System Programming
Homework 5 - Spring 2023

Barış Ayyıldız

Program Structure:

The program consists of several data structures, including structs for thread arguments (**ThreadArgs**), file information (**Files**), and a task queue (**TaskQueue**). It also defines several utility functions and signal handlers.

Data Types:

a. ThreadArgs struct:
    source[FILE_NAME_LENGTH] (char): Represents the source directory path.
    destination[FILE_NAME_LENGTH] (char): Represents the destination directory path.
b. Files struct:
    source_fd (int): File descriptor of the source file.
    destination_fd (int): File descriptor of the destination file.
    source_file_name[FILE_NAME_LENGTH] (char): Full path of the source file.
    destination_file_name[FILE_NAME_LENGTH] (char): Full path of the destination file.
c. TaskQueue struct:
    buffer (Files*): Pointer to an array of Files structs representing the task queue.
    counter (int): Number of elements currently in the task queue.
    cap (int): Capacity of the task queue.
    mutex (pthread_mutex_t): Mutex for thread synchronization.
    full (pthread_cond_t): Condition variable for signaling when the task queue is full.
    empty (pthread_cond_t): Condition variable for signaling when the task queue is empty.

Thread Communication and Synchronization

The program utilizes pthreads for multi-threading and implements synchronization mechanisms such as mutexes and condition variables to ensure safe access to shared resources. The task queue acts as a buffer, facilitating communication between the producer and consumer threads.

Functions:

destroyTaskQueue

Description: Initializes the task queue by allocating memory for the buffer, setting the initial counter and capacity, and initializing the mutex and condition variables.
Steps:
    Allocate memory for the task queue buffer using malloc().
    Set the counter to 0.
    Set the capacity to the provided buffer size.
    Initialize the mutex using pthread_mutex_init().
    Initialize the full condition variable using pthread_cond_init().

Initialize the empty condition variable using pthread_cond_init().

sigintHandler

Description: Handles the SIGINT signal (interrupt signal, typically generated by pressing Ctrl+C) and terminates the program gracefully.
Steps:

- Print a termination message.
- Exit the program using exit().
- Parameters: sig_num (int) - The signal number received.
- Return Type: void

initTaskQueue

Description: Initializes the task queue by allocating memory for the buffer, setting the initial counter and capacity, and initializing the mutex and condition variables.
Steps:

- Allocate memory for the task queue buffer using malloc().
- Set the counter to 0.
- Set the capacity to the provided buffer size.
- Initialize the mutex using pthread_mutex_init().
- Initialize the full condition variable using pthread_cond_init().
- Initialize the empty condition variable using pthread_cond_init().
- Parameters: buffer_size (int) - The size of the task queue buffer.
- Return Type: void

produce

Description: This function is executed by producer threads and recursively traverses the source directory, enqueueing files and directories for copying.
Steps:

- Open the source directory using opendir().
- Create the destination directory using mkdir() (if it doesn't already exist).
- Iterate over the entries in the source directory using readdir().
- If the entry is a subdirectory, create a new producer thread to handle the recursion and enqueue the subdirectory.
- If the entry is a file, open the source and destination files using open() with appropriate flags.
- Lock the task queue mutex using pthread_mutex_lock().
- Enqueue the file information into the task queue.
- Signal the consumer threads that the task queue is no longer empty using pthread_cond_signal().
- Unlock the task queue mutex using pthread_mutex_unlock().
- Set the finished flag to indicate that the producer thread has finished processing.
- Close the source directory using closedir().

consume

Description: This function is executed by consumer threads and dequeues files from the task queue, copying their contents to the destination.
Steps:

- Lock the task queue mutex using pthread_mutex_lock().
- If the task queue is empty and the producer threads have finished, signal the other consumer threads and exit the function.
- While the task queue is empty and the producer threads are still running, wait for the signal that the task queue is no longer empty using pthread_cond_wait().
- Dequeue a file from the task queue.
- Signal the producer threads that the task queue is no longer full using pthread_cond_signal().
- Unlock the task queue mutex using pthread_mutex_unlock().
- Read from the source file using read() and write to the destination file using write().
- Close the source and destination files using close().
- Increment the count of files copied.
- Repeat the above steps until all files have been processed.

main

Description: The main function initializes the program, creates the necessary threads, and measures the execution time.
Steps:

- Validate the command-line arguments.
- Set up the signal handler for SIGINT using signal().
- Initialize the task queue using initTaskQueue().
- Create the producer thread using pthread_create() and start the producer function produce().
- Create the consumer threads using pthread_create() and start the consumer function consume().
- Wait for the consumer threads to finish using pthread_join().
- Calculate and display the execution time using gettimeofday().
- Destroy the task queue using destroyTaskQueue().

