GTU Department of Computer Engineering
CSE 344 – Spring 2023
Homework #5 Report

Hasan Mutlu
1801042673

Implementation Details and General Workflow

The main program firstly parses and checks the input. Then, creates required resources and creates the producer thread. After that, it creates the consumer threads and goes on to wait until the producer thread finishes. After producer thread finishes, it sets the done flag to 1 and posts the consumer semaphore for all consumers to terminate, in case some of them are waiting for a file to handle. After that, it prints required information and terminates.

The producer and consumer threads makes use of a struct called FileEntry, defined in FileEntryQueue.h file. This struct holds information of names and file descriptors of source and destination files. The producer thread scans the source directory and handles its files one by one. If there is a subdirectory, it calls itself with the specified subdirectory's data. If it is a file, it puts the file in FileEntry queue.

The FileEntry queue is a queue implemented as fixed size array and act as the buffer. Its capacity is fixed to buffer size. This size is never violated during the program run.

The consumer thread checks the FileEntry queue and handles the next file copy job.

Synchronization

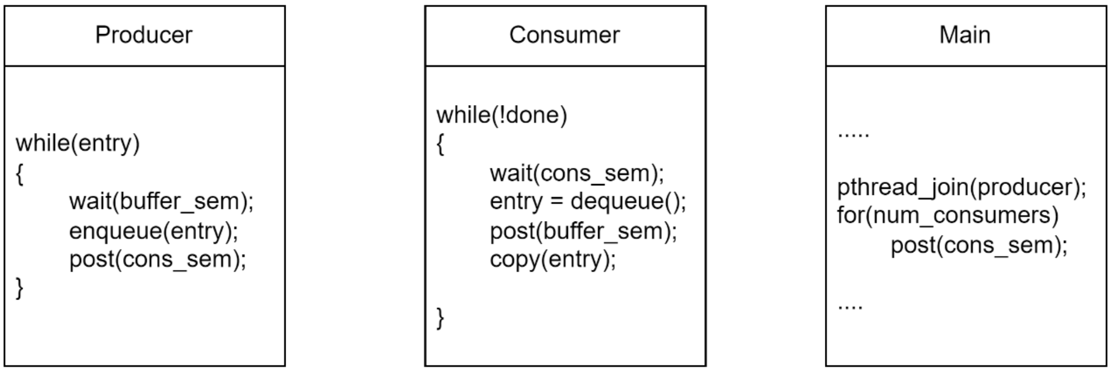
The syncronization is established using two semaphores. One is buffer_semaphore and the other is consumer_semaphore.

The buffer semaphore is initialized with the value of buffer size. The producer thread waits on this semaphore before putting the next file to be copied into the buffer. When a consumer takes a file from the buffer, it posts this semaphore. In this way, the producer is blocked when the buffer is full and allowed to put a new entry to the buffer when there is space available.

The consumer semaphore on the other hand is used for waking up consumers when there is a new entry available in the buffer. It is initilized with the value 0 and consumers are waiting on it before handling an entry from the buffer. The producer posts this semaphore whenever it added a new entry to the buffer and one consumer is woken up to handle this entry.

Also, there are two kinds of mutexes to prevent race conditions and protect critical sections. In this program, the buffer is a critical section used by multiple threads so modifying buffer (dequeue, enqueue) is protected using a mutex. Also, there might be race conditions between threads to print to standart output so it is prevented by using another mutex for each print call.

The general synchronization can be summarized in the example table below:



Producer

```
while(entry)
{
        wait(buffer_sem);
        enqueue(entry);
        post(cons_sem);
}
```

Consumer

```
while(!done)
{
        wait(cons_sem);
        entry = dequeue();
        post(buffer_sem);
        copy(entry);

}
```

Main

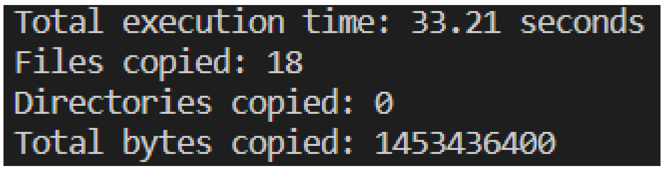
```
.....

pthread_join(producer);
for(num_consumers)
        post(cons_sem);

....
```

Test and Experiment Results

The program is tested with different directories. All files in all sub directories are successfully copied.

