Course: Operating Systems - Semester 1 of 5786
Assignment 4

Directions

A. Due Date: 13 January 2026 at 11:55pm

B. Groups of up to two (2) students may submit this assignment.

C. Code for this assignment (Ass4) must be submitted via Github using the per-assignment private repository opened for you by GitHub Classroom. More details on the repository are found below.

D. There are 100 points total on this assignment.

E. GitHub has autograding tests included that run after each push. The tests and points are not complete, but do cover the basic input/output behavior of the programs you must write. Some test files are found in repo subdirectories in the repository. Do not remove or modify those files.

   (a) You can find the test scripts to test the input-output behavior of your tool in the starter code.

   (b) The last tests with Ctrl+C interruptions are in separate scripts. They work only on Linux.

   (c) Sample outputs for each input-output test are found on Moodle.

F. What to turn in:

   (a) Makefile, all source code (*.c, *.h) and library files necessary to compile and run your programs

   (b) README.md file with the contents described in Section 11.

G. **Do not** turn in:

   - Compiled executable files
   - Object files (.o)
   - Output files

H. The test scripts create output files in a directory called outfiles. If you do local testing, make sure that outfiles is not uploaded to your repository. To be certain, ensure your GitHub repository does not contain a directory called outfiles.

General Requirements

1. All of the code below must be written in C and compilable and executable in a standard Linux Ubuntu (14+) or Linux Mint (20+) environment.

2. All code must have comments - each function must have an introductory comment detailing its purpose, input parameters, and return value.

3. Use `malloc` or `calloc` for (at least) your buffer and array allocations. Don't forget to use `free` when you're done with them.

Signals, Threads, Mutual Exclusion, and Files

In this assignment you will use the Pthread library to write some user level thread code that uses monitors. The main functionality of the program imitates a map-reduce file length and disk usage tool (parallel read and synchronization barriers). The main goal of the assignment is to gain familiarity with the Pthread library for user threads, monitors, and synchronization barriers.

A secondary goal of this assignment is the use of signal handlers in C. We'll use one to experiment with how they work.

You can find help and additional references for Pthread, mutual exclusion, and monitors at the following links:

- `https://hpc-tutorials.llnl.gov/posix/`
- `https://man7.org/linux/man-pages/man7/pthreads.7.html` (or just `man 7 pthreads`)
- `https://pubs.opengroup.org/onlinepubs/7908799/xsh/pthread.h.html`

1 Makefile (7 points)

For this assignment, you will need to make a simple Makefile with the following targets:

1. all - the default rule that compiles the executable for the assignment. It should call the filesorter target.
2. filesorter - compiles the object files for the file type sorter program to an executable called `filesorter`
3. main.o - compiles the main.c file into a file called main.o
4. file-processor.o - compiles the file-processor.c file into a file called file-processor.o
5. clean - a rule that erases all temporary and output files generated during compilation. It must leave just the code files and the Makefile.

The name of the file must be `Makefile`

Note: The name of the file must be exactly as above. Do not add any suffixes or endings to the file. For example, do NOT call the file Makefile.txt.

Points

- all target. **2 points**
- filesorter target. **2 points**
- main.o target. **1 point**
- file-processor.o target. **1 point**
- clean target. **1 point**

Warning If the Makefile doesn't work, the autograder will not succeed in compiling the program and you will not receive any points!

2 Header Files

C uses header files to expose functionality between code files. The following header files are given as part of the starter code. They have function prototypes and struct definitions that I used in my solution. You can modify them as needed.

1. monitor.h - contains the definition for the monitor, including shared state and locks.

2. file-processor.h - contains the function prototype for the `runProcessor` function that the threads run.

3 Code Documentation (8 points)

Add documentation to all functions via a comment just above the function body in all C files. This includes all functions, including ones provided in the starter code.

The function header comment must be in the following format:

```
/* [name of function] [description of function's job]
*  Input: [parameters and what they mean]
*  Output: [meaning of value returned]
*/
```

For example, a header comment on a simple adding function might be:

```
/* Add: adds two integers
*  Input: num1 - first number, num2 - second number
*  Output: Sum of the two numbers
*/
```

4 Program Input

The user will provide the following parameters to the filesorter program:

- The directories with files to process. Must be 1 or more directories.

