Processes Cont.

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

Some of these can have varying answers. If this was an exam, as long as they are reasonable and justified well, we will accept it

System Call	Number	Justification
fork()	2	Need to fork compiler and penn-shredder
execvp()	2	To exec compiler and penn-shredder
pipe()	2	To send input and get output from penn-shredder
waitpid()	2	To wait for compiler and penn-shredder to terminate
kill()	1	Debatable, can be justified if you used it. I use it to kill the child after timeout has occurred. Better than just using alarm in child since we can handle the timeout more elegantly and print out an error
signal()		
tcsetpgrp()		

Processes Cont.

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

Some of these can have varying answers. If this was an exam, as long as they are reasonable and justified well, we will accept it

System Call	Number	Justification
fork()	2	Need to fork compiler and penn-shredder
execvp()	2	To exec compiler and penn-shredder
pipe()	2	To send input and get output from penn-shredder
waitpid()	2	To wait for compiler and penn-shredder to terminate
kill()	1	… (trimmed for space see previous slides)
signal()	1	Debatable again. Used to register SIGALRM for timeout. Could be avoided if we register alarm in child
tcsetpgrp()		

Processes Cont.

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

Some of these can have varying answers. If this was an exam, as long as they are reasonable and justified well, we will accept it

System Call	Number	Justification
fork()	2	Need to fork compiler and penn-shredder
execvp()	2	To exec compiler and penn-shredder
pipe()	2	To send input and get output from penn-shredder
waitpid()	2	To wait for compiler and penn-shredder to terminate
kill()	1	… (trimmed for space see previous slides)
signal()	1	… (trimmed for space see previous slides)
tcsetpgrp()	1	Debatable again. used so penn-shredder has control of terminal and so it will get the keyboard signals and our program won't. Could instead register the signals in our program with signal and use kill in handler to send to child.

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

- 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
  - 1 read for the directory entry of home/ inside of /
  - 1 read for the directory entry of me/ inside of /home/
  - 1 read for the directory entry of .bashrc inside of /home/me/
  - 5 reads to get and read the 5th block of the file

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 via FAT
  - Assume we know where the root directory starts in the FAT.
  - You can also assume a FAT entry is 2 bytes.
  - 1 read for the directory entry of home/ inside of /
  - 1 read for the directory entry of me/ inside of /home/
  - 1 read for the directory entry of .bashrc inside of /home/me/
  - 1 read to get and read the 5th block of the file

File System Block Allocation

- I-nodes
  - assume we know where the I Node for the root directory is
  - 1 read to read in the inode of the root directory
  - 1 read for the directory entry of home/ inside of /
  - 1 read for the inode for /home/
  - 1 read for the directory entry of me/ inside of /home/
  - 1 read for the inode for /home/me/
  - 1 read for the directory entry of .bashrc inside of /home/me/
  - 1 read for the inode of .bashrc
  - 1 read to get and read the 5th block of the file
  - 8 disk reads

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?

File System Block Allocation

- How does the numbers change if we instead wanted to write to the 5th block of the file?
  - Nothing changes, we would just write to the 5th block of the file instead of reading it.

- Despite not having the best numbers, I nodes are still chosen over FAT. Why is this the case?
  - FAT takes up a lot of memory because we are caching the state of the entire filesystem in memory.
  - Inodes allow us to instead cache the information for relevant files in memory, so much lower memory consumption and similar performance for the most 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 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

Roughly every other time we access a point struct, it will already be in the cache. The other 50% of the time, it needs to be fetched from memory

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?

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?

Change point to just be:

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

Then Store two arrays:

```
point arr1[100];
color arr2[100];
// point at index I
// has color arr2[i]
```

Each time we access a point, we can now load 4 points into the cache. We now get ~25 cache misses and 75 hits