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

Mehmet Mete Şamlıoğlu
200104004093

Summarization of general operation

In this homework, our task was to perform copy operations by using multithreading, there will be two thread functions to achieve goals. The first one is the **producer,** which finds all the files in a given source path and redirects those files to the **consumer**. The consumer will copy the given source to the destination by writing the bytes one by one. The main goal is to perform this simple operation by using multithreading. There will be one thread for the producer and the rest of them are going to be competing for the consumer. In the consumer function, there will be threads as as the user desires and they will perform the copying operation simultaneously. To control each thread and prevent **DEADLOCK** I used semaphores. Whenever the producer gets the other source it will wait in the sem_wait state until one of the threads is available. When the thread in consumer is available it will pass through the sem_wait state and if the done variable is not one it is going to take the FileEntry from the buffer and it will send the source and destination to copy function which is the function where copying operation is handled. So basically, the producer finds the files, directories, and FIFOs iterating the entry or recursively and when any thread that computes in the consumer is available it will pass the entry to the consumer. This will continue until either a signal is caught or the producer is joined.

I will explain each function one by one to make everything more clear;

void signal_handler(int signal)

The signal_handler function is a signal handler that is responsible for catching and handling various signals. The function takes an input parameter signal which represents the signal that triggered the handler. Inside the function, there is a switch statement that checks the value of the signal to determine which signal was caught I just handled SIGTERM SIGINT SIGUSR1 and SIGUSR2.

For each signal case, a corresponding message is printed to indicate that the signal was caught. For example, if the signal is SIGINT, the function prints "SIGINT signal is caught."After printing the signal message, the function continues to execute and prints a general message indicating that the program is releasing all resources. The function then pauses for 2 seconds using the sleep() function. Finally, the variable done is set to 1, indicating that the program should be terminated. Additionally, the function broadcasts the producerSignal using pthread_cond_broadcast() to notify all threads that they should terminate. In summary, the signal_handler function serves as a callback for signal handling. It prints a message corresponding to the caught signal, releases resources, sets a termination flag, and broadcasts a signal to notify threads to terminate.

int is_directory(const char* directory_path)

I wrote this function as a helper function for the producer thread, whenever the file entry is a directory, it must check the destination path and see if the file already exists there. Otherwise, it has to create a directory to perform the operation. So it basically checks if the parameter is a directory or not, if it is a directory it returns one if not it returns 0.

void copy(const char* path_source, const char* path_destination)

The copy function is responsible for copying the contents of one file to another file. The copy function takes two input parameters; the source_path representing the path of the source file and the destination_path representing the path of the destination file. Inside the function, it opens the source file using the open system call with the O_RDONLY flag, which opens the file in read-only mode. It checks if the source file was successfully opened. If not, it prints an error message and returns. It opens the destination file using the open system call with the O_WRONLY | O_CREAT | O_TRUNC flags. It opens the file in write-only mode, creates the file if it doesn't exist, and truncates the file to zero length if it already exists. It checks if the destination file was successfully opened. If not, it prints an error message, closes the source file, and returns. The function uses a buffer (temporary storage) to read a chunk of data from the source file and write it to the destination file repeatedly until the entire file is copied. It reads a chunk of data from the source file using the read system call. If there is no more data to read (reached end-of-file), it breaks out of the loop. It writes the data read from the source file to the destination file using the write. If there is an error while writing to the destination file, it prints an error message and breaks out of the loop. The function continues reading and writing data until the entire file is copied. Finally, it closes both the source and destination files using the close system call to release the file descriptors and free up system resources. In summary, the copy function opens the source and destination files, reads data from the source file, and writes the data to the destination file until the entire file is copied. It handles errors during file opening and writing and then closes the files when finished. It also increases total_number_of_bytes to print in the end as the output of the copying operation.

void* producer(void* args)

The producer function is responsible for finding files in a directory and adding them to a buffer for further use in the consumer function. The producer function takes two input parameters source_dir representing the source directory path and destination_dir representing the destination directory path .Inside the function, it opens the source directory using the opendir function to start traversing its contents. It checks if the source directory was successfully opened. If not, it prints an error message and returns. The function then starts looping through the directory entries using the readdir function to read each entry one by one. For each entry, it checks if it is a regular file or a directory. If it is a regular file, it creates a FilePath structure to store the source path, destination path, and file descriptors. After that, it finds a slot in the buffer by using semaphores and mutex synchronization mechanisms to ensure thread safety. Once I put the slot in the buffer, it adds the FileEntry to the buffer and increments the count of files in the buffer. If it is a directory, it creates the corresponding destination directory as was mentioned in the PDF. It then creates a new thread (a sub-thread) to recursively process the sub-directory by passing the source and destination paths as arguments and it basically finds all the files in subdirectories and copies them to the destination one by one. The function waits for the sub-thread to finish using the pthread_join function, which blocks until the sub-thread completes its execution. Finally, it closes the source directory using the closedir function to join the pthread_join which is waiting in main function.