Some sample program inputs:

```
./filesorter dir1/
./filesorter dir1/ dir2/
./filesorter dir1/ dir2/ dir2/
```

Note that the same directory might be submitted multiples times. If a non-existent directory is given, it should be ignored in the inputs. If no directories are given or if only non-existent directories are given, the tool must exit with a usage message (see below for what to print).

5 File type, size, and block usage

The tool we're going to write will exercise some techniques and material we learned in class, but its primary goal is to demonstrate how mutual exclusion works across different threads in a single process. The tool will take one or more input directories and perform the following calculations:

1. Categorize and count the files in the directory and all subdirectories under it by file extension. The extension is the text found after the last period (.) in the file name. Files that have no period in their name should be grouped under a "No extension" category.
2. Count the total number of bytes taken up by all files in the directory and all subdirectories under it
3. Count the total number of file allocation blocks taken up by all files in the directory and all subdirectories under it.
   - As we learned in class, allocated blocks are a fixed size (512B in Linux) and may be full or only partly full. The physical allocation for a file is typically larger than the total occupied size of the file, but there are exceptions (look up "sparse files").

The last two elements can be retrieved using the `stat` structure and `lstat` function in C.

For instance, let's look at the following directory whose contents are shown using `ls -l`:

Listing 1: ls -l directory listing

```
-rw-r--r-- 1 mjmay mjmay 1023221 Dec 28 10:07 1076px-2003_AR_Proof.png
-rw-r--r-- 1 mjmay mjmay  237135 Dec 28 10:08 California_Colored.svg.png
-rw-r--r-- 1 mjmay mjmay     181 Dec 28 10:04 Flag_of_Alabama.svg
-rw-r--r-- 1 mjmay mjmay  102342 Dec 28 10:04 Seal_of_Alabama.svg
-rw-r--r-- 1 mjmay mjmay  102933 Dec 28 10:05 cover.jpg
drwxr-xr-x 2 mjmay mjmay    4096 Dec 28 10:12 dir11
-rw-r--r-- 1 mjmay mjmay  256090 Dec 28 10:04 pg11-images-kf8.mobi
-rw-r--r-- 1 mjmay mjmay  601848 Dec 28 10:02 pg46-h.zip
-rw-r--r-- 1 mjmay mjmay  641452 Dec 28 10:02 pg46-images-3.epub
```

The tool will process the directories provided in parallel using threads and take the results from all directories for a final printout. The final printout will have all file categories sorted by count (descending) and internally by alphabetical sorting (A→Z). For example, the result of the tool on the directory above will be:

Listing 2: Tool output for directory

```
Extension        | Count
----------------------------
JPG              | 2
PNG              | 2
SVG              | 2
EPUB             | 1
MOBI             | 1
TXT              | 1
ZIP              | 1
Total Files:                 10
Total Bytes:            3388773 bytes
Total Blocks:              6648 blocks
Physical Allocated:     3403776 bytes
```

The physical allocated size is precisely 512× the total blocks counted.

6 Mutual Exclusion: Monitor

The file processing will be performed using multiple threads that execute concurrently. The categories, category count, bytes, and blocks allocated must be stored in a data structure that is shared between the different threads. Therefore, to prevent race conditions and loss of data, you must use a monitor to control access to the shared data structure.

I have provided you with a monitor structure to use. The monitor has the following fields:

1. State fields - the linked list of file category information, the total byte count, and total block count.
2. Locks - two locks that you'll use to protect the contents of the monitor and state fields.
3. Barrier - the barrier that will force synchronization between threads every so often.

The monitor's definition is found in `monitor.h`. You should not need to modify it.

7 Files Processing Steps

The tool must process all of the files in the directories provided, assigning a separate thread for each directory given as a parameter. The threads must run in parallel. The files must be cataloged based on their extensions in the state fields of the monitor so they can be printed out at the end. Use `malloc` or `calloc` to allocate your state fields. While there is no guarantee how many files there will be in the test or their extensions, you may assume that each individual extension will be shorter than 200 characters.

As an exercise in synchronization, we will force the threads to synchronize after processing every 3 files (*e.g.,* 3, 6, 9, 12, etc).