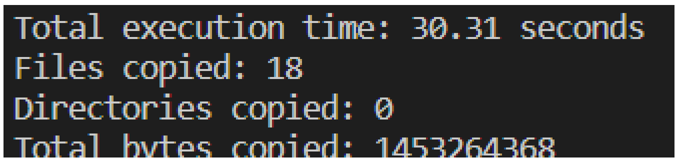
The program is experimented with different different buffer sizes and different number of consumer threads. In test scenario, a directory with 18 files of ~80MB size is used. Since my system has 8 cores, all experiments are run with at least 8 consumers. Here are the results:

- Buffer size = 50
- Num consumers = 50



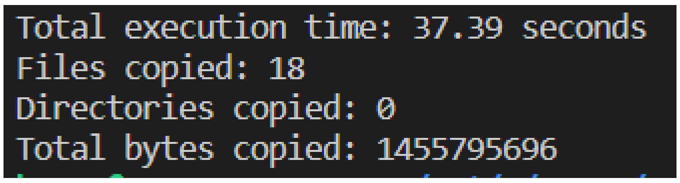
```
Total execution time: 33.21 seconds
Files copied: 18
Directories copied: 0
Total bytes copied: 1453436400
```

- Buffer size = 8
- Num consumers = 18



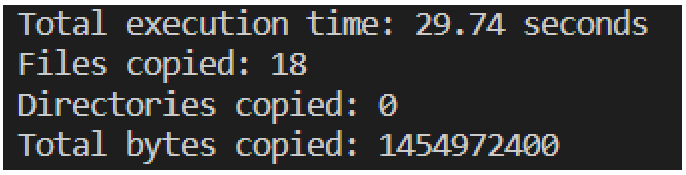
```
Total execution time: 30.31 seconds
Files copied: 18
Directories copied: 0
Total bytes copied: 1453264368
```

- Buffer size = 18
- Num consumers = 8



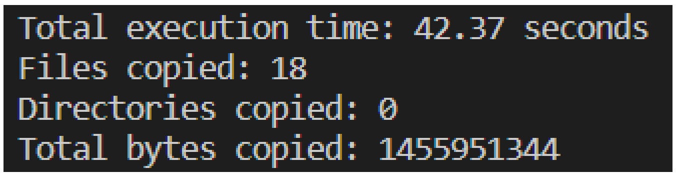
```
Total execution time: 37.39 seconds
Files copied: 18
Directories copied: 0
Total bytes copied: 1455795696
```

- Buffer size = 18
- Num consumers = 18



```
Total execution time: 29.74 seconds
Files copied: 18
Directories copied: 0
Total bytes copied: 1454972400
```