So basically, the producer function opens a source directory, iterates its contents, and either adds regular files to a buffer for further processing in the consumer thread or creates sub-threads to process sub-directories recursively. It uses semaphore synchronization mechanisms to ensure thread safety and closes the source directory when finished.

void* consumer(void* args)

The consumer function takes a single argument args, which represents the index of the consumer thread.Inside the function, it starts an infinite loop that continues until a termination condition is satisfied, which is a signal and a variable that will be sent from the main whenever the producer thread is joined or a signal is caught. It waits for a signal from the semFull semaphore, which indicates that the producer is found a path and put that in the buffer for the consumer to receive. If the termination signal done is set to 1, the thread is going to be terminating, the loop breaks to exit the function. It unlocks the lock on the mutexBuffer mutex to ensure exclusive access to the buffer. It retrieves the last file entry from the buffer by accessing the element at index **count - 1**. It decrements the count variable to remove the consumed file entry from the buffer. It releases the lock on the mutexBuffer mutex to allow other threads to access the buffer and perform the copy operation. It prints a message to indicate the successful copying of the file, displaying the source and destination paths. It calls the copy function to copy the file from the source path to the destination path. It increments the total_copied_files counter to keep track of the total number of files copied. It signals the semEmpty semaphore to indicate that a slot in the buffer has been freed and lets other threads know that they can get other FileEntries from the buffer to process the file. The loop continues to the next iteration to consume more files if available. If a termination signal is received, the loop breaks to exit the function. The function returns NULL to indicate the completion of the thread. In summary, the consumer function continuously waits for files in the buffer, consumes them, copies them to the destination path, and updates the necessary counters. It ensures synchronization using semaphores and mutexes to handle concurrent access to the buffer.

Lastly let’s see what happens in the main;

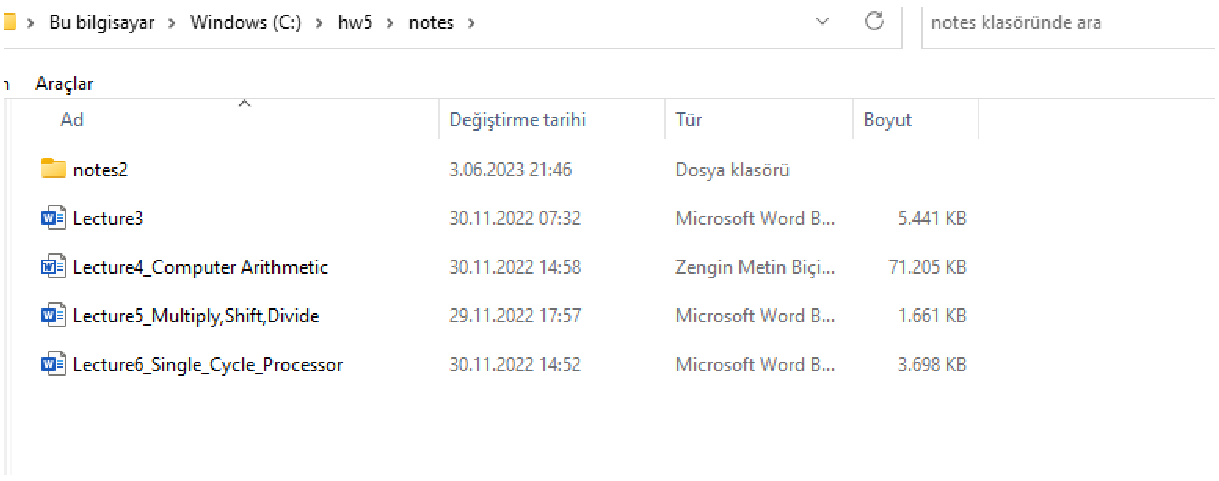
main function

The main function starts by checking if at least four parameters are provided as command-line arguments. If not, it prints an error message and returns with an error code. It initializes the mutex mutexBuffer and condition variable producerSignal using the respective initialization functions. It declares variables and retrieves command-line arguments, such as the source directory and destination directory. It sets up signal handlers for the SIGINT, SIGTERM, SIGUSR1, and SIGUSR2 signals. The signal_handler function is responsible for handling these signals.It converts the buffer size and consumer count from command-line arguments to integers.It calculates the total number of threads in the thread pool, which includes the producer thread and the consumer threads. It allocates memory for the buffer array based on the buffer size. It initializes semaphores semEmpty and semFull with appropriate initial values. It checks if the destination directory already exists. If not, it creates the destination directory using the mkdir function. It records the start time of the copying process using gettimeofday. It creates the threads in the thread pool. The first thread created is the producer thread, which invokes the producer function. The remaining threads are consumer threads, each invoking the consumer function.It waits for all threads in the thread pool to join using the pthread_join function. It prints appropriate messages to indicate the joining of threads. It records the end time of the copying process using gettimeofday. It calculates the total time taken for the copying process. It prints the final statistics, including the total time taken, the number of bytes read, the number of files copied, the number of directories copied, and the number of FIFOs copied. It cleans up by destroying the mutex and condition variable, destroying the semaphores, freeing the buffer memory, and returning 0 to indicate successful execution.

