

Gebze Technical University

Department Of Computer Engineering

CSE 344 Spring 2023

System Programming

Homework 5

Due Date: 03.06.2023

Abdurrahman Bulut  
1901042258

# Introduction

In this assignment, we are asked to implement a directory copying utility called pCp. This utility should be able to create a new thread to copy each file and subdirectory in order to perform the overall task in parallel. In order to regulate the number of active threads at any time, we are also asked to implement a worker thread pool. This will ensure that system resources are not exceeded when copying large directory trees.

The overall implementation of the utility should use a producer-consumer based synchronization mechanism. The producer thread will be responsible for reading the files in the source directory and creating a buffer of file descriptors and file names. The consumer threads will then read items from the buffer and copy the corresponding files to the destination directory. The main program will take the buffer size, number of consumers, source and destination directories as command-line arguments. The main program will then start the threads and use `pthread_join` to wait for them to complete. The total time to copy the files in the directory will then be displayed.

The utility should be able to copy both regular files and FIFOs. It should also be able to recursively copy subdirectories. The utility should keep statistics about the number and types of files copied, as well as the total number of bytes copied. We will experiment with different buffer sizes and different numbers of consumer threads to determine the best combination of parameters for performance. We will also investigate what happens when the per-process limit on the number of open file descriptors is exceeded.

The buffer size is the amount of memory that is allocated to store the file names and file descriptors of the files that are to be copied. The number of consumers is the number of threads that are used to copy the files from the source directory to the destination directory. A higher buffer size allows the producer thread to store more files in the buffer before the consumer threads start copying them. This can improve the performance of the program by reducing the number of times the producer thread needs to block while waiting for a consumer thread to become available. However, a buffer size that is too large can also have a negative impact on performance by increasing the amount of memory that is used. A higher number of consumers allows the producer thread to start copying files to the destination directory sooner. This can improve the performance of the program by reducing the amount of time that the producer thread needs to spend waiting for all of the files to be read from the source directory. However, a number of consumers that is too high can also have a negative impact on performance by increasing the amount of overhead that is associated with thread creation and management.

The ideal combination of buffer size and number of consumers will vary depending on the specific application. However, in general, it is important to find a balance between the two factors that will improve performance without sacrificing memory usage or increasing overhead.

In addition to the buffer size and the number of consumers, the number and sizes of files in the source directory can also affect the runtime of the program. A directory with a large number of small files will require more time to copy than a directory with a small number of large files. This is because the producer thread will need to spend more time reading the file names and file descriptors from the source directory.

The program also includes a check to ensure that the number of file descriptors per process does not exceed a certain limit. If this limit is exceeded, the process will be blocked until one of the open file descriptors is closed. This can happen if the source directory contains a large number of files or if the buffer size is too large. To prevent this situation, the program prints an error message and terminates in a controlled manner.

Here are some specific examples of how the buffer size and number of consumers can affect the runtime of the program:

- If the buffer size is too small, the producer thread will need to block more often, which will reduce the performance of the program.
- If the number of consumers is too small, the producer thread will need to wait for all of the files to be copied before it can start copying the next batch of files. It will also reduce the performance of the program.
- If the source directory contains a large number of small files, a larger buffer size will be required to improve the performance of the program.
- If the source directory contains a small number of large files, a smaller buffer size will be required to improve the performance of the program.

## Code explanation:

```
typedef struct
{
    int sourceFileDescriptor;
    int destinationFileDescriptor;
    char filename[1024];
} FileItem;

FileItem *buffer = NULL;

int bufferCount = 0, bufferSize = 0, numConsumerThreads = 0;
int num_of_directory_copied = 0, num_of_regular_copied = 0, num_of_fifo_copied = 0;
long long bytes_copied = 0;
bool done = false;

pthread_cond_t bufferEmpty = PTHREAD_COND_INITIALIZER, bufferFull = PTHREAD_COND_INITIALIZER;
pthread_mutex_t bufferMutex = PTHREAD_MUTEX_INITIALIZER, stdoutMutex = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t doneMutex = PTHREAD_MUTEX_INITIALIZER, numOfDirectoryCopiedMutex = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t numOfRegularCopiedMutex = PTHREAD_MUTEX_INITIALIZER, numOfFifoCopiedMutex = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t bytesCopiedMutex = PTHREAD_MUTEX_INITIALIZER;

static void copy_directory(char *readName, char *writeName, struct stat fileStat);
static void copy_file(char *readName, char *writeName, struct stat fileStat);
static void print_error(char *message, char *name);
static void add_to_buffer(char *readName, int rfd, int wfd);
static void *producer(void *arguments);
static void *consumer(void *p);
char *getSourceName(char *sourcePath);
void handleThreadError(int errorCode, char *msg);
void handleSigaction(int signo, siginfo_t *info, void *context);
```

The FileItem structure is used to store information about the files that are to be copied. The structure contains the file name, the source file descriptor, and the destination file descriptor.

The buffer variable is a pointer to an array of FileItem structures. The bufferCount variable keeps track of the number of files that are currently in the buffer.

The numConsumerThreads variable is the number of consumer threads that are used to copy the files.

The num\_of\_directory\_copied, num\_of\_regular\_copied, and num\_of\_fifo\_copied variables are used to keep track of the number of directories, regular files, and FIFOs that have been copied.

The bytes\_copied variable is used to keep track of the number of bytes that have been copied.

The done variable is a flag that is used to indicate when the copying process is complete.

The bufferEmpty and bufferFull condition variables are used to synchronize the producer and consumer threads.

The bufferMutex, stdoutMutex, doneMutex, numOfDirectoryCopiedMutex, numOfRegularCopiedMutex, numOfFifoCopiedMutex, and bytesCopiedMutex mutexes are used to protect shared data from concurrent access.

The copy\_directory(), copy\_file(), print\_error(), add\_to\_buffer(), producer(), consumer(), getSourceName(), handleThreadError(), and handleSigaction() functions are used to implement the different features of my program.

The first thing that the main() function does is to check the number of command-line arguments. If there are not five arguments, then the program prints an error message and exits.

The next thing that the main() function does is to get the number of consumer threads and the buffer size from the command-line arguments. If either of these values is negative, then the program prints an error message and exits. The program runs with the below command.

