Travis McGaha & Joel Ramirez Spring 2025
CIS4480

CIS4480 Midterm Exam

This is a closed calculator/computer exam. You may consult one double-sided sheet of notes. No other references are allowed.

If you provide an incorrect answer, any explanation about how you arrived at your solution may lead to partial credit.

Full Name [print]: _____

PennID: _____

Exam Details & Instructions

- The exam consists of 5 questions (and a short bonus 6th question) worth 100 points total.
- You have 90 minutes to complete the exam.
- The exam is closed book. No textbooks, phones, laptops, wearable devices, or any other electronics are allowed beyond the permitted notes.
- You may use one 8.5" x 11" double-sided sheet of paper for notes.
- Turn off and put away all electronic devices and noise-making items.
- Remove all hats, headphones, and watches.

Pledge: I neither cheated nor helped anyone else cheat on this exam. All answers are my own. Violating this pledge may result in a failing grade.

Sign Here: _____

Advice

- There are 5 questions (with 14 parts total). Budget your time to complete all of them.
- Don't worry if there is more space than you need for answers. Extra space is provided just in case.
- Take a deep breath and relax. A bad grade on this exam is not the end of the world. You can improve your score with the Midterm "Clobber" Policy (see the course syllabus).

Please initial or write your PennID on each page to ensure they can be matched if they become separated.

The last page is blank for extra space. If you use it, indicate clearly where your answer continues and include your full name and PennID at the top.

1 Files, Files, and Files [17 Points]

Take a look at the following block of code, which forks a child process, establishes a pipe between the parent and child, and transmits the entire contents of a file through the pipe using the function `write_file_pipe`.

You may assume that your PennShell is the only running process that references the controlling terminal prior to it executing the following code as a **single job.**

```
1  const char *argv[] = {"./myprogram", NULL};
2
3  int main(){
4
5      char *file_name_str = "my_file.txt"; // question 2.3 getline(....);
6      int file_fd = open(file_name_str, O_RDONLY);
7
8      int pipe_arr[2];
9      pipe2(pipe_arr, O_CLOEXEC);
10
11     pid_t child = fork(); //Line 11
12
13     if(child == 0){
14         close(file_fd);
15         dup2(pipe_arr[0], STDIN_FILENO); //line 15
16         execvp(argv[0], argv);
17         exit(EXIT_FAILURE);
18     }
19
20     close(pipe_arr[0]);
21
22     write_file_pipe(pipe_arr[1], file_fd);
23
24     close(file_fd);
25     close(pipe_arr[1]);
26
27     waitpid(-1, NULL, 0);
28     printf("Finished waiting for child. \n");
29
30 }
```

Problem continues onto the next page

(a) How many descriptors? [5 Points]

The **job** is executed by your PennShell **in the foreground**. By the time we reach `line 11` (i.e. right before the fork but haven't executed it), **how many file descriptor(s) are in use by this foreground job? Justify your answer.**

(b) How many can write? How many can read? [6 Points]

The **job** is executed by your PennShell **in the foreground**. By the time `line 11` (i.e. the fork) is executed successfully, **how many *processes* are allowed to write to the terminal? How many are able to read from it? Justify your answer. Assume the parent and child have not gone further than the fork.**

(c) get(in)line PennShell Partner [6 Points]

Your PennShell partner wants to make your life harder! (Go figure.) Instead of hardcoding which file to open, they suggest using `getline(...)` to prompt the user for a filename. They then modify the code accordingly and run the job **in the background**.

What should happen when this job is run? **Explain.**

2 A Set of Signals is Called a `sigset_t` [16 Points]

As a senior software engineer at your highly lucrative company, you're responsible for mentoring an intern. Additionally, your team is in charge of developing a ***New Operating System***. The ***New Operating System*** makes preemption of a task more easily controllable by user code by having the kernel send two new signals, `SIGSLEEP` and `SIGWAKE`, **at any time**. These signals cause a process to suspend itself immediately (`SIGSLEEP`) or resume execution (`SIGWAKE`) ***under all circumstances***. While these signals can be blocked, they ***cannot be 'ignored'***.