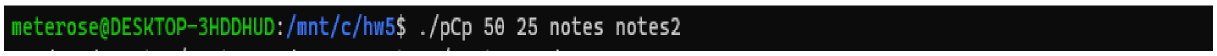
Running Commands

Example 1

So I will try to copy the following directory to another directory by creating;



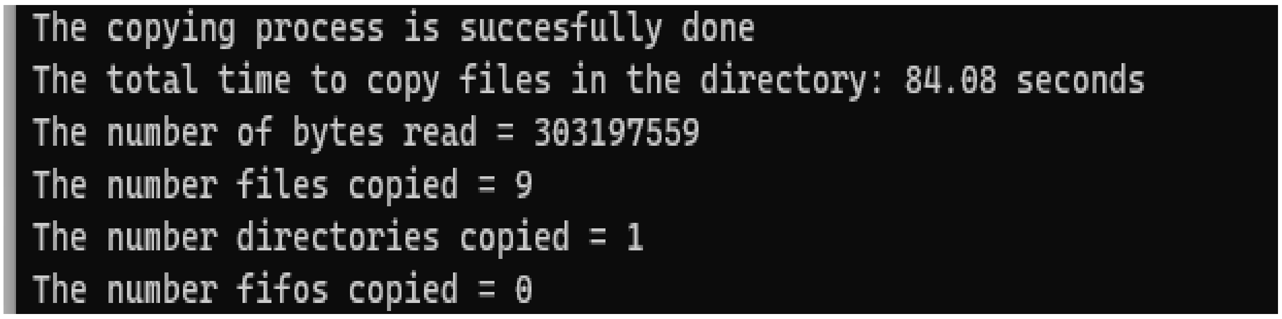
Ad	Değiştirme tarihi	Tür	Boyut
notes2	3.06.2023 21:46	Dosya klasörü	
Lecture3	30.11.2022 07:32	Microsoft Word B...	5.441 KB
Lecture4_Computer Arithmetic	30.11.2022 14:58	Zengin Metin Biçi...	71.205 KB
Lecture5_Multiply,Shift,Divide	29.11.2022 17:57	Microsoft Word B...	1.661 KB
Lecture6_Single_Cycle_Processor	30.11.2022 14:52	Microsoft Word B...	3.698 KB



```
meterose@DESKTOP-3HDDHUD:/mnt/c/hw5$ ./pCp 50 25 notes notes2
```

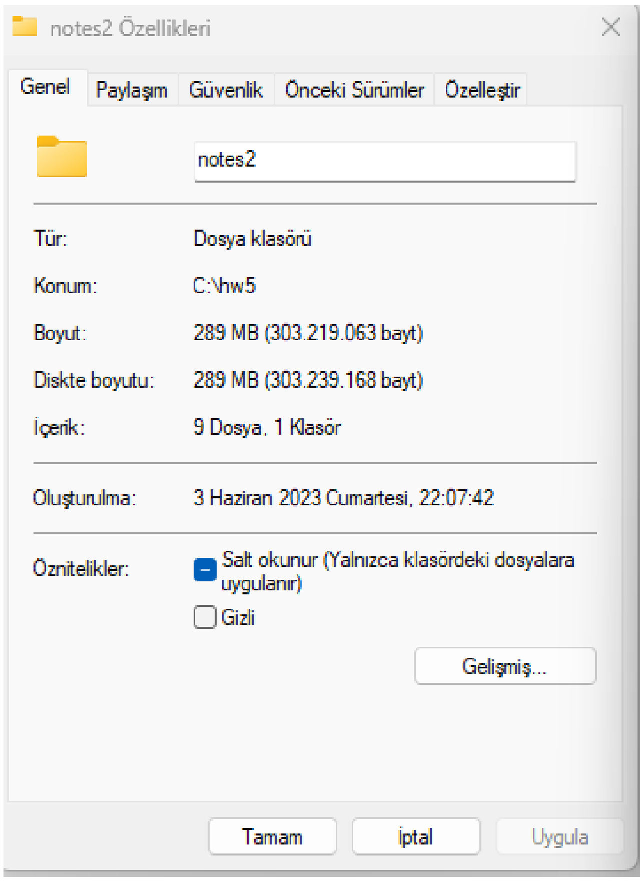
So I will give 50 for buffer, and create 25 threads for consumer, in the end lets check if the note and notes2 are the same.

So the output is as follows;



```
The copying process is succesfully done
The total time to copy files in the directory: 84.08 seconds
The number of bytes read = 303197559
The number files copied = 9
The number directories copied = 1
The number fifos copied = 0
```

Let’s check the size of the two files;



notes2 Özellikleri

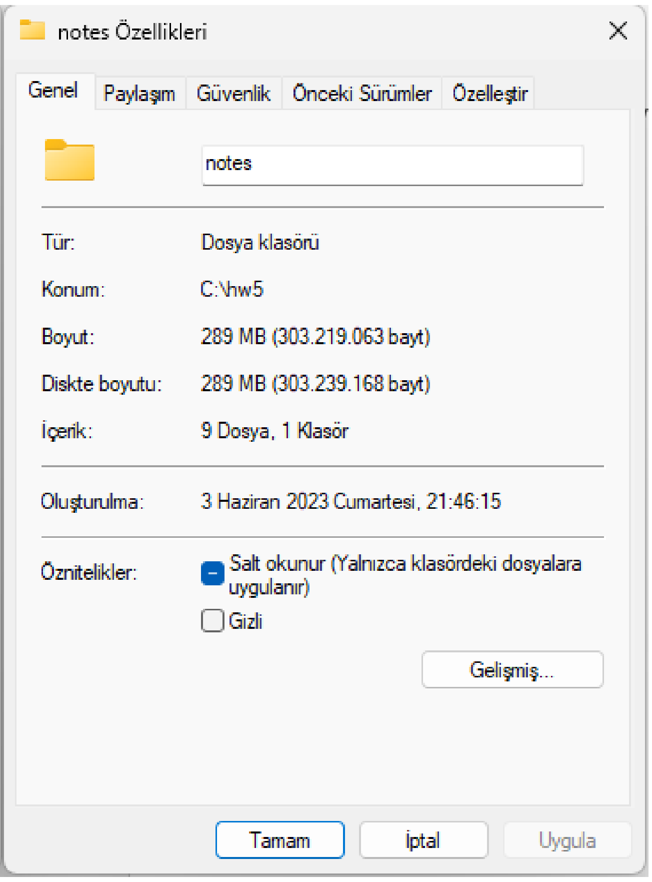
Genel | Paylaşım | Güvenlik | Önceki Sürümler | Özelleştir

