CSE 344 - Homework 5
Sena Özbelen - 1901042601

`./pCp buffersize #consumers source destination (buffer size in bytes)`

`This creates a directory in destination even if destination exists or not. source -> destination/source (the same behavior as cp -R when destination` **`does exist`**`)`

`Valgrind might say sem_open is still reachable even though semaphore is` **`closed and unlinked.`**

We are asked to implement a file coping utility. This program consists of 2 programs: consumer.c and producer.c. All common variables and libraries are located in common.h

common.h

The buffer is located in a shared memory which is accessible by producer and consumer.

```
struct shm_entry
{
    int fd1; // read fd
    int fd2; // write fd
    char path[PATH_LEN]; // the exact path
    int valid; // flag, 1 when it is not empty, 0 when it is empty
};
```

This is the main structure of the shared memory. By using the size of this struct, the max entry that the buffer can hold is calculated. Using max entry number, the exact size of shared memory is calculated and created.

producer.c

Producer calculates max entry and size of the shared memory at first. Then, it creates a struct to send paths as parameters to producer thread.

```
struct param
{
    char path1[PATH_LEN];
    char path2[PATH_LEN];
};
```

Path2 should be changed since the given destination path doesn't consists of the directory's name.
destination -> destination/source(only the name of the target directory using get_dirname)

In copy_op function, it opens the directories first (if not available, then create it). It also creates/opens (if already created) shared memory and truncates it (if not truncated before). With mapping, it accesses the shared memory by using entries object. In a while loop, it checks if the buffer is already full by comparing number of current entries and max entry number. If it is not full, then it gets the item from the source directory. It uses dirent.h library for directory operations. readdir function is used to get every item in the directory.

It checks if the item is a directory. If so, then it writes all entries to the buffer and calls consumer to copy these entries. After copying, it calls copy_op recursively. For new directory, the operations above are applied. It opens the shared memory for this new directory and it starts to traverse all items in this directory. If this item is not a directory but a fifo, then it creates a fifo file using mkdir (it removes if there is a fifo having the same name before creating). If it is not a fifo, then it opens the file for reading and writing. These information is stored in an entry (shm_entry) and it is written to shared memory. When the buffer is full, it calls consumer to handle file operations. While consumer handles them, producer sleeps using a semaphore.

Not to exceed the open file limit, it checks if the limit is exceeded or not because it fails when the program exceeds the limit

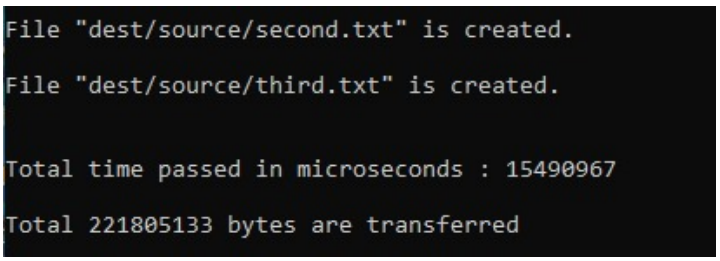
consumer.c

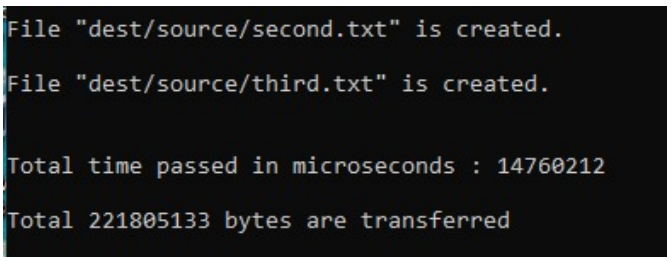
When consumer is called, it opens the shared memory and creates a semaphore for stdout. It starts to send entries to consumers. For each consumer, it assigns an entry and consumers copy the assigned file. Since there are limited consumers, the whole buffer might not be handled at the same time. After consumers completes copy operations, it checks if there are remaining entries. If so, then it assigns them to consumers until no entry is left. After copying all files in the shared memory, producer is waken up and producer continues to fill the buffer. Since they don't work at the same time, there is no need for shared memory protection. However, stdout and num_bytes are protected by a mutex.

Signal handling is done by using a flag. The producer code is in a while loop. There is also another while loop to get all items in the directory. These loops are controlled by a flag "terminate". Since it has to break the loop even if signal is not caught, after every item is traversed by the producer, it sets terminate to 1. However, this is a recursive function so when it goes back to the outer directory, the flag remains 1. I reset it when it comes back from recursive call for inner directory so that it remains 1 only in the first directory and finishes the program. I use signal_flag not to set terminate to 0 since a signal is caught.

Test Cases

Buffer: 1000 ~ 3 entries per buffer, Consumers: 1 (left) and 3 (right)





Thread number affects the passed time since some of copying processes are done in parallel.

Buffer: 10000 ~ approx. 30 entries per buffer, Consumers: 1 (first row left) and 10 (first row right) and 20 (second row left) and 30 (second row right)

```
File "dest/source/second.txt" is created.

File "dest/source/third.txt" is created.


Total time passed in microseconds : 14913794

Total 221805133 bytes are transferred
```

```
File "dest/source/second.txt" is created.

File "dest/source/third.txt" is created.


Total time passed in microseconds : 14366772

Total 221805133 bytes are transferred
```

```
File "dest/source/second.txt" is created.

File "dest/source/third.txt" is created.


Total time passed in microseconds : 15036341

Total 221805133 bytes are transferred
```

```
File "dest/source/second.txt" is created.

File "dest/source/third.txt" is created.


Total time passed in microseconds : 14961613

Total 221805133 bytes are transferred
```

Increasing the size doesn't have a huge effect on time. Increasing threads doesn't always give a better result. It should be optimal number. In this case, thread number 10 gives the best result rather than 20 or 30.

CTRL-C Case

```
File "dest/source/sc - Copy/OneDrive_1_4-26-2023/Week05.pdf" is created.
^C
SIGINT is received. Terminating!

File "dest/source/sc - Copy/OneDrive_1_4-26-2023/Week06.pdf" is created.

File "dest/source/sc - Copy/OneDrive_1_4-26-2023/Week07.pdf" is created.

File "dest/source/sc - Copy/OneDrive_1_4-26-2023/Week08.pdf" is created.


Total time passed in microseconds : 5449238

Total 82959039 bytes are transferred
```

SIGINT/SIGTERM handling is done by terminate flag and it is in producer code so before terminating the program, it waits for consumer to finish its job.