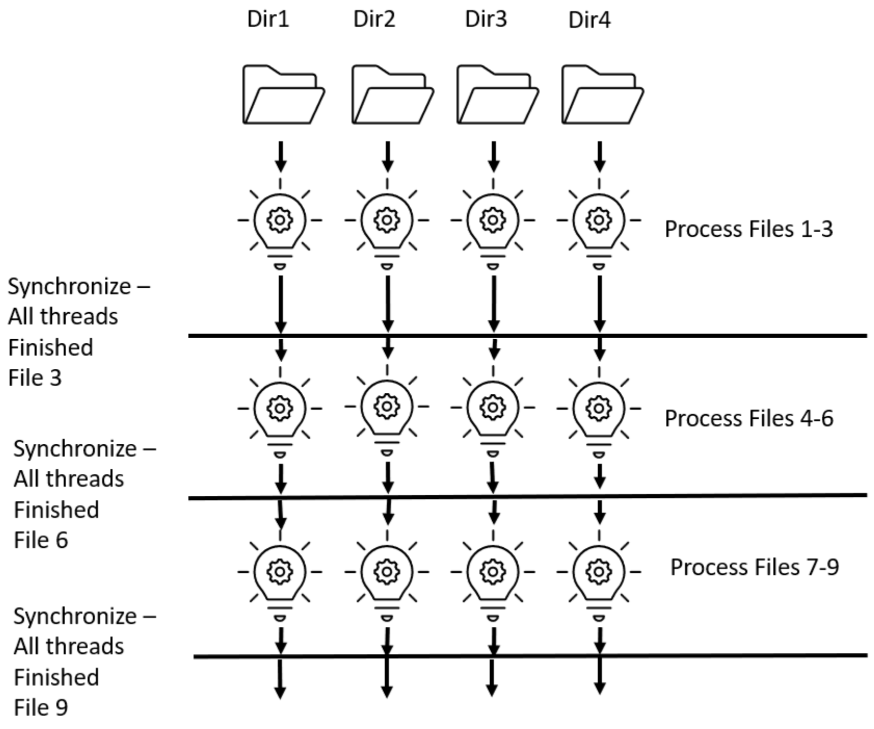
Dir1
Dir2
Dir3
Dir4
Process Files 1-3
Synchronize – All threads Finished File 3
Process Files 4-6
Synchronize – All threads Finished File 6
Process Files 7-9
Synchronize – All threads Finished File 9

Figure 1: File processing steps.

After processing 3 files, each thread must pause processing until all other threads have reached the end of their current set of 3. After all have finished their set of 3, the threads will proceed with the next set of 3 files. This process is shown in Figure 1. In the figure, there are four directories being processed. Each one starts processing files 1–3. When done, they all wait until all are ready to begin files 4–6. When done, they all wait until all are ready to begin files 7–9. The synchronization continues until all files are completed. For convenience, all test directories will be with an identical number of files.

When all threads have finished processing their chapters, the main thread resumes and outputs the resulting table of extensions, the total byte count, the total block count, and the total allocated size.

The file category table must be alphabetized within the file count. The total physical allocation is the total number of blocks 512× (each block in Linux is 512 bytes) Figure 2 shows the output final steps.

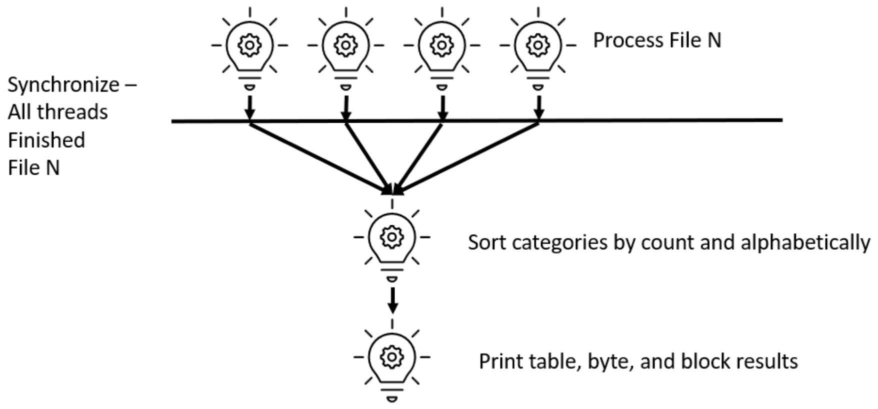
Process File N
Synchronize – All threads Finished File N
Sort categories by count and alphabetically
Print table, byte, and block results