notes2

Tür:	Dosya klasörü
Konum:	C:\hw5
Boyut:	289 MB (303.219.063 bayt)
Diskte boyutu:	289 MB (303.239.168 bayt)
İçerik:	9 Dosya, 1 Klasör
Oluşturulma:	3 Haziran 2023 Cumartesi, 22:07:42
Öznitelikler:	■ Salt okunur (Yalnızca klasördeki dosyalara uygulanır) ☐ Gizli

Gelişmiş...

Tamam | İptal | Uygula



notes Özellikleri

Genel | Paylaşım | Güvenlik | Önceki Sürümler | Özelleştir

notes

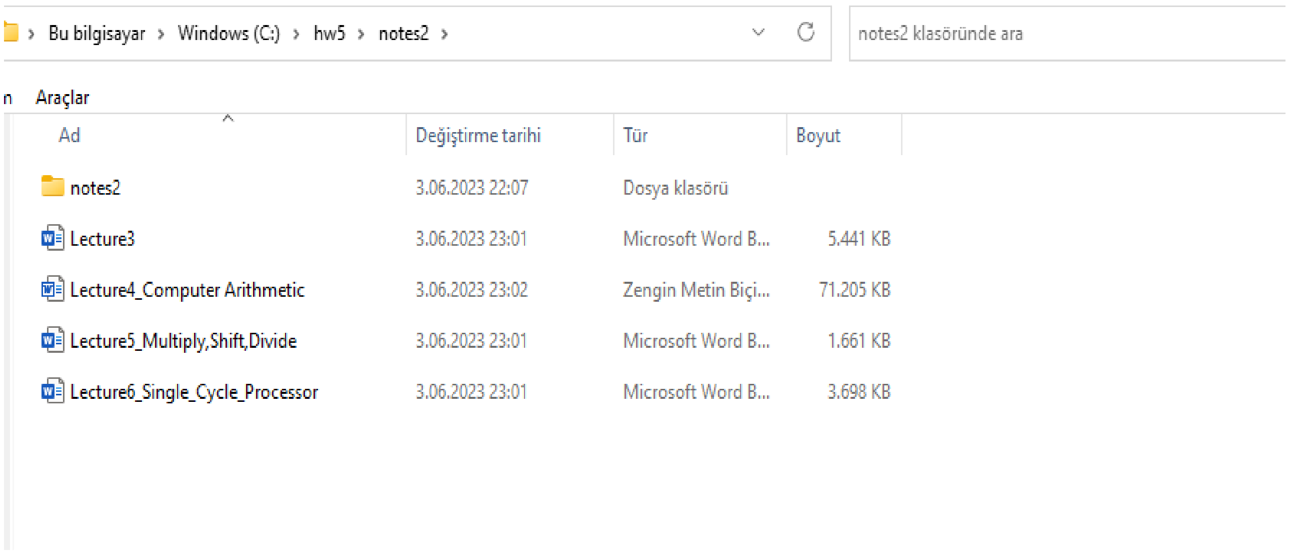
Tür:	Dosya klasörü
Konum:	C:\hw5
Boyut:	289 MB (303.219.063 bayt)
Diskte boyutu:	289 MB (303.239.168 bayt)
İçerik:	9 Dosya, 1 Klasör
Oluşturulma:	3 Haziran 2023 Cumartesi, 21:46:15
Öznitelikler:	■ Salt okunur (Yalnızca klasördeki dosyalara uygulanır) ☐ Gizli

Gelişmiş...

Tamam | İptal | Uygula

As you can see the sizes **are the same**

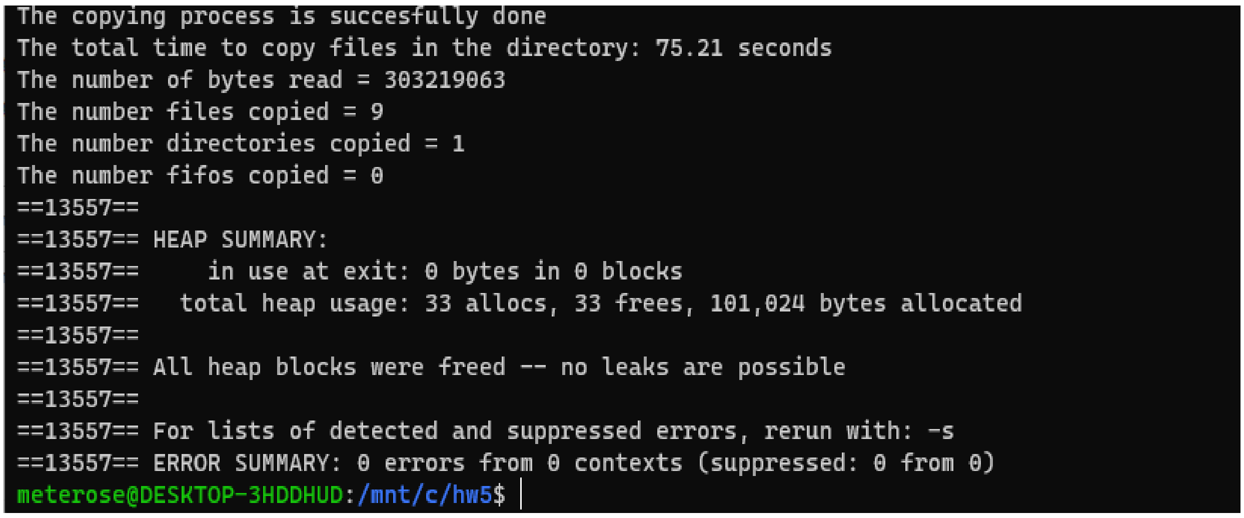
Let’s see also the content of the notes2