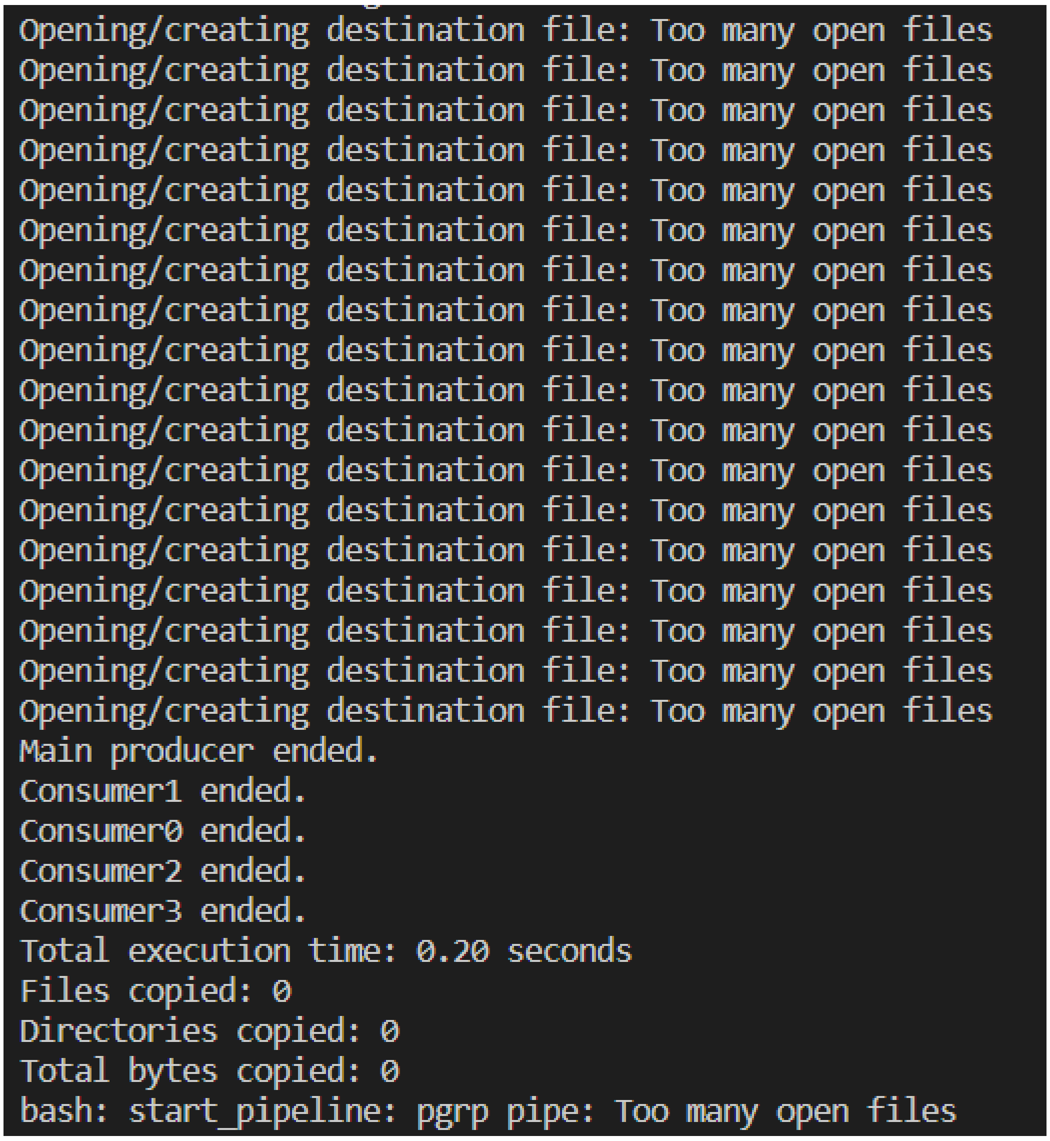
- Buffer size = 4
- Num consumers = 8



```
Total execution time: 42.37 seconds
Files copied: 18
Directories copied: 0
Total bytes copied: 1455951344
```

Conclusion: It is observed that the best result is obtained when buffer size and number of consumers are equal to the number of files. Also, number of consumers has more impact on performance since they are the ones that do the main job, copying. Buffer size doesn’t have a huge impact after some point since whatever the buffer size is, the consumers immediately handle entries from the buffer and the buffer is never filled unless number of consumers is smaller.

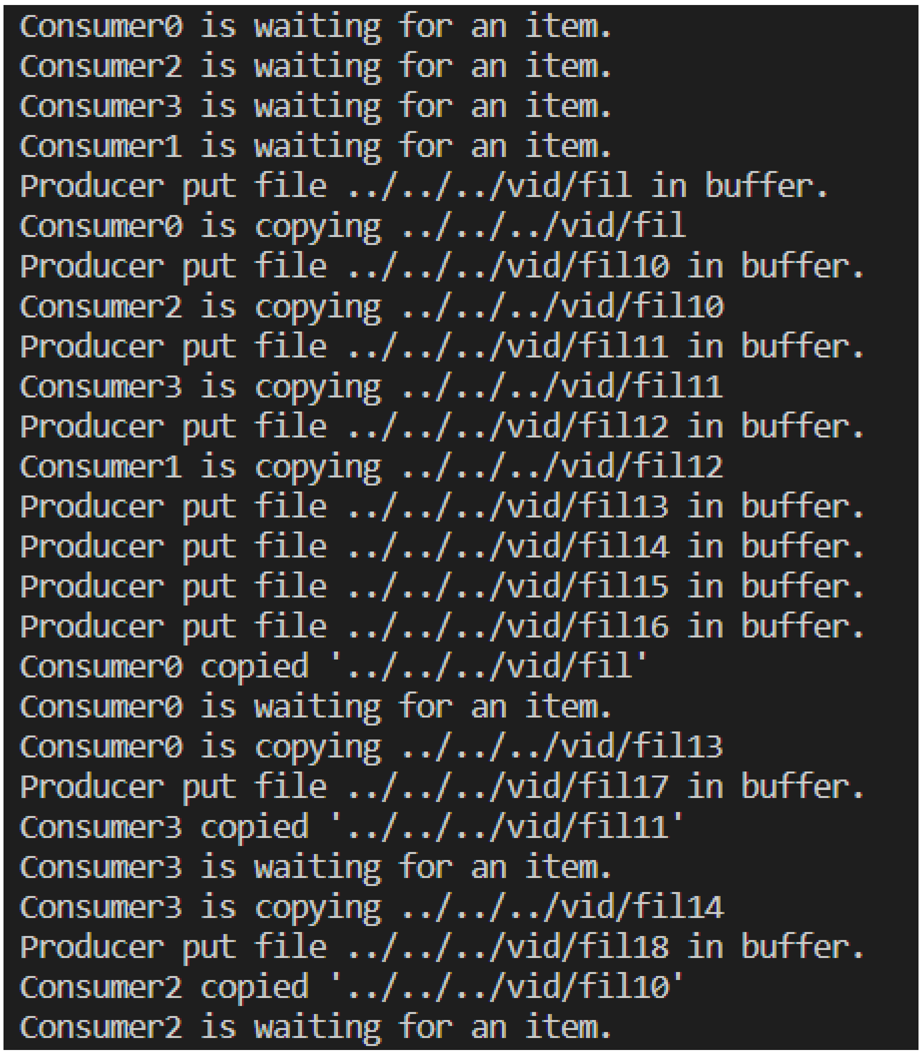
Exceeding the File Descriptor Limit:



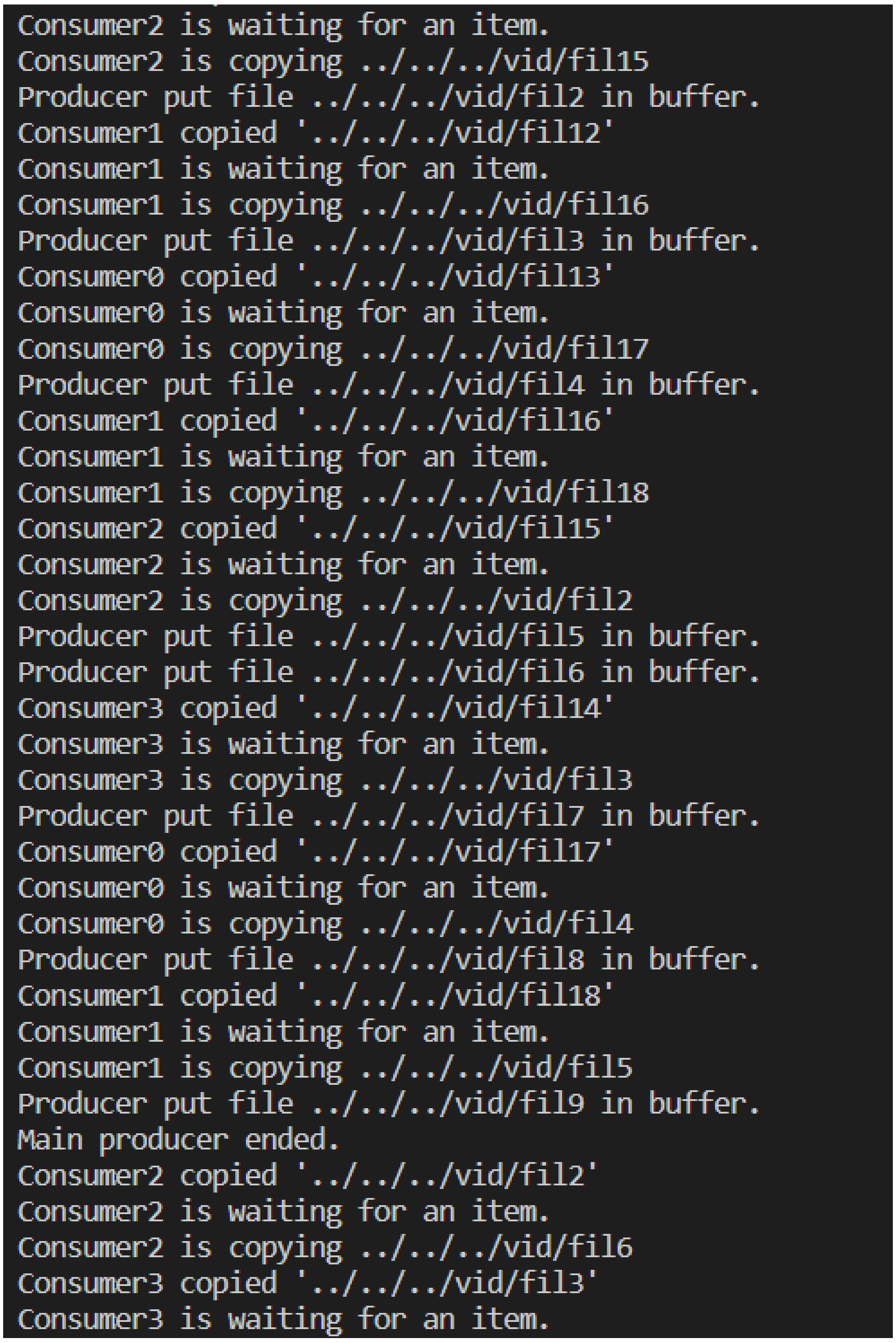
```
Opening/creating destination file: Too many open files
Opening/creating destination file: Too many open files
Opening/creating destination file: Too many open files
Opening/creating destination file: Too many open files
Opening/creating destination file: Too many open files
Opening/creating destination file: Too many open files
Opening/creating destination file: Too many open files
Opening/creating destination file: Too many open files
Opening/creating destination file: Too many open files
Opening/creating destination file: Too many open files
Opening/creating destination file: Too many open files
Opening/creating destination file: Too many open files
Opening/creating destination file: Too many open files
Opening/creating destination file: Too many open files
Opening/creating destination file: Too many open files
Opening/creating destination file: Too many open files
Opening/creating destination file: Too many open files
Opening/creating destination file: Too many open files
Main producer ended.
Consumer1 ended.
Consumer0 ended.
Consumer2 ended.
Consumer3 ended.
Total execution time: 0.20 seconds
Files copied: 0
Directories copied: 0
Total bytes copied: 0
bash: start_pipeline: pgrp pipe: Too many open files
```