Figure 2: Output final steps

7.1 Processing Example

Consider the following three directories:

Listing 3: ls -l dir-3

```
-rw-r--r-- 1 mjmay mjmay 15748 Dec 28 16:58 file1.pdf
-rw-r--r-- 1 mjmay mjmay 45469 Dec 28 16:58 file2.txt
-rw-r--r-- 1 mjmay mjmay 26448 Dec 28 16:59 file3.pdf
```

Listing 4: ls -l dir-4

```
-rw-r--r-- 1 mjmay mjmay 328070 Dec 28 17:00 file1.epub
-rw-r--r-- 1 mjmay mjmay 187469 Dec 28 17:00 file2.pdf
-rw-r--r-- 1 mjmay mjmay  46868 Dec 28 17:00 file3.zip
```

Listing 5: ls -l dir-5

```
-rw-r--r-- 1 mjmay mjmay  1346 Dec 28 17:01 file1.epub
-rw-r--r-- 1 mjmay mjmay 14701 Dec 28 17:01 file2.zip
-rw-r--r-- 1 mjmay mjmay  4117 Dec 28 17:01 file3.zip
```

Running the tool on the directories above will show the following final output:

Listing 6: Final output from running file sorter on directories

```
./filesorter dir-4/ dir-5/ dir-3/

Mon Dec 29 20:16:16 2025 [Thread 0] Scanned  3 files. Waiting...
Mon Dec 29 20:16:16 2025 [Thread 1] Scanned  3 files. Waiting...
Mon Dec 29 20:16:16 2025 [Thread 2] Scanned  3 files. Waiting...

Extension       | Count
--------------------------
PDF             | 3
ZIP             | 3
EPUB            | 2
TXT             | 1
Total Files:                  9
Total Bytes:             670236 bytes
Total Blocks:              1352 blocks
Physical Allocated:      692224 bytes
```

7.2 Race Conditions

We will avoid race conditions in the code by using locks for mutual exclusion on the bookkeeping variables. This is essential when adding files, bytes, or blocks to the shared state or searching for existing extensions.

8 Barrier Synchronization

The threads will wait for each other at the end of each set of 3 files using a reusable synchronization barrier from Pthreads using the `pthread_barrier_wait(pthread_barrier_t)` function . A synchronization barrier is an algorithm in which a fixed number of threads must arrive at a condition variable before any one of them can exit. The barrier must be configured with the expected number of threads so that it forces them to wait until that number of threads has reached the barrier. Once reached, all threads can proceed.

Implementing a simple synchronization barrier is not hard, but making a reusable one is tricky. Since we want to enable threads to enter (for instance) for files 1–3, go on to process files 4–6, and then wait again for files 7–9, we must prevent race conditions that might lead a thread getting ahead of others. For example, if a thread finished files 4–6, we would not want to let it move on to files 7–9 if the other threads are just getting started on file 4.

To do that, we will use the Pthreads barrier wait function mentioned above.

To make the synchronization clear during the output, each thread must output a line when it finishes scanning three files starts to wait at the barrier. The line must be prefixed with the date and time (use **strftime** with the **%c** flag). For instance, consider the following output from processing three directories with 10 files each:

Listing 7: Waiting updates from 3 directories

```
Sun Dec 28 22:42:06 2025 [Thread 1] Scanned  3 files. Waiting...
Sun Dec 28 22:42:06 2025 [Thread 0] Scanned  3 files. Waiting...
Sun Dec 28 22:42:08 2025 [Thread 0] Scanned  6 files. Waiting...
Sun Dec 28 22:42:08 2025 [Thread 1] Scanned  6 files. Waiting...
Sun Dec 28 22:42:10 2025 [Thread 1] Scanned  9 files. Waiting...
Sun Dec 28 22:42:10 2025 [Thread 0] Scanned  9 files. Waiting...
```

9 Signal Handling

To make things more interesting, we'll add a signal handler to the tool as well. Processing small directories will be pretty short, so we'll add 1 second sleep after processing each file to make it take a bit longer artificially. Since things will take a few seconds now, the user might want to know how many files have been processed so far without stopping the processing. To support that, add a signal handler that lets the user to press Ctrl+C to receive an update on the tool's progress. The update will show the total number of files processed so far, the total number of bytes counted so far, and the total number of allocated bytes so far.