Ad	Değiştirme tarihi	Tür	Boyut
notes2	3.06.2023 22:07	Dosya klasörü	
Lecture3	3.06.2023 23:01	Microsoft Word B...	5.441 KB
Lecture4_Computer Arithmetic	3.06.2023 23:02	Zengin Metin Biçi...	71.205 KB
Lecture5_Multiply,Shift,Divide	3.06.2023 23:01	Microsoft Word B...	1.661 KB
Lecture6_Single_Cycle_Processor	3.06.2023 23:01	Microsoft Word B...	3.698 KB

As you can see the content of note and note2 are exactly the same, the subdirectories, content of files, and pdfs are all the same. So I created the exact copy of the same directory.

Let’s find the if there is a memory leak or not by running the command with Valgrind;

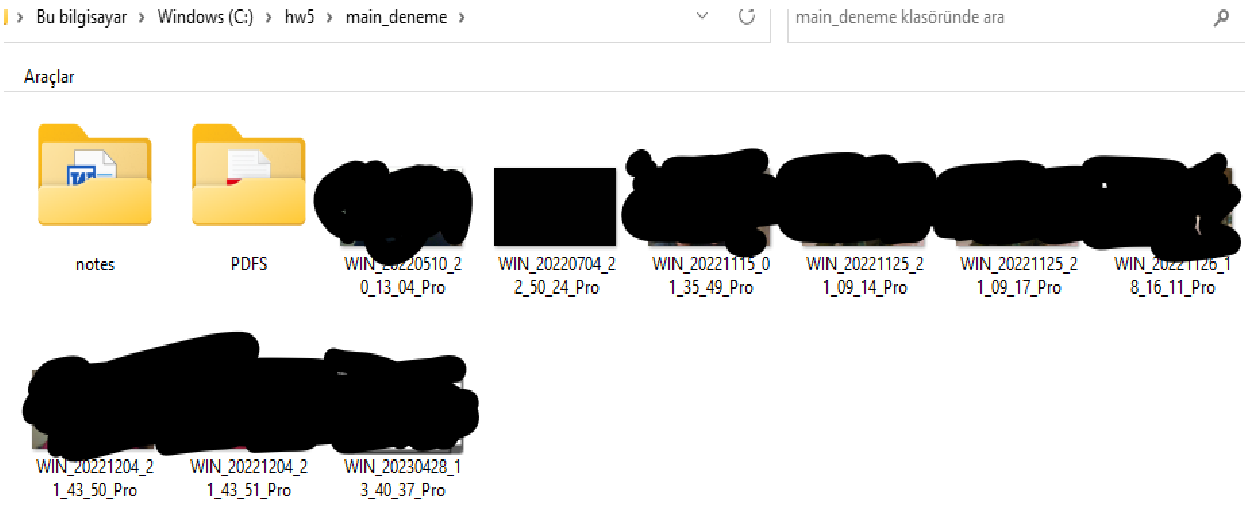


```
The copying process is succesfully done
The total time to copy files in the directory: 75.21 seconds
The number of bytes read = 303219063
The number files copied = 9
The number directories copied = 1
The number fifos copied = 0
==13557==
==13557== HEAP SUMMARY:
==13557==     in use at exit: 0 bytes in 0 blocks
==13557==   total heap usage: 33 allocs, 33 frees, 101,024 bytes allocated
==13557==
==13557== All heap blocks were freed -- no leaks are possible
==13557==
==13557== For lists of detected and suppressed errors, rerun with: -s
==13557== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
meterose@DESKTOP-3HDDHUD:/mnt/c/hw5$
```

As you can see **there is no memory leak**

Example 2

So I will try to copy another file which includes pdf, videos and photos.