*./pCp 10 5 ./from ./to*

```
// Setup signal handling
struct sigaction sa;
memset(&sa, 0, sizeof(sa));
sa.sa_sigaction = handleSigaction;
sa.sa_flags = SA_SIGINFO;

if (sigaction(SIGINT, &sa, NULL) == -1)
{
    perror("sigaction");
    exit(EXIT_FAILURE);
}
```

The main() function then sets up signal handling. This is done by creating a signal action structure and setting the sa\_sigaction field to the address of the handleSigaction() function. The sa\_flags field is set to SA\_SIGINFO so that the handleSigaction() function can receive information about the signal that was received.

```
size_t stacksize;
pthread_attr_t attr;

pthread_attr_init(&attr);
pthread_attr_getstacksize(&attr, &stacksize);
pthread_attr_setstacksize(&attr, 5 * stacksize);

buffer = (FileItem *)malloc(bufferSize * sizeof(FileItem));

char **arguments = (char **)malloc(2 * sizeof(char *));

for (int j = 0; j < 2; ++j)
    arguments[j] = (char *)malloc(512 * sizeof(char));

strcpy(arguments[0], argv[3]);
strcpy(arguments[1], argv[4]);
```

The first lines of code allocate a thread attribute object and set the stack size to five times the default stack size. This is done to ensure that the threads have enough stack space to operate. The next line of the code allocates memory for the buffer. The buffer is used to store information about the files that are to be copied. Then it allocates memory for the arguments array. The arguments array is used to pass the source and destination directories to the producer and consumer threads.

```

// Check if destination directory exists and create it if it doesn't.
struct stat st = {0};
if (stat(arguments[1], &st) == -1) {
    printf("Destination folder %s does not exist. It is created..\n", arguments[1]);
    mkdir(arguments[1], 0700);
}

strcat(arguments[1], "/");
char *temp = getSourceName(arguments[0]);
strcat(arguments[1], temp);
free(temp);

// File stats and folder creation
lstat(arguments[0], &fileStat) == -1 ? mkdir(arguments[1], 0666) : mkdir(arguments[1], fileStat.st_mode);

// Start timer
gettimeofday(&startTime, NULL);

// Thread creation and joining
errorCode = pthread_create(&producerThread, &attr, producer, (void *)arguments);

if (errorCode)
    handleThreadError(errorCode, "pthread_create error for Producer!");

consumerThreads = (pthread_t *)malloc(numConsumerThreads * sizeof(pthread_t));

for (int i = 0; i < numConsumerThreads; ++i)
{
    errorCode = pthread_create(&consumerThreads[i], NULL, consumer, NULL);
    if (errorCode)
        handleThreadError(errorCode, "pthread_create error for Consumer!");
}

```

Above code first checks if the destination directory exists. If it does not exist, the directory is created. The destination directory is then created. if the source directory is a directory. If it is not, the directory is created. And then I start a timer. After that the code creates the producer and consumer threads. The producer thread is responsible for reading the files from the source directory and adding them to the buffer. The consumer threads are responsible for reading the files from the buffer and copying them to the destination directory.

```

pthread_attr_destroy(&attr);

errorCode = pthread_join(producerThread, NULL);

if (errorCode)
    handleThreadError(errorCode, "pthread_join error for Producer!");

pthread_mutex_lock(&doneMutex);
done = true;
pthread_mutex_unlock(&doneMutex);

pthread_cond_broadcast(&bufferFull);

for (int i = 0; i < numConsumerThreads; ++i)
{
    errorCode = pthread_join(consumerThreads[i], &bytesCopied);
    if (errorCode)
        handleThreadError(errorCode, "pthread_join error for Consumer!");

    pthread_mutex_lock(&bytesCopiedMutex);
    bytes_copied += *(long long *)bytesCopied;
    pthread_mutex_unlock(&bytesCopiedMutex);

    free(bytesCopied);
}

// End timer
gettimeofday(&endTime, NULL);

```

It waits for the producer thread to finish. Once the producer thread has finished, the done flag is set to true. This indicates to the consumer threads that they should stop copying files. Then it joins the consumer.

```

// Print results
printf("\n");
printf("-----\n");
printf("| Copying finished!           \n");
printf("-----\n");
printf("| Directories copied:  %d          \n", num_of_directory_copied);
printf("| Regular files copied: %d          \n", num_of_regular_copied);
printf("| FIFO files copied:   %d          \n", num_of_fifo_copied);
printf("| Total bytes copied:  %lld  \n", bytes_copied);
printf("| Time taken:         %.0lf μs      \n", (double)(endTime.tv_usec - startTime.tv_usec) + (double)(endTime.tv_sec - startTime.tv_sec) * 1000000);
printf("-----\n");
printf("\n");

// Cleanup
pthread_mutex_destroy(&doneMutex);
pthread_mutex_destroy(&numOfDirectoryCopiedMutex);
pthread_mutex_destroy(&bufferMutex);
pthread_mutex_destroy(&stdoutMutex);
pthread_mutex_destroy(&numOfRegularCopiedMutex);
pthread_mutex_destroy(&numOfFifoCopiedMutex);
pthread_mutex_destroy(&bytesCopiedMutex);

for (int j = 0; j < 2; ++j)
    free(arguments[j]);

free(arguments);
free(consumerThreads);
free(buffer);

return 0;

```



It prints a message to standard output that indicates that the copying process has finished. The message includes the number of directories, regular files, FIFOs, and total bytes that were copied. Then it cleans up the resources that were allocated by the program. The `pthread_mutex_destroy()` function is used to destroy the mutexes that were created. The `free()` function is used to free the memory that was allocated for the arguments array, the consumer threads, and the buffer.

## Helper functions

The ***copy\_directory()*** function is responsible for copying a directory from the source directory to the destination directory. The function first increments the number of directories that have been copied. The function then allocates memory for the arguments array. The function then populates the arguments array with the source and destination directories. The function then checks if the destination directory already exists. If it does, the function prints a message and replaces it. The function then calls the producer thread to copy the directory. The function finally frees the memory that was allocated for the arguments array.

The ***copy\_file()*** function is responsible for copying a file from the source directory to the destination directory. The function first checks if the file is a regular file. If it is, the function opens the file for reading. The function then checks if the file could be opened for reading. If it could not, the function prints an error message and returns. The function then opens the file for writing. The function then checks if the file could be opened for writing. If it could not, the function prints an error message and closes the file that was opened for reading. The function then checks if the number of open file descriptors is greater than 1000. If it is, the function prints an error message and sets the done flag. The function then adds the file to the buffer.

The ***print\_error()*** function is used to print error messages to standard output. The function first locks the `stdoutMutex` mutex to ensure that only one thread can print to standard output at a time. The function then prints the error message to standard output. The function finally unlocks the `stdoutMutex` mutex.

The ***add\_to\_buffer()*** function is used to add a file to the buffer. The function first creates a `FileItem` structure to store the file name, source file descriptor, and destination file descriptor. The function then locks the `bufferMutex` mutex to ensure that only one thread can modify the buffer at a time. The function then checks if the buffer is full. If the buffer is full, the function waits for the `bufferEmpty` condition variable to be signaled. Once the buffer is not full, the function adds the `FileItem` structure to the buffer. The function then increments the buffer count and signals the `bufferFull` condition variable to let other threads know that the buffer is not full. The function finally unlocks the `bufferMutex` mutex.

The ***producer()*** function is responsible for copying the files from the source directory to the destination directory. The function first checks if the destination directory exists. If it does not exist, the function creates it. The function then opens the source directory. The function then iterates over the files in the source directory. For each file, the function constructs the file names for the source and destination files. The function then checks if the file is a directory. If it is, the function copies the directory. Otherwise, the function copies the regular file. The function finally closes the directory and returns NULL to indicate that the thread has finished executing.

The ***consumer()*** function is responsible for copying the files from the buffer to the destination directory. The function first locks the bufferMutex mutex to ensure that only one thread can access the buffer at a time. The function then waits for the buffer to become non-empty. If the buffer is empty and done is set, then the copying process is finished. If the buffer is not empty, the function removes an item from the buffer. The function then unlocks the bufferMutex mutex and processes the item. The function then increments the total number of bytes copied and prints a message indicating that the file has been copied successfully. The function then increments the number of regular files or FIFOs copied, depending on the file type.