The update message (*i.e.* signal handler) must output its text to STDERR. Be sure to use the correct output parameters when writing so that the regular output goes to STDOUT but the signal handler's output goes to STDERR.

9.1 Output Sample

Let's say we ran the tool on three directories with 10 files each. The tool's run time is 10 seconds. After about 2 seconds, the user presses Ctrl+C (SIGINT) and the tool outputs its current state. The user does this 2 more times, for a total of 3 times. The output to STDERR is as follows:

Listing 8: Interrupt status messages

```
--------- Status ---------
Files Scanned:      4
Logical Bytes:      1571319
Physical Allocated: 1576960

--------- Status ---------
Files Scanned:      8
Logical Bytes:      2265866
Physical Allocated: 2277376

--------- Status ---------
Files Scanned:      10
Logical Bytes:      2471141
Physical Allocated: 2486272
```

At each point, the signal handler outputs the total number of files, bytes, and allocated bytes counted so far. Despite the repeated Ctrl+C interruptions, the tool completes correctly, finally outputting the word list ordered as above.

Note: You may use **printf** and **fprintf** in your signal handler here, even though technically they are not safe for use in interrupt handlers like this. There is a small chance that you'll hit Ctrl+C while another thread is in the middle of running **printf** and you'll cause a deadlock since the functions are not reentrant.

See `https://man7.org/linux/man-pages/man7/signal-safety.7.html` for more details on this. You do not need to worry about that problem in this assignment, though.

10 Input Ouptput Tests (80 points)

The autograder will perform a series of tests on the tool and check its output. The expected outputs for the tests are given here. Sample output files are included on Moodle.

In all outputs shown with dates and times, the time and date should be adjusted as appropriate to the actual time. Since we are using multithreading, the precise interleaving of the threads will change between runs.

10.1 1 Directory with 10 files (6 points)

```
./filesorter dir1-10/
```

Expected output:

```
Mon Dec 29 13:36:15 2025 [Thread 0] Scanned  3 files. Waiting...
Mon Dec 29 13:36:17 2025 [Thread 0] Scanned  6 files. Waiting...
Mon Dec 29 13:36:19 2025 [Thread 0] Scanned  9 files. Waiting...

Extension       | Count
---------------------------
JPG             | 2
PNG             | 2
SVG             | 2
EPUB            | 1
MOBI            | 1
TXT             | 1
ZIP             | 1
Total Files:                 10
Total Bytes:            3388773 bytes
Total Blocks:              6648 blocks
Physical Allocated:     3403776 bytes
```

10.2 2 Directories with 10 files (6 points)

```
./filesorter dir1-10/ dir2-10/
```

Expected output:

```
Mon Dec 29 13:40:14 2025 [Thread 1] Scanned  3 files. Waiting...
Mon Dec 29 13:40:14 2025 [Thread 0] Scanned  3 files. Waiting...
Mon Dec 29 13:40:16 2025 [Thread 0] Scanned  6 files. Waiting...
Mon Dec 29 13:40:16 2025 [Thread 1] Scanned  6 files. Waiting...
Mon Dec 29 13:40:18 2025 [Thread 1] Scanned  9 files. Waiting...
Mon Dec 29 13:40:18 2025 [Thread 0] Scanned  9 files. Waiting...

Extension       | Count
---------------------------
JPG             | 6
SVG             | 5
MOBI            | 2
PNG             | 2
EPUB            | 1
MID             | 1
OGG             | 1
TXT             | 1
ZIP             | 1
Total Files:                 20
Total Bytes:            4613713 bytes
Total Blocks:              9080 blocks
Physical Allocated:     4648960 bytes
```

10.3 3 Directories with 10 files (6 points)

```
./filesorter dir1-10/ dir2-10/ dir3-10/
```