Bu bilgisayar › Windows (C:) › hw5 › main_deneme ›

main_deneme klasöründe ara

Araçlar

notes

PDFS

[redacted] WIN_20220510_20_13_04_Pro

[redacted] WIN_20220704_22_50_24_Pro

[redacted] WIN_20221115_01_35_49_Pro

[redacted] WIN_20221125_21_09_14_Pro

[redacted] WIN_20221125_21_09_17_Pro

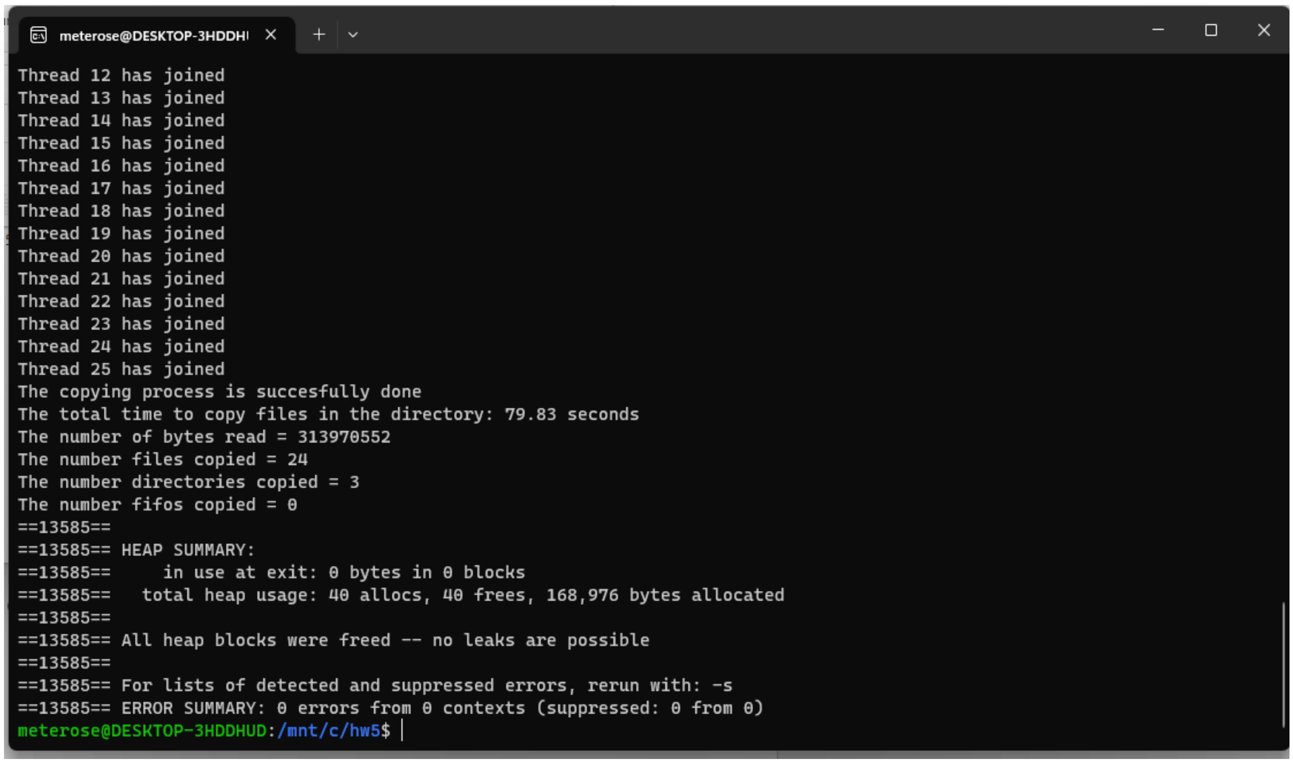
[redacted] WIN_20221126_18_16_11_Pro

[redacted] WIN_20221204_21_43_50_Pro

WIN_20221204_21_43_51_Pro

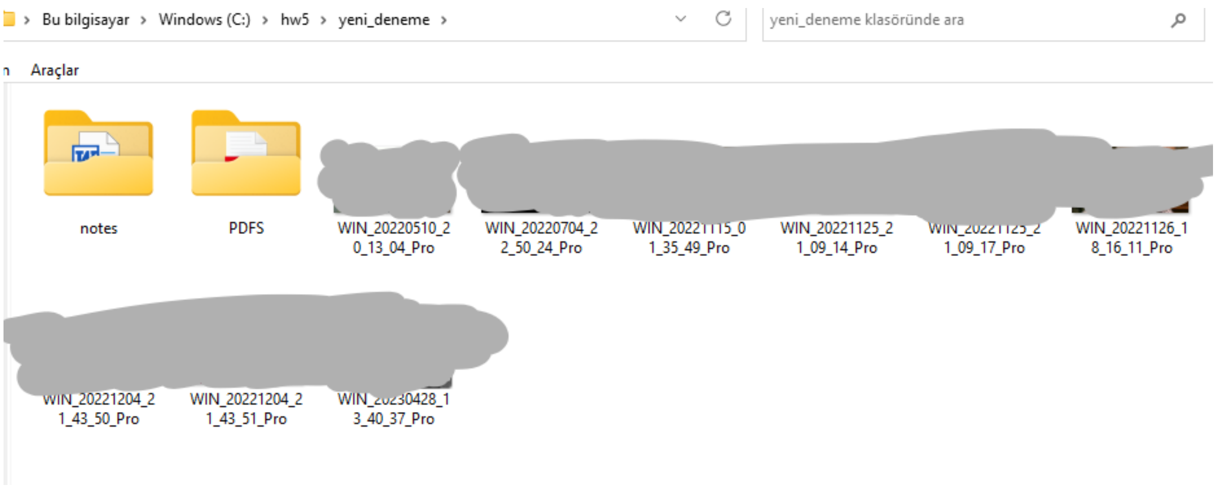
WIN_20230428_13_40_37_Pro

So lets try to copy that and after that lets compare the contents and sizes.


