CSE344 – System Programming

HW5 – Report

Ömer Faruk Bitikçioğlu

161044010

Introduction

The project aims to copy files from a source directory to a destination directory using multiple threads. The program consists of a producer thread responsible for opening files and passing them to a buffer, and multiple consumer threads that copy the files in the buffer from the source directory to the destination directory.

Code Overview

The code begins with the inclusion of necessary header files and the declaration of required data structures, variables, mutexes, semaphores, and statistics counters.

The file_t structure is defined to hold information about opened file descriptors and file names. It has the following members:

- fileName: a character array to store the name of the file.
- srcFd: an integer representing the source file descriptor.
- destFd: an integer representing the destination file descriptor.

The code uses the following mutexes to provide thread synchronization:

- bufferMutex: a mutex used to protect access to the buffer.
- flagMutex: a mutex used to protect access to the producerDone flag.
- printMutex: a mutex used to protect print statements.
- statMutex: a mutex used to protect access to the statistics counters.

The code also uses the following semaphores to control the buffer capacity:

- semEmpty: a semaphore representing the number of available empty slots in the buffer.
- semFull: a semaphore representing the number of filled slots in the buffer.

The statistics counters are used to keep track of the number of files copied, the number of bytes copied, the number of regular files copied, and the number of FIFOs (named pipes) copied.

The main function is responsible for parsing command-line arguments, setting up signal handling, initializing mutexes and semaphores, creating threads, joining threads, printing statistics, and cleaning up resources.

The sigint_handler function is a signal handler for the SIGINT signal (interrupt signal). It sets the stopFlag variable to 1, indicating that the program should stop. It is called when the user presses Ctrl+C to terminate the program.

The program is divided into two main threads: the producer thread and the consumer threads.

Producer Thread

The producer thread is responsible for opening files in the source directory and passing them to the buffer. It receives arguments from the main function, which include the source directory name, destination directory name, and the number of consumer threads.

The openFiles function is a helper recursive function called by the producer thread. It opens the source and destination directories and iterates over the directory entries. For each entry, it checks the type of the file (regular file, directory, or FIFO), and performs the following actions:

If the entry is a directory, it recursively calls the openFiles function with the subdirectories' paths.

If the entry is a FIFO, it extracts the mode of the FIFO, creates the same FIFO in the destination folder, and increments the numOfFifoCopied counter.

If the entry is a regular file, it opens the source file for reading and the destination file for writing. Then, it creates a file_t structure, fills it with the file information, adds it to the buffer, and increments the numOfRegularFileCopied counter.

The producer thread starts by calling the openFiles function to open files in the source directory. If there is an error in opening the files, the stuck consumer threads are woken up by calling sem_post(&semFull), and the producer thread exits.

After opening the files, the producer thread sets the producerDone flag to 1, indicating that it has finished producing files. This is done inside a critical section protected by the flagMutex.

If the producer thread successfully finishes opening a file and add it to the buffer, it signals the consumer threads to wake up and start processing the file in the buffer. This is done by calling sem_post(&semFull) for each consumer thread.

Consumer Threads

The consumer threads are responsible for copying the contents of the files in the buffer from the source directory to the destination directory. Each consumer thread runs an infinite loop until the producerDone flag is set to 1 and the buffer is empty. Inside the loop, a consumer thread does the following:

1. It waits for a file to be available in the buffer by calling sem_wait(&semFull).
2. Once a file is available, it acquires the bufferMutex to protect access to the buffer.
3. It retrieves the file item from the buffer and releases the bufferMutex.
4. It opens the the retrieved file item’s source and destination files using the file descriptors stored in the file_t structure.
5. It then proceeds to read the contents of the source file and write them to the destination file.
6. After copying the file contents, it closes the source and destination files and updates the statistics counters inside a critical section protected by the statMutex.
7. It prints a message indicating the successful copy operation, again inside a critical section protected by the printMutex.
8. It repeats the loop to process the next file.

Main Function

The main function starts by parsing the command-line arguments to obtain the source directory, destination directory, and the number of consumer threads.

Next, it sets up the signal handler for the SIGINT signal using the sigaction() function, which associates the sigint_handler function with the SIGINT signal.

It then initializes the mutexes and semaphores using the corresponding initialization functions provided by the pthread library.