Expected output:

```
Mon Dec 29 13:40:21 2025 [Thread 1] Scanned  3 files. Waiting...
Mon Dec 29 13:40:21 2025 [Thread 0] Scanned  3 files. Waiting...
Mon Dec 29 13:40:21 2025 [Thread 2] Scanned  3 files. Waiting...
Mon Dec 29 13:40:23 2025 [Thread 1] Scanned  6 files. Waiting...
Mon Dec 29 13:40:23 2025 [Thread 0] Scanned  6 files. Waiting...
Mon Dec 29 13:40:23 2025 [Thread 2] Scanned  6 files. Waiting...
Mon Dec 29 13:40:25 2025 [Thread 2] Scanned  9 files. Waiting...
Mon Dec 29 13:40:25 2025 [Thread 1] Scanned  9 files. Waiting...
Mon Dec 29 13:40:25 2025 [Thread 0] Scanned  9 files. Waiting...

Extension        | Count
---------------------------
JPG              | 6
PNG              | 6
SVG              | 6
MOBI             | 3
TXT              | 3
MID              | 2
OGG              | 2
EPUB             | 1
ZIP              | 1
Total Files:                 30
Total Bytes:            7087395 bytes
Total Blocks:             13936 blocks
Physical Allocated:     7135232 bytes
```

10.4 3 Directories with 10 files, Repeated Dir (6 points)

```
./filesorter dir1-10/ dir2-10/ dir2-10/
```

Expected output:

```
Mon Dec 29 13:40:28 2025 [Thread 0] Scanned  3 files. Waiting...
Mon Dec 29 13:40:28 2025 [Thread 1] Scanned  3 files. Waiting...
Mon Dec 29 13:40:28 2025 [Thread 2] Scanned  3 files. Waiting...
Mon Dec 29 13:40:30 2025 [Thread 0] Scanned  6 files. Waiting...
Mon Dec 29 13:40:30 2025 [Thread 2] Scanned  6 files. Waiting...
Mon Dec 29 13:40:30 2025 [Thread 1] Scanned  6 files. Waiting...
Mon Dec 29 13:40:32 2025 [Thread 1] Scanned  9 files. Waiting...
Mon Dec 29 13:40:32 2025 [Thread 2] Scanned  9 files. Waiting...
Mon Dec 29 13:40:32 2025 [Thread 0] Scanned  9 files. Waiting...

Extension        | Count
---------------------------
JPG              | 10
SVG              | 8
MOBI             | 3
MID              | 2
OGG              | 2
PNG              | 2
EPUB             | 1
TXT              | 1
ZIP              | 1
Total Files:                 30
Total Bytes:            5838653 bytes
Total Blocks:             11512 blocks
Physical Allocated:     5894144 bytes
```

10.5 4 Directories with 10 files (7 points)

```
./filesorter dir1-10/ dir2-10/ dir3-10/ dir4-10/
```

Expected output:

```
Mon Dec 29 13:40:35 2025 [Thread 0] Scanned  3 files. Waiting...
Mon Dec 29 13:40:35 2025 [Thread 2] Scanned  3 files. Waiting...
Mon Dec 29 13:40:35 2025 [Thread 1] Scanned  3 files. Waiting...
Mon Dec 29 13:40:35 2025 [Thread 3] Scanned  3 files. Waiting...
Mon Dec 29 13:40:37 2025 [Thread 0] Scanned  6 files. Waiting...
Mon Dec 29 13:40:37 2025 [Thread 2] Scanned  6 files. Waiting...
Mon Dec 29 13:40:37 2025 [Thread 3] Scanned  6 files. Waiting...
Mon Dec 29 13:40:37 2025 [Thread 1] Scanned  6 files. Waiting...
Mon Dec 29 13:40:39 2025 [Thread 1] Scanned  9 files. Waiting...
Mon Dec 29 13:40:39 2025 [Thread 2] Scanned  9 files. Waiting...
Mon Dec 29 13:40:39 2025 [Thread 3] Scanned  9 files. Waiting...
Mon Dec 29 13:40:39 2025 [Thread 0] Scanned  9 files. Waiting...

Extension       | Count
---------------------------
JPG             | 11
SVG             | 7
PNG             | 6
EPUB            | 3
MOBI            | 3
TXT             | 3
ZIP             | 3
MID             | 2
OGG             | 2
Total Files:                  40
Total Bytes:             9947473 bytes
Total Blocks:              19568 blocks
Physical Allocated:     10018816 bytes
```