The ***getSourceName()*** function uses the basename() function to extract the base name from a path string. The basename() function returns a pointer to the base name, which is the last component of the path string, everything after the last '/' character. The getSourceName() function then allocates memory for a string to store the base name and copies the base name from the basename() result to the allocated string. The function finally returns the allocated string.

For example:

```
char *sourcePath = "/home/user/source/file.txt";
char *fileName = getSourceName(sourcePath);
printf("The file name is: %s\n", fileName);
```

This code will print the following output:

```
The file name is: file.txt
```

The ***handleThreadError()*** function is responsible for handling errors that occur in threads. The function first sets the errno variable to the error code. The function then prints an error message to standard error. The function finally exits the program with the status EXIT\_FAILURE.

The *handleSigaction()* function is responsible for handling the SIGINT signal, which is the signal that is sent when the user presses Ctrl+C. The function first checks if the signal is SIGINT. If it is, the function locks the doneMutex mutex. The function then sets the done flag to true. The function unlocks the doneMutex mutex and prints a message indicating that the program is exiting.

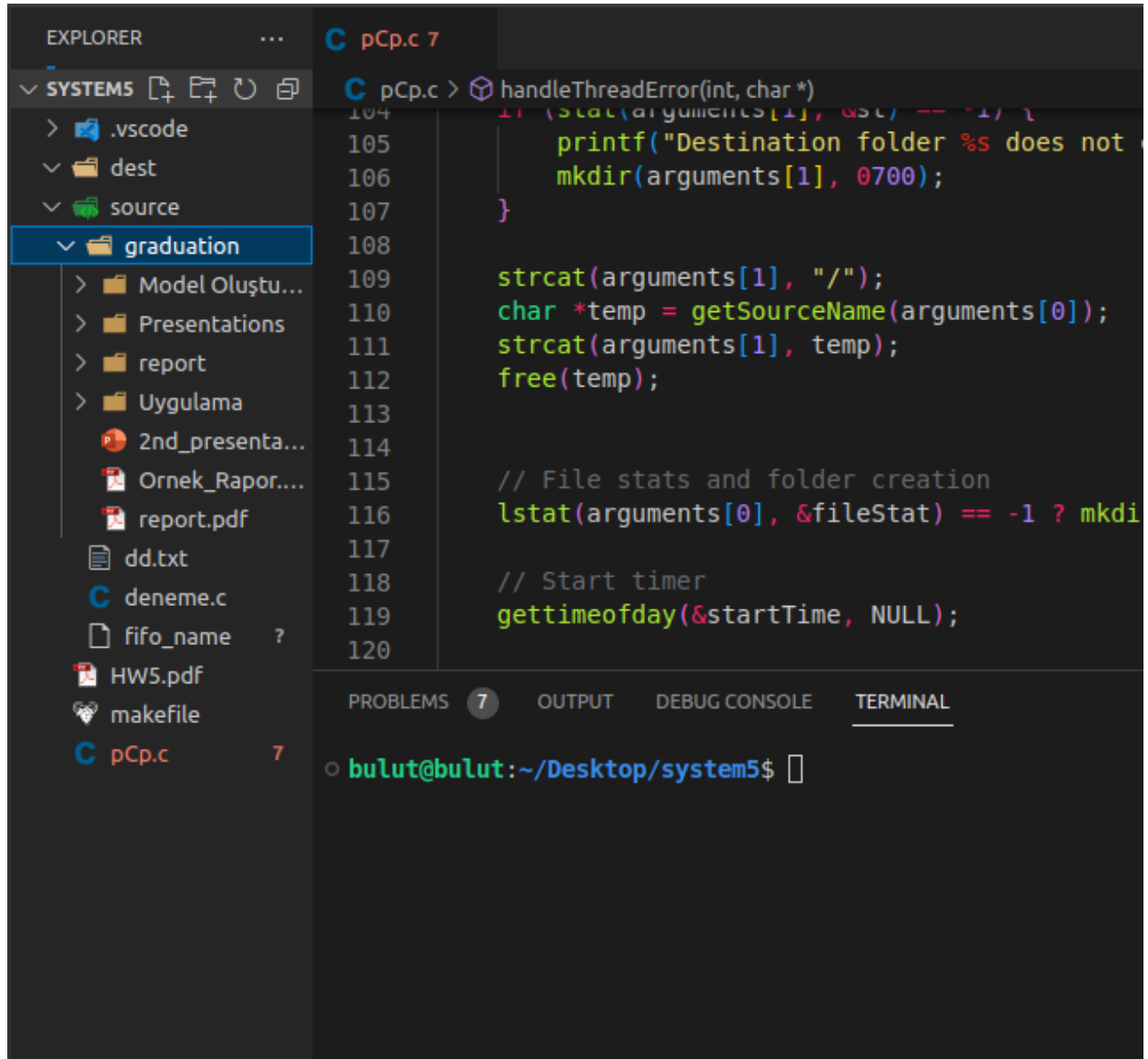
## Conclusion

I used a producer-consumer pattern to copy files from one directory to another. The producer thread iterates over the files in the source directory and adds them to a buffer. The consumer threads then remove files from the buffer and copy them to the destination directory.

I used a producer-consumer pattern to parallelize the copying process. This is a good way to speed up the copying process, especially on systems with multiple cores. I used a mutex to protect the buffer from concurrent access by the producer and consumer threads. This prevents data corruption. I used a condition variable to signal the consumer threads when there are new files in the buffer. This prevents the consumer threads from wasting time waiting for new files to be added to the buffer.

## Tests

Initial structure: My graduation folder is about 411GB. Besides that I have some other files such as deneme.c, dd.txt and fifo\_name. fifo\_name is fifo file that I created via terminal.