meterose@DESKTOP-3HDDHI

```
Thread 12 has joined
Thread 13 has joined
Thread 14 has joined
Thread 15 has joined
Thread 16 has joined
Thread 17 has joined
Thread 18 has joined
Thread 19 has joined
Thread 20 has joined
Thread 21 has joined
Thread 22 has joined
Thread 23 has joined
Thread 24 has joined
Thread 25 has joined
The copying process is succesfully done
The total time to copy files in the directory: 79.83 seconds
The number of bytes read = 313970552
The number files copied = 24
The number directories copied = 3
The number fifos copied = 0
==13585==
==13585== HEAP SUMMARY:
==13585==     in use at exit: 0 bytes in 0 blocks
==13585==   total heap usage: 40 allocs, 40 frees, 168,976 bytes allocated
==13585==
==13585== All heap blocks were freed -- no leaks are possible
==13585==
==13585== For lists of detected and suppressed errors, rerun with: -s
==13585== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
meterose@DESKTOP-3HDDHUD:/mnt/c/hw5$
```

It is the output of the copying command as you can see **above there is no memory** leak and I also got all threads so it also means that **no DEADLOCK happened**



Bu bilgisayar › Windows (C:) › hw5 › yeni_deneme ›

yeni_deneme klasöründe ara

Araçlar

notes

PDFS

[redacted]

WIN_20220510_20_13_04_Pro

WIN_20220704_22_50_24_Pro

WIN_20221115_01_35_49_Pro

WIN_20221125_21_09_14_Pro

WIN_20221125_21_09_17_Pro

WIN_20221126_18_16_11_Pro

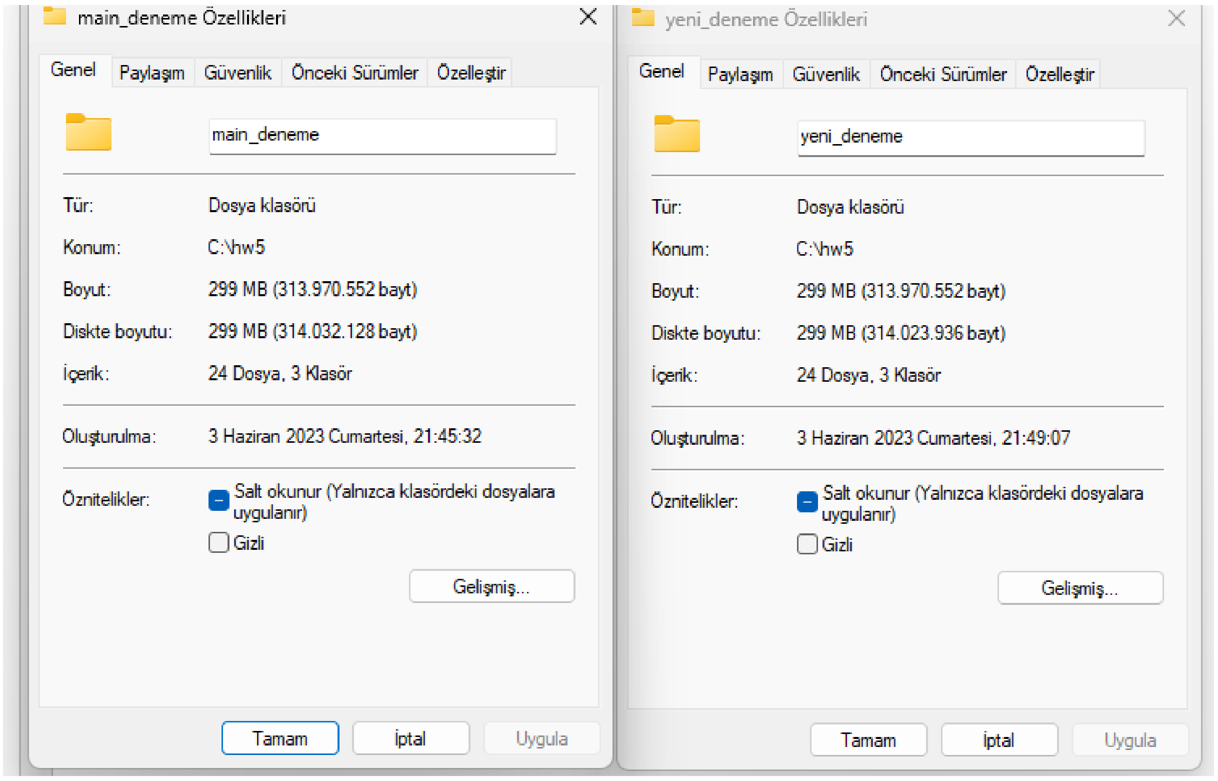
[redacted]

WIN_20221204_21_43_50_Pro

WIN_20221204_21_43_51_Pro

WIN_20230428_13_40_37_Pro

As you can see above the content of the copy file is exactly the same lets compare their sizes



main_deneme Özellikleri

Genel | Paylaşım | Güvenlik | Önceki Sürümler | Özelleştir

main_deneme

Tür:	Dosya klasörü
Konum:	C:\hw5
Boyut:	299 MB (313.970.552 bayt)
Diskte boyutu:	299 MB (314.032.128 bayt)
İçerik:	24 Dosya, 3 Klasör
Oluşturulma:	3 Haziran 2023 Cumartesi, 21:45:32
Öznitelikler:	Salt okunur (Yalnızca klasördeki dosyalara uygulanır)
	Gizli

Gelişmiş...