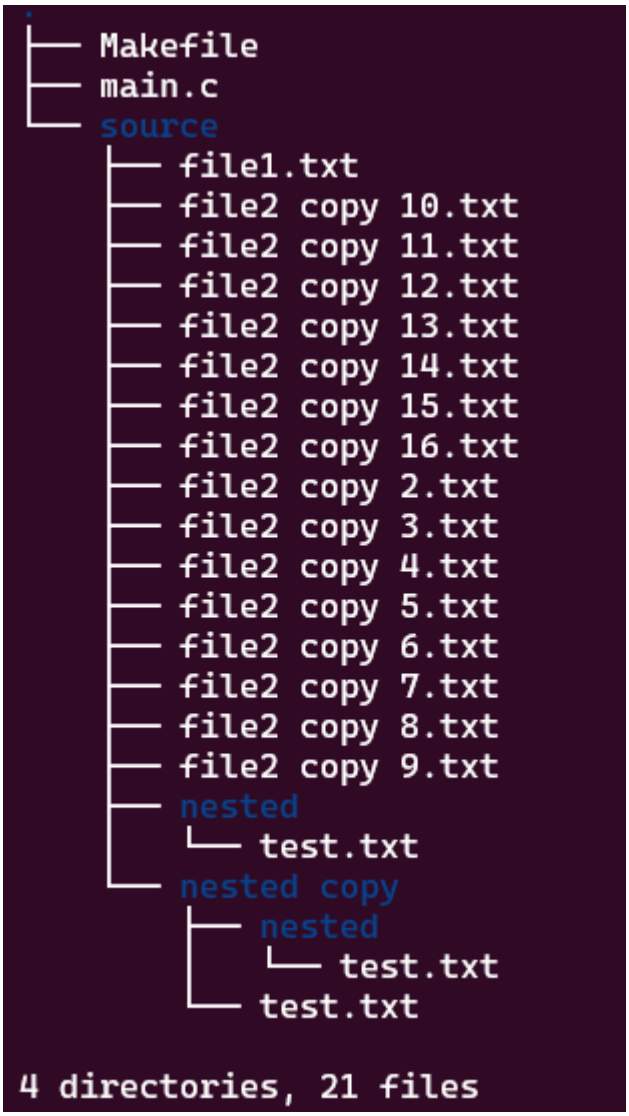
Algorithm

- The program follows a producer-consumer model, where producer threads recursively traverse the source directory and enqueue files and directories for copying, while consumer threads dequeue files from the task queue and copy their contents to the destination directory.
- The producer threads use a depth-first search approach to traverse the source directory and enqueue files and directories.
- The consumer threads use a synchronized task queue to ensure safe access to shared resources.

- The program utilizes mutexes and condition variables for thread synchronization and communication.
- The producer threads enqueue file information into the task queue and signal the consumer threads when the queue is no longer empty.
- The consumer threads dequeue files from the task queue, copy their contents to the destination directory, and signal the producer threads when the queue is no longer full.
- The program terminates gracefully when all producer and consumer threads have finished their tasks or when interrupted by the SIGINT signal.

Test Cases

This is my file structure before running the program



```
Results....
Number of files copied : 19
Total time taken: 0.024046 seconds
barisayyildiz@DESKTOP-2V8A48Q:~/system_hw5$
```

This is the file structure now

```
barisayyildiz@DESKTOP-2V8A48Q:~/system_hw5$ tree
.
├── Makefile
├── destination
│   ├── file1.txt
│   ├── file2 copy 10.txt
│   ├── file2 copy 11.txt
│   ├── file2 copy 12.txt
│   ├── file2 copy 13.txt
│   ├── file2 copy 14.txt
│   ├── file2 copy 15.txt
│   ├── file2 copy 16.txt
│   ├── file2 copy 2.txt
│   ├── file2 copy 3.txt
│   ├── file2 copy 4.txt
│   ├── file2 copy 5.txt
│   ├── file2 copy 6.txt
│   ├── file2 copy 7.txt
│   ├── file2 copy 8.txt
│   ├── file2 copy 9.txt
│   ├── nested
│   │   └── test.txt
│   └── nested copy
│       ├── nested
│       │   └── test.txt
│       └── test.txt
├── main
├── main.c
└── source
    ├── file1.txt
    ├── file2 copy 10.txt
    ├── file2 copy 11.txt
    ├── file2 copy 12.txt
    ├── file2 copy 13.txt
    ├── file2 copy 14.txt
    ├── file2 copy 15.txt
    ├── file2 copy 16.txt
    ├── file2 copy 2.txt
    ├── file2 copy 3.txt
    ├── file2 copy 4.txt
    ├── file2 copy 5.txt
    ├── file2 copy 6.txt
    ├── file2 copy 7.txt
    ├── file2 copy 8.txt
    ├── file2 copy 9.txt
    ├── nested
    │   └── test.txt
    └── nested copy
        ├── nested
        │   └── test.txt
        └── test.txt

8 directories, 41 files
```

Here are some of the results I received after running the program with different number of consumers and buffer sizes with the same dataset

Buffer size : 5, number of consumers : 3

Buffer size : 10, number of consumers : 3

Buffer size : 5, number of consumer : 5

Buffer size : 5, number of consumers : 10

How to run:

make

This will compile main.c

./main 10 5 source destination

This will copy all the files and subfolder under source folder to destination with buffer size 10 and number of consumers 5

rm main

This will remove the executable main