The image shows a Visual Studio Code editor window. On the left, the Explorer sidebar displays the file structure of a project named 'SYSTEM5'. The 'graduation' folder is selected, showing its contents: 'Model Oluştur...', 'Presentations', 'report', 'Uygulama', '2nd\_presenta...', 'Ornek\_Rapor...', 'report.pdf', 'dd.txt', 'deneme.c', 'fifo\_name', 'HW5.pdf', 'makefile', and 'pCp.c'. The main editor area shows the code for 'pCp.c', which includes a function 'handleThreadError' and various system calls like 'mkdir', 'strcat', 'getSourceName', 'lstat', and 'gettimeofday'. The bottom panel shows the 'TERMINAL' tab with the prompt 'bulut@bulut:~/Desktop/system5\$'.

```
EXPLORER
SYSTEM5
  .vscode
  dest
  source
  graduation
    Model Oluştur...
    Presentations
    report
    Uygulama
    2nd_presenta...
    Ornek_Rapor...
    report.pdf
    dd.txt
    deneme.c
    fifo_name
    HW5.pdf
    makefile
    pCp.c

pCp.c
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120

handleThreadError(int, char *)
{
    if (lstat(arguments[1], &st) == -1) {
        printf("Destination folder %s does not exist\n", arguments[1]);
        mkdir(arguments[1], 0700);
    }

    strcat(arguments[1], "/");
    char *temp = getSourceName(arguments[0]);
    strcat(arguments[1], temp);
    free(temp);

    // File stats and folder creation
    lstat(arguments[0], &fileStat) == -1 ? mkdir(arguments[1], 0700) : 0;

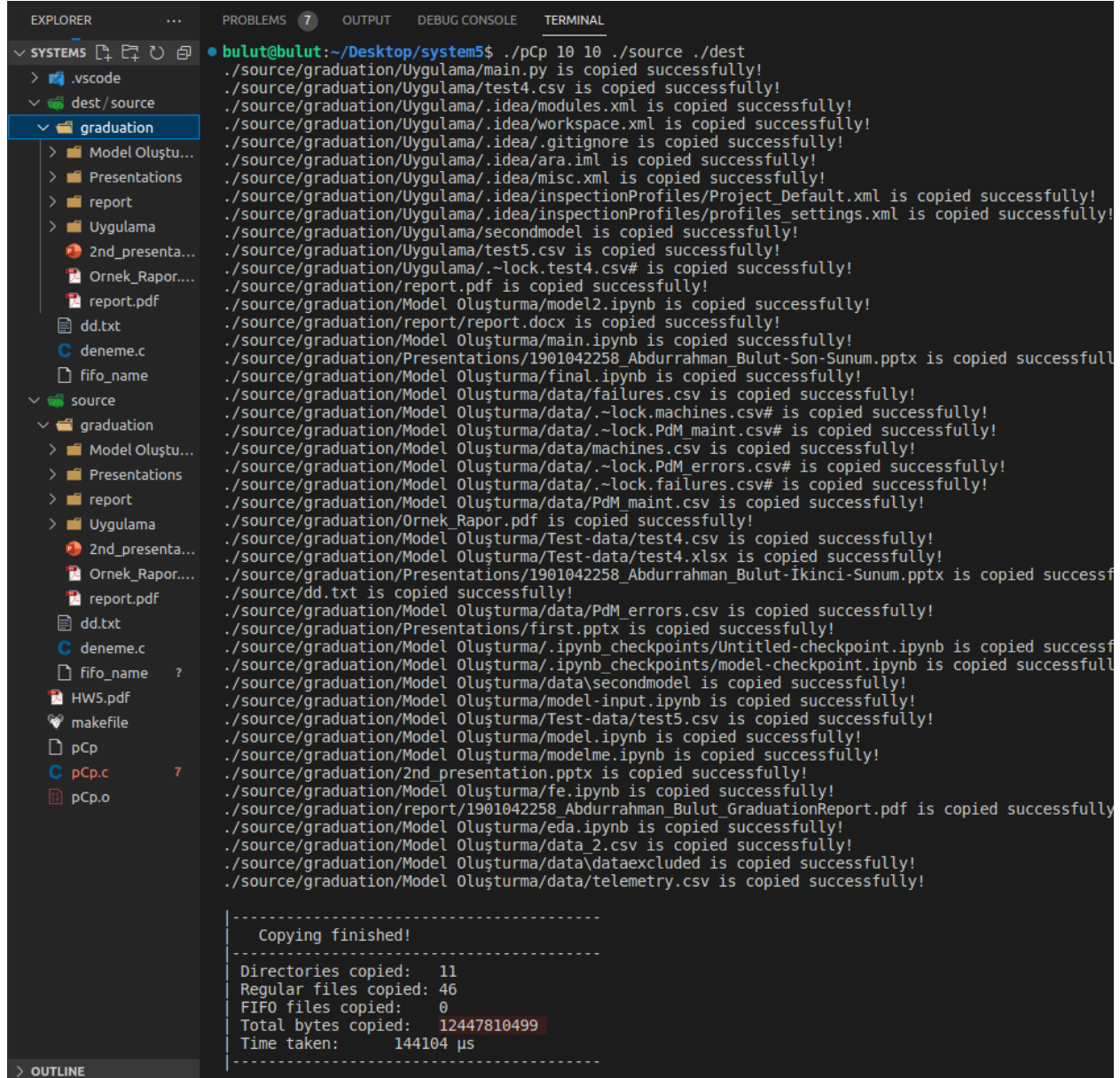
    // Start timer
    gettimeofday(&startTime, NULL);
}
```

```
bulut@bulut:~/Desktop/system5$
```

If source and dest folders are available or exist then the source folder copied to the inside of the dest folder successfully.