10.6 1 Directory with 20 files (8 points)

```
./filesorter dir5-20/
```

Expected output:

```
Mon Dec 29 13:40:42 2025 [Thread 0] Scanned  3 files. Waiting...
Mon Dec 29 13:40:44 2025 [Thread 0] Scanned  6 files. Waiting...
Mon Dec 29 13:40:46 2025 [Thread 0] Scanned  9 files. Waiting...
Mon Dec 29 13:40:48 2025 [Thread 0] Scanned  12 files. Waiting...
Mon Dec 29 13:40:50 2025 [Thread 0] Scanned  15 files. Waiting...
Mon Dec 29 13:40:52 2025 [Thread 0] Scanned  18 files. Waiting...

Extension       | Count
---------------------------
JPG             | 5
PNG             | 4
SVG             | 3
EPUB            | 2
TXT             | 2
MID             | 1
MOBI            | 1
OGG             | 1
ZIP             | 1
Total Files:                  20
Total Bytes:             4738531 bytes
Total Blocks:               9328 blocks
Physical Allocated:      4775936 bytes
```

10.7 3 Directories with 20 files (8 points)

```
./filesorter dir5-20/ dir6-20/ dir7-20/
```

Expected output:

```
Mon Dec 29 13:40:56 2025 [Thread 0] Scanned  3 files. Waiting...
Mon Dec 29 13:40:56 2025 [Thread 1] Scanned  3 files. Waiting...
Mon Dec 29 13:40:56 2025 [Thread 2] Scanned  3 files. Waiting...
Mon Dec 29 13:40:59 2025 [Thread 2] Scanned  6 files. Waiting...
Mon Dec 29 13:40:59 2025 [Thread 1] Scanned  6 files. Waiting...
Mon Dec 29 13:40:59 2025 [Thread 0] Scanned  6 files. Waiting...
Mon Dec 29 13:41:01 2025 [Thread 0] Scanned  9 files. Waiting...
Mon Dec 29 13:41:01 2025 [Thread 2] Scanned  9 files. Waiting...
Mon Dec 29 13:41:01 2025 [Thread 1] Scanned  9 files. Waiting...
Mon Dec 29 13:41:03 2025 [Thread 2] Scanned  12 files. Waiting...
Mon Dec 29 13:41:03 2025 [Thread 1] Scanned  12 files. Waiting...
Mon Dec 29 13:41:03 2025 [Thread 0] Scanned  12 files. Waiting...
Mon Dec 29 13:41:05 2025 [Thread 0] Scanned  15 files. Waiting...
Mon Dec 29 13:41:05 2025 [Thread 1] Scanned  15 files. Waiting...
Mon Dec 29 13:41:05 2025 [Thread 2] Scanned  15 files. Waiting...
Mon Dec 29 13:41:07 2025 [Thread 0] Scanned  18 files. Waiting...
Mon Dec 29 13:41:07 2025 [Thread 1] Scanned  18 files. Waiting...
Mon Dec 29 13:41:07 2025 [Thread 2] Scanned  18 files. Waiting...

Extension        | Count
---------------------------
JPG              | 14
PNG              | 12
SVG              | 9
EPUB             | 6
TXT              | 6
ZIP              | 4
MID              | 3
MOBI             | 3
OGG              | 3
Total Files:                   60
Total Bytes:             14730699 bytes
Total Blocks:               28984 blocks
Physical Allocated:      14839808 bytes
```

10.8 1 Directory with Ctrl+C Interrupt (8 points)

```
./filesorter dir1-10/
Ctrl+C
```

The timing of the Ctrl+C signal will affect the precise output of the interrupt handler. One output (skipping the summary table) might be:

```
Mon Dec 29 14:27:05 2025 [Thread 0] Scanned  3 files. Waiting...

--------- Status ---------
Files Scanned:      4
Logical Bytes:      1582777
Physical Allocated: 1585152

Mon Dec 29 14:27:07 2025 [Thread 0] Scanned  6 files. Waiting...
Mon Dec 29 14:27:09 2025 [Thread 0] Scanned  9 files. Waiting...
```

10.9 3 Directories with 2 Ctrl+C Interrupts (8 points)

```
./filesorter dir1-10/ dir2-10/ dir3-10/
Ctrl+C
Ctrl+C
```