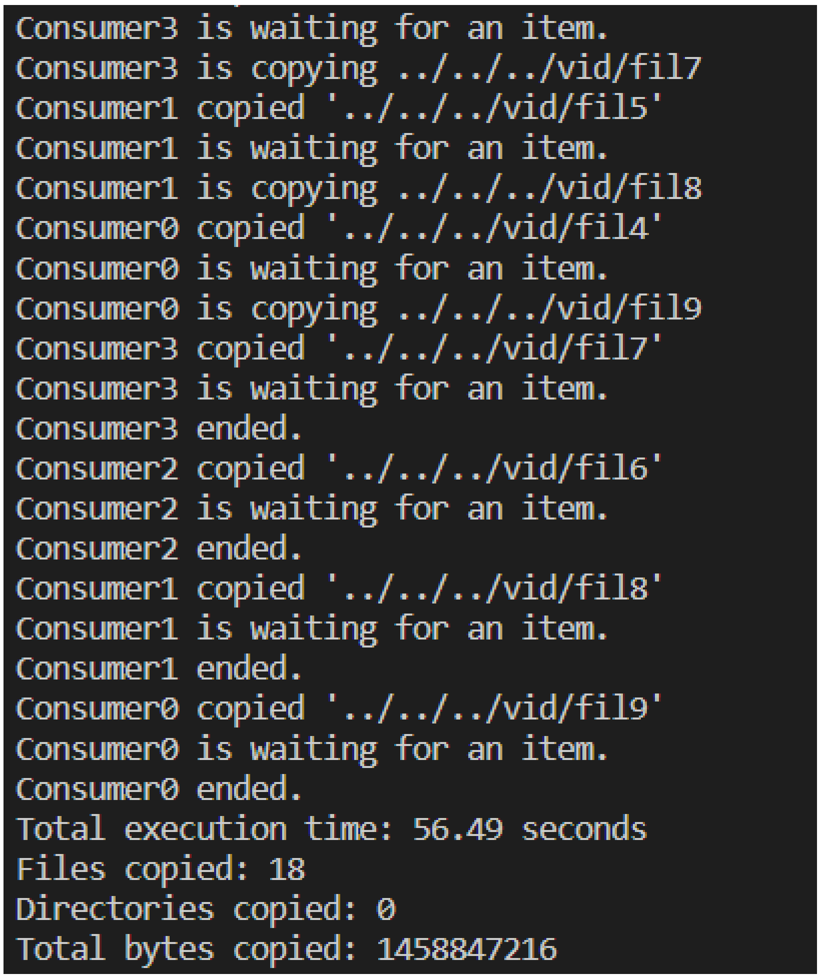
When open file descriptor limit is exceeded, an error message is printed for each file that couldn’t open and the program terminates.