Command:

```
./pCp 10 10 ./source ./dest
```



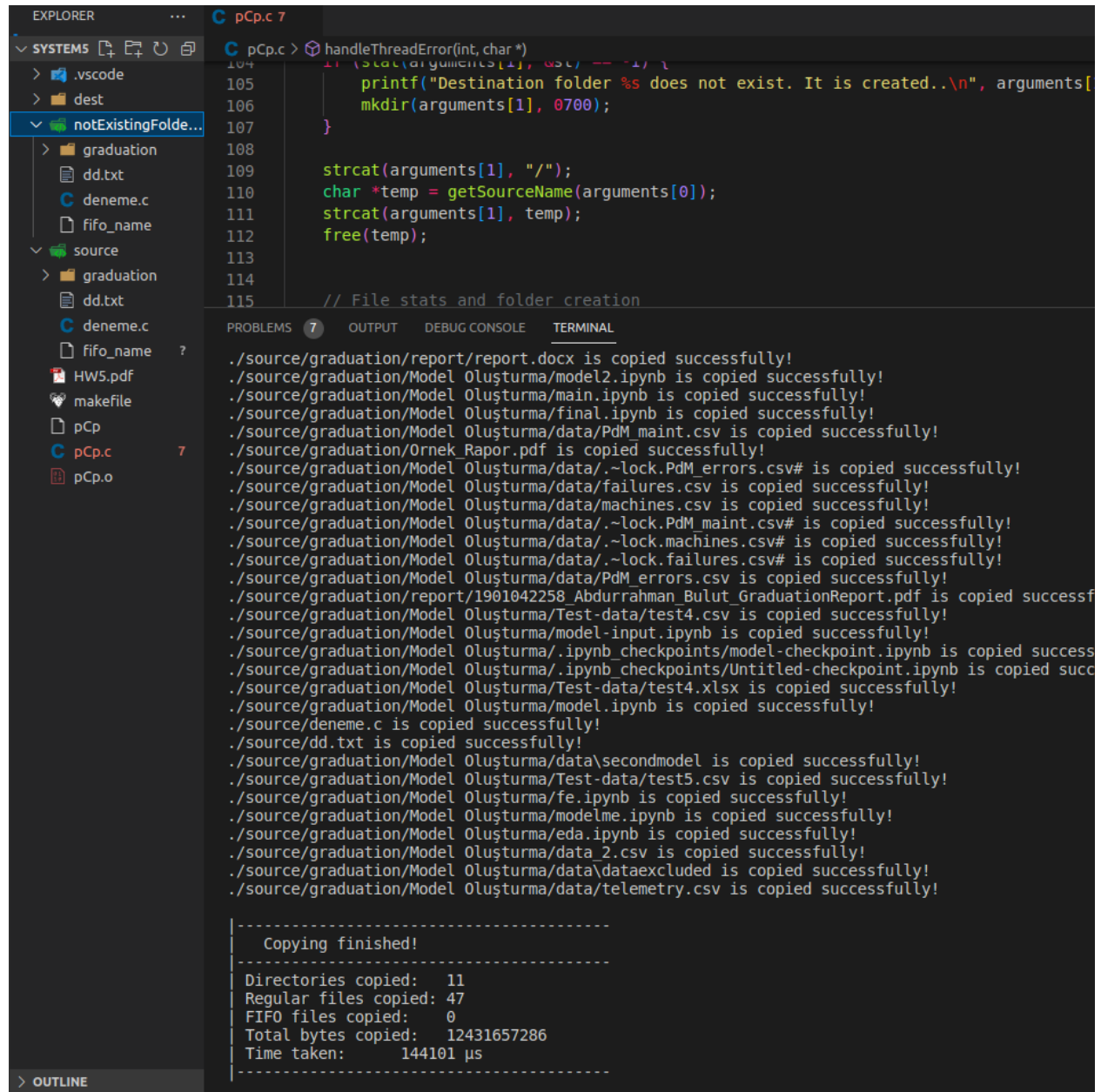
```
bulut@bulut:~/Desktop/system5$ ./pCp 10 10 ./source ./dest
./source/graduation/Uygulama/main.py is copied successfully!
./source/graduation/Uygulama/test4.csv is copied successfully!
./source/graduation/Uygulama/.idea/modules.xml is copied successfully!
./source/graduation/Uygulama/.idea/workspace.xml is copied successfully!
./source/graduation/Uygulama/.idea/.gitignore is copied successfully!
./source/graduation/Uygulama/.idea/ara.iml is copied successfully!
./source/graduation/Uygulama/.idea/misc.xml is copied successfully!
./source/graduation/Uygulama/.idea/inspectionProfiles/Project_Default.xml is copied successfully!
./source/graduation/Uygulama/.idea/inspectionProfiles/profiles_settings.xml is copied successfully!
./source/graduation/Uygulama/secondmodel is copied successfully!
./source/graduation/Uygulama/test5.csv is copied successfully!
./source/graduation/Uygulama/.~lock.test4.csv# is copied successfully!
./source/graduation/report.pdf is copied successfully!
./source/graduation/Model Olusturma/model2.ipynb is copied successfully!
./source/graduation/report/report.docx is copied successfully!
./source/graduation/Model Olusturma/main.ipynb is copied successfully!
./source/graduation/Presentations/1901042258_Abdurrahman Bulut-Son-Sunum.pptx is copied successfully!
./source/graduation/Model Olusturma/final.ipynb is copied successfully!
./source/graduation/Model Olusturma/data/failures.csv is copied successfully!
./source/graduation/Model Olusturma/data/.~lock.machines.csv# is copied successfully!
./source/graduation/Model Olusturma/data/.~lock.PdM_maint.csv# is copied successfully!
./source/graduation/Model Olusturma/data/machines.csv is copied successfully!
./source/graduation/Model Olusturma/data/.~lock.PdM_errors.csv# is copied successfully!
./source/graduation/Model Olusturma/data/.~lock.failures.csv# is copied successfully!
./source/graduation/Model Olusturma/data/PdM_maint.csv is copied successfully!
./source/graduation/Ornek_Rapor.pdf is copied successfully!
./source/graduation/Model Olusturma/Test-data/test4.csv is copied successfully!
./source/graduation/Model Olusturma/Test-data/test4.xlsx is copied successfully!
./source/graduation/Presentations/1901042258_Abdurrahman Bulut-Ikinci-Sunum.pptx is copied successfully!
./source/dd.txt is copied successfully!
./source/graduation/Model Olusturma/data/PdM_errors.csv is copied successfully!
./source/graduation/Presentations/first.pptx is copied successfully!
./source/graduation/Model Olusturma/.ipynb_checkpoints/Untitled-checkpoint.ipynb is copied successfully!
./source/graduation/Model Olusturma/.ipynb_checkpoints/model-checkpoint.ipynb is copied successfully!
./source/graduation/Model Olusturma/data/secondmodel is copied successfully!
./source/graduation/Model Olusturma/model-input.ipynb is copied successfully!
./source/graduation/Model Olusturma/Test-data/test5.csv is copied successfully!
./source/graduation/Model Olusturma/model.ipynb is copied successfully!
./source/graduation/Model Olusturma/modelme.ipynb is copied successfully!
./source/graduation/2nd_presentation.pptx is copied successfully!
./source/graduation/Model Olusturma/fe.ipynb is copied successfully!
./source/graduation/report/1901042258_Abdurrahman Bulut GraduationReport.pdf is copied successfully!
./source/graduation/Model Olusturma/eda.ipynb is copied successfully!
./source/graduation/Model Olusturma/data2.csv is copied successfully!
./source/graduation/Model Olusturma/data\dataexcluded is copied successfully!
./source/graduation/Model Olusturma/data/telemetry.csv is copied successfully!

-----
Copying finished!
-----
Directories copied: 11
Regular files copied: 46
FIFO files copied: 0
Total bytes copied: 12447810499
Time taken: 144104 µs
-----
```

If destination folder is not exists, it will be created and source folder will be copied into newly created destination folder.

Command:

```
./pCp 10 10 ./source ./notExistingFolder
```



The screenshot shows a Visual Studio Code editor with a C program named `pCp.c` and its execution output in the terminal. The Explorer panel on the left shows a file tree with a folder named `notExistingFolder` selected. The C program in the editor is a recursive copy utility that handles file and directory copying, including error handling for non-existent destination folders. The terminal output shows the successful execution of the command `./pCp 10 10 ./source ./notExistingFolder`, listing all copied files and directories, and providing a summary of the copying process.