The timing of the Ctrl+C signal will affect the precise output of the interrupt handler. One output (skipping the summary table) might be:

```
--------- Status ---------
Files Scanned:      6
Logical Bytes:      1845292
Physical Allocated: 1855488

Mon Dec 29 14:33:26 2025 [Thread 0] Scanned  3 files. Waiting...
Mon Dec 29 14:33:26 2025 [Thread 2] Scanned  3 files. Waiting...
Mon Dec 29 14:33:26 2025 [Thread 1] Scanned  3 files. Waiting...

--------- Status ---------
Files Scanned:      15
Logical Bytes:      3241095
Physical Allocated: 3264512

Mon Dec 29 14:33:28 2025 [Thread 0] Scanned  6 files. Waiting...
Mon Dec 29 14:33:28 2025 [Thread 2] Scanned  6 files. Waiting...
Mon Dec 29 14:33:28 2025 [Thread 1] Scanned  6 files. Waiting...
Mon Dec 29 14:33:30 2025 [Thread 2] Scanned  9 files. Waiting...
[...]
```

10.10 3 Directories with 3 Ctrl+C Interrupts (8 points)

```
./filesorter dir5-20/ dir6-20/ dir7-20/
Ctrl+C
Ctrl+C
Ctrl+C
```

The timing of the Ctrl+C signal will affect the precise output of the interrupt handler. One output (skipping the summary table) might be:

```
Mon Dec 29 14:34:37 2025 [Thread 1] Scanned  3 files. Waiting...
Mon Dec 29 14:34:37 2025 [Thread 0] Scanned  3 files. Waiting...
Mon Dec 29 14:34:37 2025 [Thread 2] Scanned  3 files. Waiting...

--------- Status ---------
Files Scanned:      12
Logical Bytes:      3027755
Physical Allocated: 3047424

Mon Dec 29 14:34:39 2025 [Thread 1] Scanned  6 files. Waiting...
Mon Dec 29 14:34:39 2025 [Thread 0] Scanned  6 files. Waiting...
Mon Dec 29 14:34:39 2025 [Thread 2] Scanned  6 files. Waiting...

--------- Status ---------
Files Scanned:      21
Logical Bytes:      5263349
Physical Allocated: 5300224

--------- Status ---------
Files Scanned:      24
Logical Bytes:      5652383
Physical Allocated: 5693440

Mon Dec 29 14:34:41 2025 [Thread 2] Scanned  9 files. Waiting...
[...]
```

10.11 No directories (3 points)

Tests that the tool outputs a usage message if no directories are given.

```
./filesorter
```

Expected output:

```
Usage: ./filesorter <dir1> ...
```

10.12 Non-existent directories (3 points)

Tests that the tool outputs a usage message if an invalid directory is given.

```
./filesorter fake_dir1/
```

Expected output:

```
Usage: ./filesorter <dir1> ...
```

10.13 Existing and Non-existent directories together (3 points)

Tests that the tool ignores any invalid directories given.

```
./filesorter dir1-10/ fake-dir2/
```

Expected output:

```
Mon Dec 29 14:42:41 2025 [Thread 0] Scanned  3 files. Waiting...
Mon Dec 29 14:42:43 2025 [Thread 0] Scanned  6 files. Waiting...
Mon Dec 29 14:42:45 2025 [Thread 0] Scanned  9 files. Waiting...

Extension       | Count
---------------------------
JPG             | 2
PNG             | 2
SVG             | 2
EPUB            | 1
MOBI            | 1
TXT             | 1
ZIP             | 1
Total Files:                 10
Total Bytes:            3388773 bytes
Total Blocks:              6648 blocks
Physical Allocated:     3403776 bytes
```

11 Using Git and GitHub (5 points)

Submit all of your code for the homework using the private repository for your work on GitHub. You must perform the following actions on the repository:

1. Each team member must make at least 1 substantive commit of code or text to the repository.
2. The team must make **at least** two (2) commits to the repository and add a comment to each. Write good commit messages, following the instructions at this tutorial: `https://www.theodinproject.com/lessons/foundations-commit-messages`
3. The last commit must include the comment "Submitted for grading"
4. The repository must include a README.md file with the following details:
   - Student names and IDs
   - Assignment name (Assignment 4)
   - Course name (Operating Systems)
   - Semester (A/B) and Year

- Number of hours spent on the assignment total.

Repositories missing any of the above actions will be penalized.