A `SIGWAKE` is **always preceded** by a `SIGSLEEP`. If a process receives a `SIGSLEEP`, it is guaranteed to receive a `SIGWAKE` at some unknown future time. If both SIGWAKE and SIGSLEEP are pending signals (blocked), they are not delivered to the process and are discarded. Additionally, **the only way to continue a process suspended via SIGSLEEP is with a SIGWAKE.**

Your intern writes this code to demonstrate that signals work in the new operating system. The code overview is as follows:

- Install a handler for `tested_signal`.
- Block the signal, `tested_signal`.
- Send the process that signal using `kill`.
- Unblock the signal.
- Install a handler for `SIGALRM`.
- Call `wait_for_alarm` to wait for the alarm that the `tested_signal`'s handler registered.

```c
void signal_handler(int sig); //Calls ALARM(10);
void alrm_signal_handler(int sig); //Writes to the terminal "Alarm caught!".

void test_signal_mask(int signal) {
    sigset_t new_set;
    sigemptyset(&new_set);
    sigaddset(&new_set, signal);
    sigprocmask(SIG_BLOCK, &new_set, NULL);
    kill(0, signal); // Send the signal to itself
    sigprocmask(SIG_UNBLOCK, &new_set, NULL);
}

//This function should RETURN ONLY IF and WHEN A SIGALRM IS RECEIVED.
// An alarm should be registered (by the tested signal's handler) to make this function return.
void wait_for_alarm() {
    sigset_t set;
    sigfillset(&set); //Adds all signals to the set
    sigdelset(&set, SIGALRM);// Removes SIGALRM from the set
    sigsuspend(&set); // suspend execution with a specified mask till a signal is recieved
}

int main(){
    int tested_signal = SIGINT; // Can be most signals. NOT SIGALRM, SIGSLEEP or SIGWAKE
    install_handler(tested_signal, signal_handler); // installs handler with sigaction
    test_signal_mask(tested_signal);
    install_handler(SIGALRM, alrm_signal_handler);  // installs handler with sigaction
    wait_for_alarm();
}
```

Your intern's code is supposed to always print 'Alarm caught!' and then terminate. However, after executing your intern's code, sometimes 'Alarm caught!' is correctly written to the terminal and exits; at other times, the program is terminated seemingly at random, and sometimes it hangs after printing 'Alarm caught!'."

What seems to be the issue(s) in the intern's code? **Explain and give the possible fix(es). You can assume that the code compiles and that the only signals blocked in the handlers are the signals that triggered them.**

3 Extending the Linux File System ext2 [24 Points]

We have seen that filesystems, such as ext2, are more complex than they initially appear. In this question, you will design an enhanced version of ext2 called *PennExt*, focusing on secure block management.

(a) Why Wipe Blocks? [3 Points]

Before you begin designing PennExt, it is crucial to consider the implications of leaving old data intact on newly allocated blocks. When implementing a kernel, your job isn't to just write a kernel that maintains processes and files, but also to ensure the security within processes and file accesses.

What security and privacy risks could arise if blocks are not wiped before being reallocated to a new file?

(b) Free Blocks [7 Points]

In a basic design, free blocks could be tracked with a bitmap spanning multiple blocks. However, you are restricted from introducing any new data structures and must rely on ext2's existing reserved inodes (10 in total, with the second inode always being the root).

How can you leverage these reserved inodes to manage free blocks? Where exactly would these block references be stored, and how would you maintain and update them?

(c) When Writing, How Long Do You Wait? [3 Points]

Files can be truncated or deleted in several ways, such as by calling `OPEN` with the `O_TRUNC` flag, which reduces a file's size by truncating it, or by using the `ftruncate` function, which you may not have encountered before.

In your design, blocks must be wiped using `erase(block_number)` before being returned to the free list. Since there is only a single structure tracking free blocks, blocks must be erased and re-added to the free list immediately; otherwise, they will be lost.