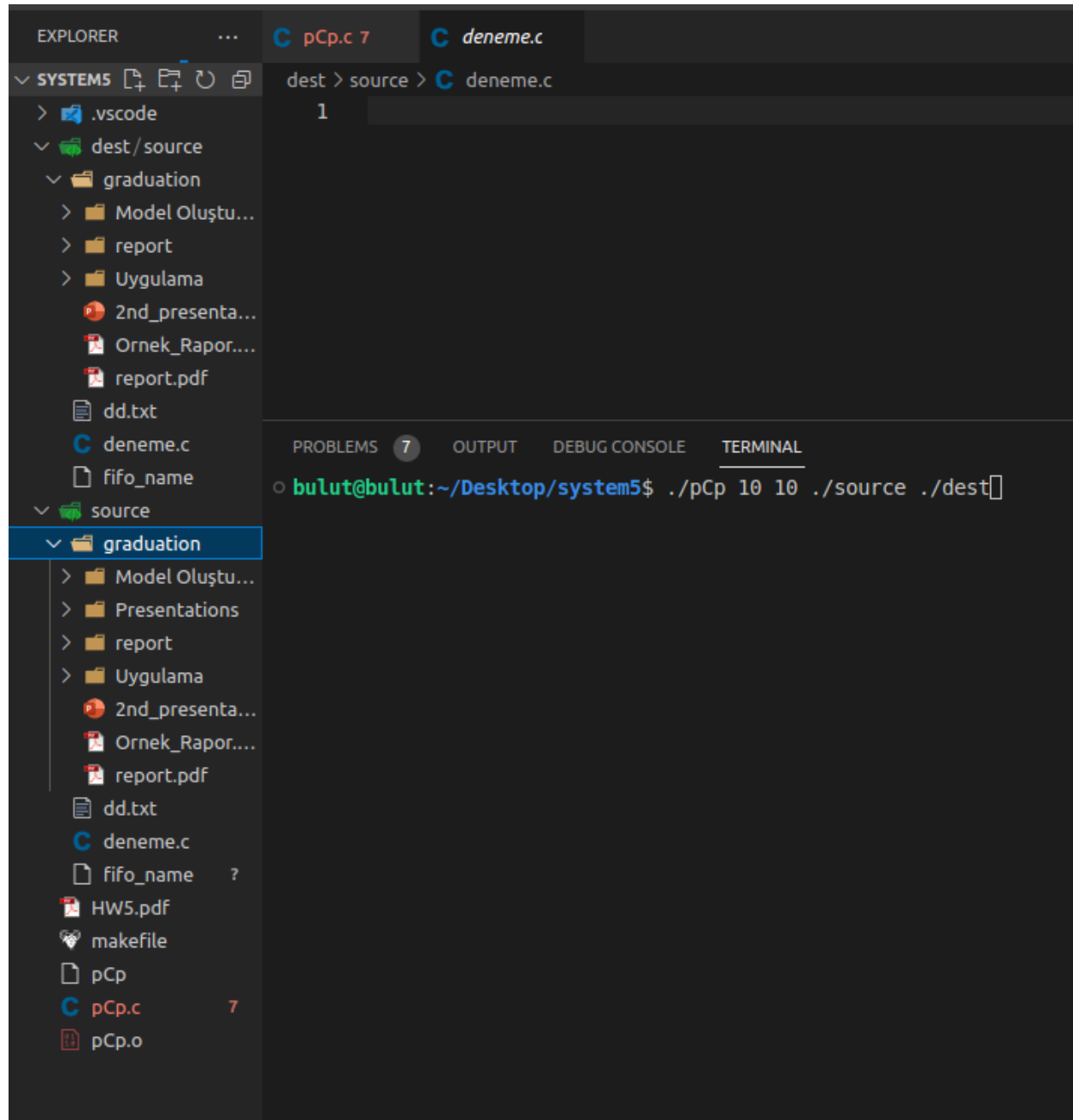
```
104 handleThreadError(int, char *)
105 if (strcmp(arguments[1], "cst") == 0) {
106     printf("Destination folder %s does not exist. It is created.\n", arguments[1]);
107     mkdir(arguments[1], 0700);
108 }
109
110 strcat(arguments[1], "/");
111 char *temp = getSourceName(arguments[0]);
112 strcat(arguments[1], temp);
113 free(temp);
114
115 // File stats and folder creation
```

```
./source/graduation/report/report.docx is copied successfully!
./source/graduation/Model Olusturma/model2.ipynb is copied successfully!
./source/graduation/Model Olusturma/main.ipynb is copied successfully!
./source/graduation/Model Olusturma/final.ipynb is copied successfully!
./source/graduation/Model Olusturma/data/PdM_maint.csv is copied successfully!
./source/graduation/Ornek Rapor.pdf is copied successfully!
./source/graduation/Model Olusturma/data/~lock.PdM_errors.csv# is copied successfully!
./source/graduation/Model Olusturma/data/failures.csv is copied successfully!
./source/graduation/Model Olusturma/data/machines.csv is copied successfully!
./source/graduation/Model Olusturma/data/~lock.PdM_maint.csv# is copied successfully!
./source/graduation/Model Olusturma/data/~lock.machines.csv# is copied successfully!
./source/graduation/Model Olusturma/data/~lock.failures.csv# is copied successfully!
./source/graduation/Model Olusturma/data/PdM_errors.csv is copied successfully!
./source/graduation/report/1901042258_Abdurrahman_Bulut_GraduationReport.pdf is copied successfully!
./source/graduation/Model Olusturma/Test-data/test4.csv is copied successfully!
./source/graduation/Model Olusturma/model-input.ipynb is copied successfully!
./source/graduation/Model Olusturma/.ipynb_checkpoints/model-checkpoint.ipynb is copied successfully!
./source/graduation/Model Olusturma/.ipynb_checkpoints/Untitled-checkpoint.ipynb is copied successfully!
./source/graduation/Model Olusturma/Test-data/test4.xlsx is copied successfully!
./source/graduation/Model Olusturma/model.ipynb is copied successfully!
./source/deneme.c is copied successfully!
./source/dd.txt is copied successfully!
./source/graduation/Model Olusturma/data\secondmodel is copied successfully!
./source/graduation/Model Olusturma/Test-data/test5.csv is copied successfully!
./source/graduation/Model Olusturma/fe.ipynb is copied successfully!
./source/graduation/Model Olusturma/modelme.ipynb is copied successfully!
./source/graduation/Model Olusturma/eda.ipynb is copied successfully!
./source/graduation/Model Olusturma/data 2.csv is copied successfully!
./source/graduation/Model Olusturma/data\dataexcluded is copied successfully!
./source/graduation/Model Olusturma/data/telemetry.csv is copied successfully!

-----
Copying finished!
-----
Directories copied: 11
Regular files copied: 47
FIFO files copied: 0
Total bytes copied: 12431657286
Time taken: 144101 µs
-----
```

If destination folder already has the source folder contents, it will be overwritten. As an example I deleted the presentation folder in my graduation project folder. Initial view is:

Command: `./pCp 10 10 ./source ./dest`



View after copying:

```
EXPLORER  ...  pCp.c  deneme.c
└─ SYSTEMS
  └─ dest/source
    └─ graduation
      └─ Model Oluştur...
      └─ Presentations
      └─ report
      └─ Uygulama
      └─ 2nd_presenta...
      └─ Ornek_Rapor...
      └─ report.pdf
      └─ dd.txt
      └─ deneme.c
      └─ fifo_name
    └─ source
      └─ graduation
        └─ Model Oluştur...
        └─ Presentations
        └─ report
        └─ Uygulama
        └─ 2nd_presenta...
        └─ Ornek_Rapor...
        └─ report.pdf
        └─ dd.txt
        └─ deneme.c
        └─ fifo_name ?
      └─ HWS.pdf
      └─ makefile
      └─ pCp
      └─ pCp.c
      └─ pCp.o

dest > source > deneme.c
1

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
Directory ./dest/source/graduation/Model Oluşturma/data exist. It will be replaced.
./source/graduation/Model Oluşturma/data/PdM_maint.csv is copied successfully!
./source/graduation/Model Oluşturma/data/machines.csv is copied successfully!
./source/graduation/Model Oluşturma/data/.~lock.machines.csv# is copied successfully!
./source/graduation/Model Oluşturma/final.ipynb is copied successfully!
./source/graduation/Model Oluşturma/data/.~lock.PdM_errors.csv# is copied successfully!
./source/graduation/Model Oluşturma/data/failures.csv is copied successfully!
./source/graduation/Model Oluşturma/data/.~lock.PdM_maint.csv# is copied successfully!
./source/graduation/Model Oluşturma/data/.~lock.failures.csv# is copied successfully!

Directory ./dest/source/graduation/Model Oluşturma/Test-data exist. It will be replaced.
./source/graduation/Model Oluşturma/Test-data/test4.csv is copied successfully!
./source/graduation/Model Oluşturma/data/PdM_errors.csv is copied successfully!
./source/graduation/Model Oluşturma/Test-data/test4.xlsx is copied successfully!
./source/graduation/Model Oluşturma/Test-data/test5.csv is copied successfully!
./source/graduation/Model Oluşturma/model.ipynb is copied successfully!
./source/graduation/Model Oluşturma/data/secondmodel is copied successfully!
./source/graduation/Model Oluşturma/modelme.ipynb is copied successfully!

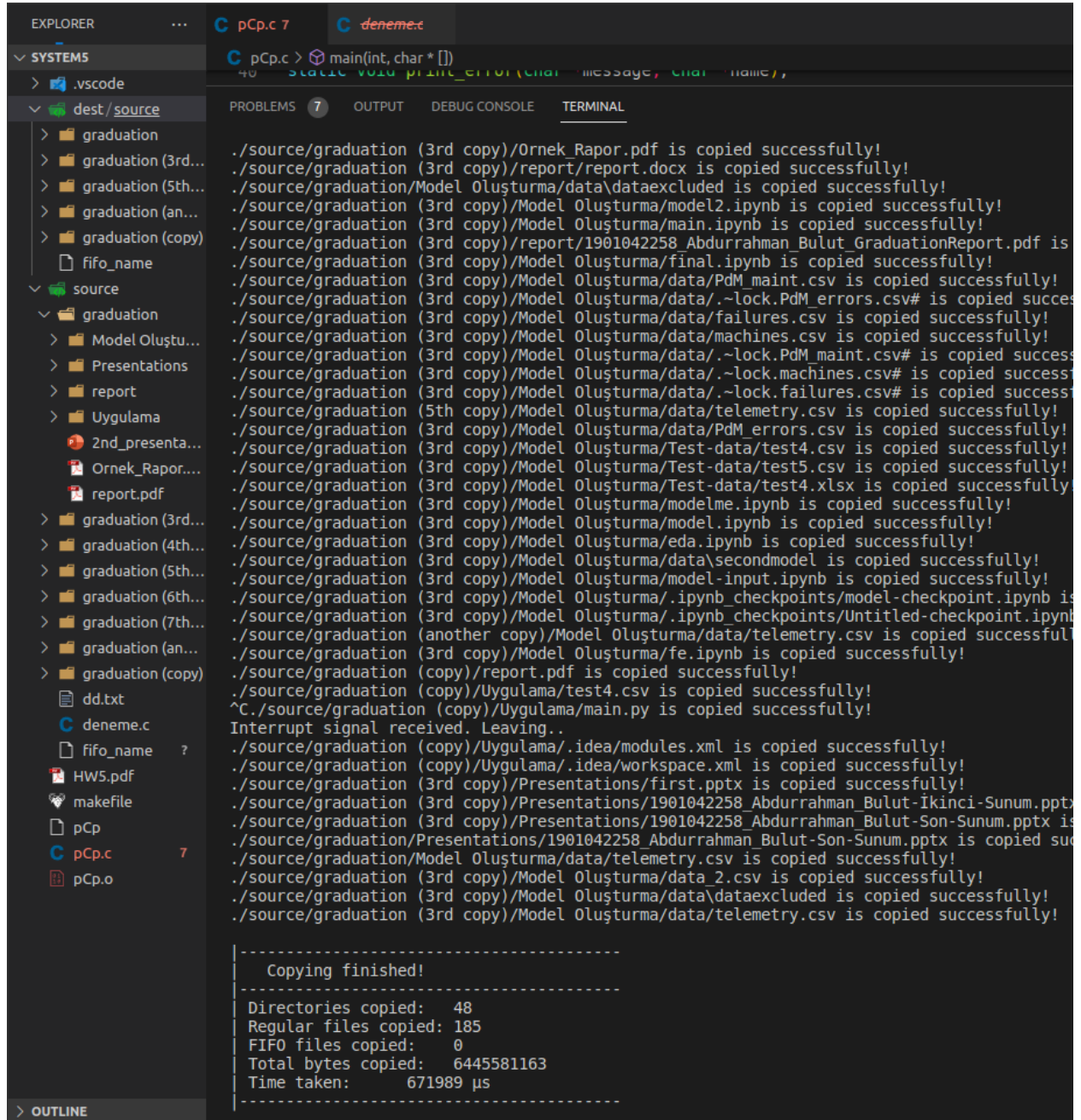
Directory ./dest/source/graduation/Model Oluşturma/.ipynb_checkpoints exist. It will be replaced.
./source/graduation/Model Oluşturma/model-input.ipynb is copied successfully!
./source/graduation/Model Oluşturma/.ipynb_checkpoints/model-checkpoint.ipynb is copied successfully!
./source/graduation/Model Oluşturma/.ipynb_checkpoints/Untitled-checkpoint.ipynb is copied successfully!
./source/deneme.c is copied successfully!
./source/dd.txt is copied successfully!
./source/graduation/Model Oluşturma/eda.ipynb is copied successfully!
./source/graduation/Model Oluşturma/fe.ipynb is copied successfully!
./source/graduation/Model Oluşturma/data_2.csv is copied successfully!
./source/graduation/Model Oluşturma/data\dataexcluded is copied successfully!
./source/graduation/Model Oluşturma/data/telemetry.csv is copied successfully!

-----
| Copying finished!
|-----
| Directories copied: 11
| Regular files copied: 47
| FIFO files copied: 0
| Total bytes copied: 12325067320
| Time taken: 260408 µs
|-----
```



When I press CTRL+C while copying, the signal handler will work and stop the process. Copied folders until signal received stay in dest folder.