Tamam | İptal | Uygula

yeni_deneme Özellikleri

Genel | Paylaşım | Güvenlik | Önceki Sürümler | Özelleştir

yeni_deneme

Tür:	Dosya klasörü
Konum:	C:\hw5
Boyut:	299 MB (313.970.552 bayt)
Diskte boyutu:	299 MB (314.023.936 bayt)
İçerik:	24 Dosya, 3 Klasör
Oluşturulma:	3 Haziran 2023 Cumartesi, 21:49:07
Öznitelikler:	Salt okunur (Yalnızca klasördeki dosyalara uygulanır)
	Gizli

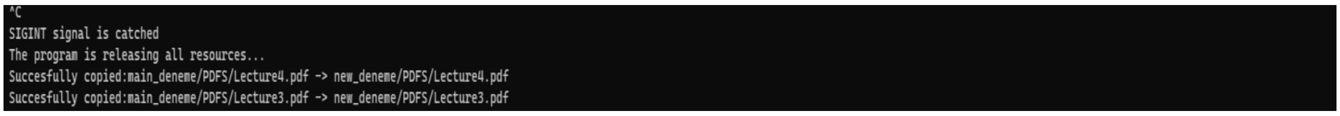
Gelişmiş...

Tamam | İptal | Uygula

The sizes are exactly the same just like the contents so I achieved to create copy file

Signal Test

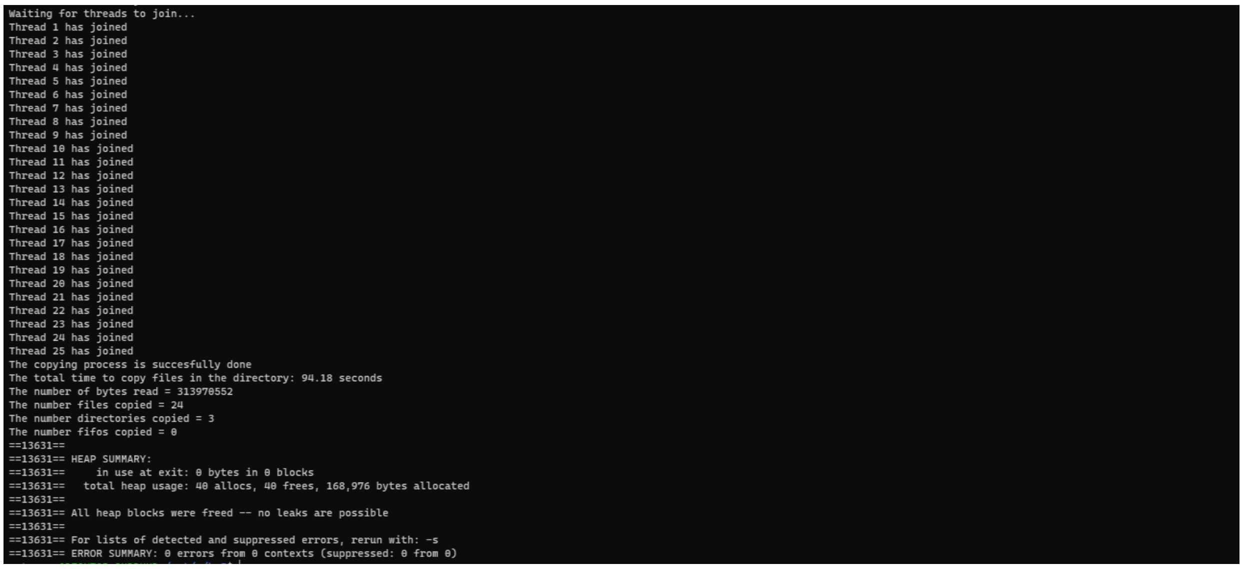
Lastly, I will try to press CTRL-C to trigger SIGINT signal during the copying process and check if the the process terminates or not.



```
^C
SIGINT signal is catched
The program is releasing all resources...
Succesfully copied:main_deneme/PDFS/Lecture4.pdf -> new_deneme/PDFS/Lecture4.pdf
Succesfully copied:main_deneme/PDFS/Lecture3.pdf -> new_deneme/PDFS/Lecture3.pdf
```

So when I press CTRL-C, the program is caught the signal immediately and makes done = 1 which terminates the producer thread and after that, it terminates all consumer threads. For the threads that are in the copying process, it takes time to terminate because they are not leaving that function until they copy the content of that particular file

So the output is;



```
Waiting for threads to join...
Thread 1 has joined
Thread 2 has joined
Thread 3 has joined
Thread 4 has joined
Thread 5 has joined
Thread 6 has joined
Thread 7 has joined
Thread 8 has joined
Thread 9 has joined
Thread 10 has joined
Thread 11 has joined
Thread 12 has joined
Thread 13 has joined
Thread 14 has joined
Thread 15 has joined
Thread 16 has joined
Thread 17 has joined
Thread 18 has joined
Thread 19 has joined
Thread 20 has joined
Thread 21 has joined
Thread 22 has joined
Thread 23 has joined
Thread 24 has joined
Thread 25 has joined
The copying process is succesfully done
The total time to copy files in the directory: 94.18 seconds
The number of bytes read = 313970552
The number files copied = 24
The number directories copied = 3
The number fifos copied = 0
==13631==
==13631== HEAP SUMMARY:
==13631==     in use at exit: 0 bytes in 0 blocks
==13631==   total heap usage: 40 allocs, 40 frees, 168,976 bytes allocated
==13631==
==13631== All heap blocks were freed -- no leaks are possible
==13631==
==13631== For lists of detected and suppressed errors, rerun with: -s
==13631== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

So as you can see right after the SIGINT is caught it terminates the whole threads and releases all the resources by not causing any memory leak or deadlock.

Mehmet Mete Şamlıoğlu

200104004093