Assume that when a file is truncated (reducing its number of blocks) or deleted, its blocks are **immediately erased and returned to the free list**. **Why might this approach be suboptimal?**

(d) But, is there a solution? [9 Points]

You can mitigate the trade-off by using the kernel **when it is idle**. The kernel can execute a `synchronize()` function that synchronizes the file system so that all unused blocks that previously held file data are wiped and added to your free list design from **part (b).**

What would you need to add to your design in (part b) to make this `synchronize()` feasible? Explain how your approach alleviates the drawbacks from the previous question.

(e) How do we know what these inodes do? [2 Points]

Finally, before your new system design hits production, you need to indicate the functionality these inodes now implore. Where in the file system would you need to mark this change? **Briefly** explain your answer. *Hint: In which block?*

4 Threads are cooooooooooooool [10 Points]

Consider the following code.

```
int global_counter = 0;

void* thread_main(void* arg) {
  while (global_counter < 2) {
    global_counter += 1;
  }
  return NULL;
}
```

Assume that two threads are running the function thread_main and return from the function normally. Additionally, there are no hardware or compiler optimizations that occur.

(a) Minimum [4 Points]

What is the minimum value that global_counter can be after both threads finish? **Briefly justify your answer.**

(b) Maximum [6 Points]

What is the maximum value that global_counter can be after both threads finish? **Briefly justify your answer.**

5 I want to schedule more sleep time [30 Points]

Let's introduce a new scheduling algorithm, **Shortest Remaining Time First (SRTF)**, which is a preemptive version of **Shortest Job First (SJF)**.

Shortest Remaining Time First works as follows:

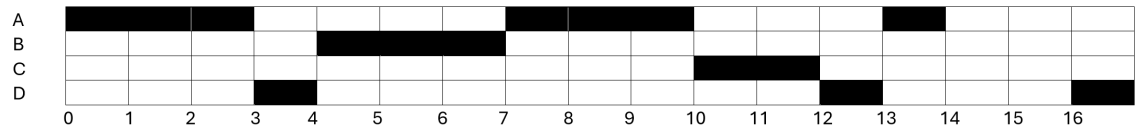
- The first task received becomes the current running task.
- The current running task is the only one that is making progress and thus it's "remaining time" gets smaller (since it is currently being executed)
- The current running task is pre-empted if a new task with shorter remaining time arrives. Otherwise, the run current task runs to completion or till it willingly gives up the CPU.

Consider the table of tasks below:

Task	Arrival Time	Run Time
A	0	7
B	2	3
C	6	2
D	1*	1*

Note that task D is a special "reoccurring" task. It runs for only 1 unit of time before it gets blocked on I/O (e.g. "finishes"). However, after 3 time units of not running it is received as a task again.

If we were to schedule this with the round robin algorithm with a quantum of 3, we get:

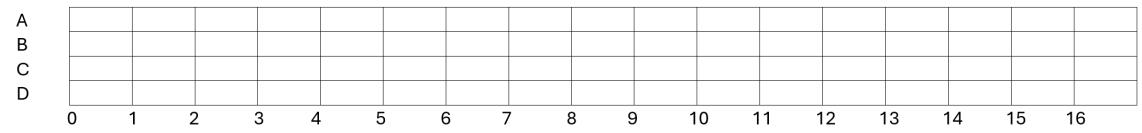


Note how there is a long time between the first time and second time task D ran; it simply became "*runnable*" after 3 time units of not running and was re-added to the queue until its turn.

(a) Let's Schedule It [20]

Schedule the same tasks that we ran with round robin, but this time with SRTF. You can make the following assumptions if needed:

- A task can be scheduled to run as soon as it arrives.
- If there are two or more tasks that the scheduler could run with the same remaining time left, then it chooses the one that comes first alphabetically (e.g. if A and C tied, A would run).



(b) Pros [5 Points]

In general (not specific to this scenario), what is one way SRTF is better than round robin? **Please justify your answer.**

(c) Cons [5 Points]

In general (not specific to this scenario), what is one reason SRTF may be worse than a round robin? **Please justify your answer.**

6 The Last Question [3 Points]

What is one bug that you and your partner got stuck on in Penn-Shell? What was the source of the bug? How did you fix it?

Did you learn anything from that bug? Or, you could just write (or draw) anything you want here. :)

Scratch Work Page