Next, the producer thread is created using the pthread_create() function, passing the producer function as the thread function and the necessary arguments.

The consumer threads are created using a loop, where each thread is created by calling pthread_create() and passing the consumer function as the thread function.

The main function then waits for the producer thread to finish by calling pthread_join(). If the producer thread encounters an error in opening the files, it wakes up the consumer threads by calling sem_post(&semFull), and the main function exits.

After the producer thread finishes, the main function waits for each consumer thread to finish by calling pthread_join() in a loop.

Once all threads have finished, the main function prints the statistics counters, cleans up the resources by freeing buffer, destroying the mutexes and semaphores, and exits.

Tests

A directory containing ~975MB of data is copied to the destination with different buffer sizes and different number of consumer threads. Results are shown below:

Buffer size: 1, Consumer thread: 1

```
Elapsed time: 36 seconds and 869541 microseconds
Number of files copied: 49 (974566050 bytes)
        49 regular file
        0 FIFO
```

Buffer size: 5, Consumer thread: 1

```
Elapsed time: 36 seconds and 511190 microseconds
Number of files copied: 49 (974566050 bytes)
        49 regular file
        0 FIFO
```

Buffer size: 5, Consumer thread: 5

```
Elapsed time: 58 seconds and 226767 microseconds
Number of files copied: 49 (974566050 bytes)
        49 regular file
        0 FIFO
```

Buffer size: 5, Consumer thread: 10

```
Elapsed time: 65 seconds and 94213 microseconds.
Number of files copied: 49 (974566050 bytes)
        49 regular file
        0 FIFO
```

Buffer size: 10, Consumer thread: 5

```
Elapsed time: 38 seconds and 970065 microseconds
Number of files copied: 49 (974566050 bytes)
        49 regular file
        0 FIFO
```

Buffer size: 20, Consumer thread: 5

```
Elapsed time: 34 seconds and 599678 microseconds
Number of files copied: 49 (974566050 bytes)
        49 regular file
        0 FIFO
```

Buffer size: 50, Consumer thread: 10

```
Elapsed time: 31 seconds and 139905 microseconds
Number of files copied: 49 (974566050 bytes)
        49 regular file
        0 FIFO
```

Buffer size: 50, Consumer thread: 5

```
Elapsed time: 30 seconds and 954381 microseconds
Number of files copied: 49 (974566050 bytes)
        49 regular file
        0 FIFO
```

Buffer size: 100, Consumer thread: 5

```
Elapsed time: 26 seconds and 343437 microseconds
Number of files copied: 49 (974566050 bytes)
        49 regular file
        0 FIFO
```

Buffer size: 100, Consumer thread: 20

```
Elapsed time: 32 seconds and 517475 microseconds
Number of files copied: 49 (974566050 bytes)
        49 regular file
        0 FIFO
```

Buffer size: 200, Consumer thread: 5

```
Elapsed time: 34 seconds and 660261 microseconds
Number of files copied: 49 (974566050 bytes)
        49 regular file
        0 FIFO
```

Buffer size: 200, Consumer thread: 10

```
Elapsed time: 28 seconds and 611090 microseconds
Number of files copied: 49 (974566050 bytes)
        49 regular file
        0 FIFO
```

Buffer size: 1000, Consumer thread: 10

```
Elapsed time: 36 seconds and 206010 microseconds
Number of files copied: 49 (974566050 bytes)
        49 regular file
        0 FIFO
```

Buffer size: 1000, Consumer thread: 50

```
Elapsed time: 42 seconds and 377435 microseconds
Number of files copied: 49 (974566050 bytes)
        49 regular file
        0 FIFO
```

Conclusion:

Incrementing the buffer size to 100, reduced the execution time. When the buffer size incremented to 1000 the difference is gone.

When we increment the number of consumer threads, especially above the buffer size, elapsed time is increased.

I think the optimum buffer size/number of consumer is around 20, and the buffer size should be around 100 for this execution.

Incrementing the buffer size improved the performance since the producer thread will wait less than before to produce more data.

When the number of consumer threads are increased the synchronization is taking more time and reduces the positive effect of having multiple threads. Also the hardware should support the thread count to effectively execute it on multiple cores.