Full Example Output:



```
Consumer0 is waiting for an item.
Consumer2 is waiting for an item.
Consumer3 is waiting for an item.
Consumer1 is waiting for an item.
Producer put file ../../../vid/fil in buffer.
Consumer0 is copying ../../../vid/fil
Producer put file ../../../vid/fil10 in buffer.
Consumer2 is copying ../../../vid/fil10
Producer put file ../../../vid/fil11 in buffer.
Consumer3 is copying ../../../vid/fil11
Producer put file ../../../vid/fil12 in buffer.
Consumer1 is copying ../../../vid/fil12
Producer put file ../../../vid/fil13 in buffer.
Producer put file ../../../vid/fil14 in buffer.
Producer put file ../../../vid/fil15 in buffer.
Producer put file ../../../vid/fil16 in buffer.
Consumer0 copied '../../../vid/fil'
Consumer0 is waiting for an item.
Consumer0 is copying ../../../vid/fil13
Producer put file ../../../vid/fil17 in buffer.
Consumer3 copied '../../../vid/fil11'
Consumer3 is waiting for an item.
Consumer3 is copying ../../../vid/fil14
Producer put file ../../../vid/fil18 in buffer.
Consumer2 copied '../../../vid/fil10'
Consumer2 is waiting for an item.
```


```
Consumer2 is waiting for an item.
Consumer2 is copying ../../../vid/fil15
Producer put file ../../../vid/fil2 in buffer.
Consumer1 copied '../../../vid/fil12'
Consumer1 is waiting for an item.
Consumer1 is copying ../../../vid/fil16
Producer put file ../../../vid/fil3 in buffer.
Consumer0 copied '../../../vid/fil13'
Consumer0 is waiting for an item.
Consumer0 is copying ../../../vid/fil17
Producer put file ../../../vid/fil4 in buffer.
Consumer1 copied '../../../vid/fil16'
Consumer1 is waiting for an item.
Consumer1 is copying ../../../vid/fil18
Consumer2 copied '../../../vid/fil15'
Consumer2 is waiting for an item.
Consumer2 is copying ../../../vid/fil2
Producer put file ../../../vid/fil5 in buffer.
Producer put file ../../../vid/fil6 in buffer.
Consumer3 copied '../../../vid/fil14'
Consumer3 is waiting for an item.
Consumer3 is copying ../../../vid/fil3
Producer put file ../../../vid/fil7 in buffer.
Consumer0 copied '../../../vid/fil17'
Consumer0 is waiting for an item.
Consumer0 is copying ../../../vid/fil4
Producer put file ../../../vid/fil8 in buffer.
Consumer1 copied '../../../vid/fil18'
Consumer1 is waiting for an item.
Consumer1 is copying ../../../vid/fil5
Producer put file ../../../vid/fil9 in buffer.
Main producer ended.
Consumer2 copied '../../../vid/fil2'
Consumer2 is waiting for an item.
Consumer2 is copying ../../../vid/fil6
Consumer3 copied '../../../vid/fil3'
Consumer3 is waiting for an item.
```



```
Consumer3 is waiting for an item.
Consumer3 is copying ../../../vid/fil7
Consumer1 copied '../../../vid/fil5'
Consumer1 is waiting for an item.
Consumer1 is copying ../../../vid/fil8
Consumer0 copied '../../../vid/fil4'
Consumer0 is waiting for an item.
Consumer0 is copying ../../../vid/fil9
Consumer3 copied '../../../vid/fil7'
Consumer3 is waiting for an item.
Consumer3 ended.
Consumer2 copied '../../../vid/fil6'
Consumer2 is waiting for an item.
Consumer2 ended.
Consumer1 copied '../../../vid/fil8'
Consumer1 is waiting for an item.
Consumer1 ended.
Consumer0 copied '../../../vid/fil9'
Consumer0 is waiting for an item.
Consumer0 ended.
Total execution time: 56.49 seconds
Files copied: 18
Directories copied: 0
Total bytes copied: 1458847216
```