Command: `./pCp 5 5 ./source ./dest`



```
EXPLORER
...
C pCp.c 7
C deneme.c

SYSTEMS
> .vscode
> dest/source
  > graduation
  > graduation (3rd copy)
  > graduation (5th copy)
  > graduation (another copy)
  > graduation (copy)
  > fifo_name
  > source
    > graduation
    > Model Olusturma
    > Presentations
    > report
    > Uygulama
    > 2nd_presentation
    > Ornek_Rapor.pdf
    > report.pdf
    > graduation (3rd copy)
    > graduation (4th copy)
    > graduation (5th copy)
    > graduation (6th copy)
    > graduation (7th copy)
    > graduation (another copy)
    > graduation (copy)
    > dd.txt
    > deneme.c
    > fifo_name
    > HW5.pdf
    > makefile
    > pCp
    > pCp.c
    > pCp.o

PROBLEMS 7 OUTPUT DEBUG CONSOLE TERMINAL

./source/graduation (3rd copy)/Ornek_Rapor.pdf is copied successfully!
./source/graduation (3rd copy)/report/report.docx is copied successfully!
./source/graduation/Model Olusturma/data/dataexcluded is copied successfully!
./source/graduation (3rd copy)/Model Olusturma/model2.ipynb is copied successfully!
./source/graduation (3rd copy)/Model Olusturma/main.ipynb is copied successfully!
./source/graduation (3rd copy)/report/1901042258_Abdurrahman Bulut GraduationReport.pdf is copied successfully!
./source/graduation (3rd copy)/Model Olusturma/final.ipynb is copied successfully!
./source/graduation (3rd copy)/Model Olusturma/data/PdM_maint.csv is copied successfully!
./source/graduation (3rd copy)/Model Olusturma/data/.~lock.PdM_errors.csv# is copied successfully!
./source/graduation (3rd copy)/Model Olusturma/data/failures.csv is copied successfully!
./source/graduation (3rd copy)/Model Olusturma/data/machines.csv is copied successfully!
./source/graduation (3rd copy)/Model Olusturma/data/.~lock.PdM_maint.csv# is copied successfully!
./source/graduation (3rd copy)/Model Olusturma/data/.~lock.machines.csv# is copied successfully!
./source/graduation (3rd copy)/Model Olusturma/data/.~lock.failures.csv# is copied successfully!
./source/graduation (5th copy)/Model Olusturma/data/telemetry.csv is copied successfully!
./source/graduation (3rd copy)/Model Olusturma/data/PdM_errors.csv is copied successfully!
./source/graduation (3rd copy)/Model Olusturma/Test-data/test4.csv is copied successfully!
./source/graduation (3rd copy)/Model Olusturma/Test-data/test5.csv is copied successfully!
./source/graduation (3rd copy)/Model Olusturma/Test-data/test4.xlsx is copied successfully!
./source/graduation (3rd copy)/Model Olusturma/modelme.ipynb is copied successfully!
./source/graduation (3rd copy)/Model Olusturma/model.ipynb is copied successfully!
./source/graduation (3rd copy)/Model Olusturma/eda.ipynb is copied successfully!
./source/graduation (3rd copy)/Model Olusturma/data/secondmodel is copied successfully!
./source/graduation (3rd copy)/Model Olusturma/model-input.ipynb is copied successfully!
./source/graduation (3rd copy)/Model Olusturma/.ipynb_checkpoints/model-checkpoint.ipynb is copied successfully!
./source/graduation (3rd copy)/Model Olusturma/.ipynb_checkpoints/Untitled-checkpoint.ipynb is copied successfully!
./source/graduation (another copy)/Model Olusturma/data/telemetry.csv is copied successfully!
./source/graduation (3rd copy)/Model Olusturma/fe.ipynb is copied successfully!
./source/graduation (copy)/report.pdf is copied successfully!
./source/graduation (copy)/Uygulama/test4.csv is copied successfully!
^C./source/graduation (copy)/Uygulama/main.py is copied successfully!
Interrupt signal received. Leaving..
./source/graduation (copy)/Uygulama/.idea/modules.xml is copied successfully!
./source/graduation (copy)/Uygulama/.idea/workspace.xml is copied successfully!
./source/graduation (3rd copy)/Presentations/first.pptx is copied successfully!
./source/graduation (3rd copy)/Presentations/1901042258_Abdurrahman Bulut-Ikinci-Sunum.pptx is copied successfully!
./source/graduation (3rd copy)/Presentations/1901042258_Abdurrahman Bulut-Son-Sunum.pptx is copied successfully!
./source/graduation/Presentations/1901042258_Abdurrahman Bulut-Son-Sunum.pptx is copied successfully!
./source/graduation/Model Olusturma/data/telemetry.csv is copied successfully!
./source/graduation (3rd copy)/Model Olusturma/data 2.csv is copied successfully!
./source/graduation (3rd copy)/Model Olusturma/data\dataexcluded is copied successfully!
./source/graduation (3rd copy)/Model Olusturma/data/telemetry.csv is copied successfully!

-----
Copying finished!
-----
Directories copied: 48
Regular files copied: 185
FIFO files copied: 0
Total bytes copied: 6445581163
Time taken: 671989 µs
-----
```

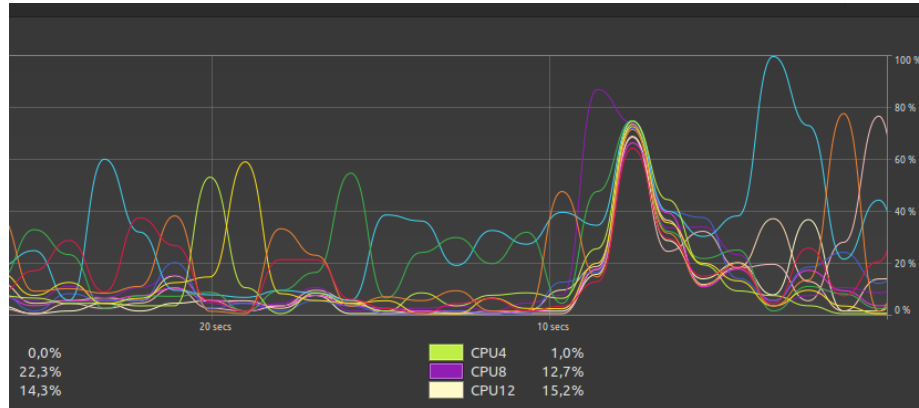
## ***Memory leak test***

Command: `valgrind ./pCp 5 5 ./source ./dest`

```
|-----  
==7831==  
==7831== HEAP SUMMARY:  
==7831==    in use at exit: 0 bytes in 0 blocks  
==7831== total heap usage: 920 allocs, 920 frees, 1,417,421,082 bytes allocated  
==7831==  
==7831== All heap blocks were freed -- no leaks are possible  
==7831==
```

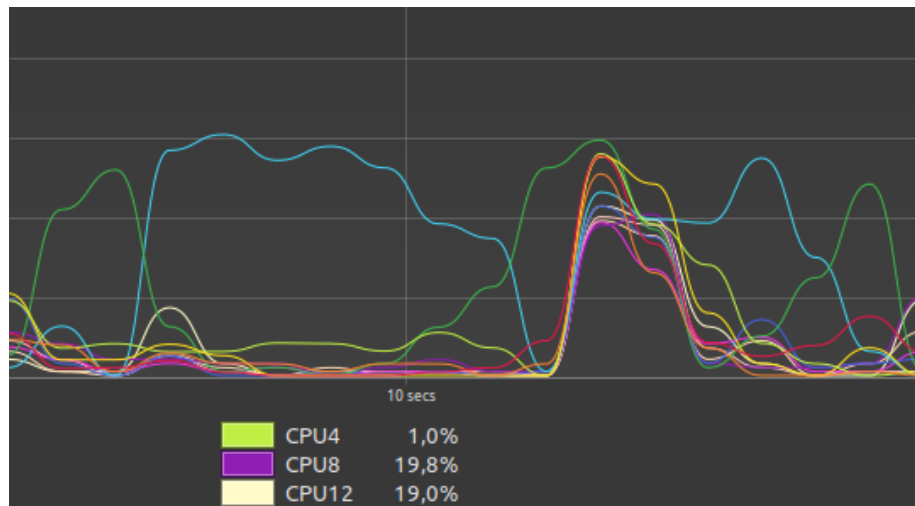
As we can see there is no memory leak.

## Experiment for different buffer sizes and different consumer threads



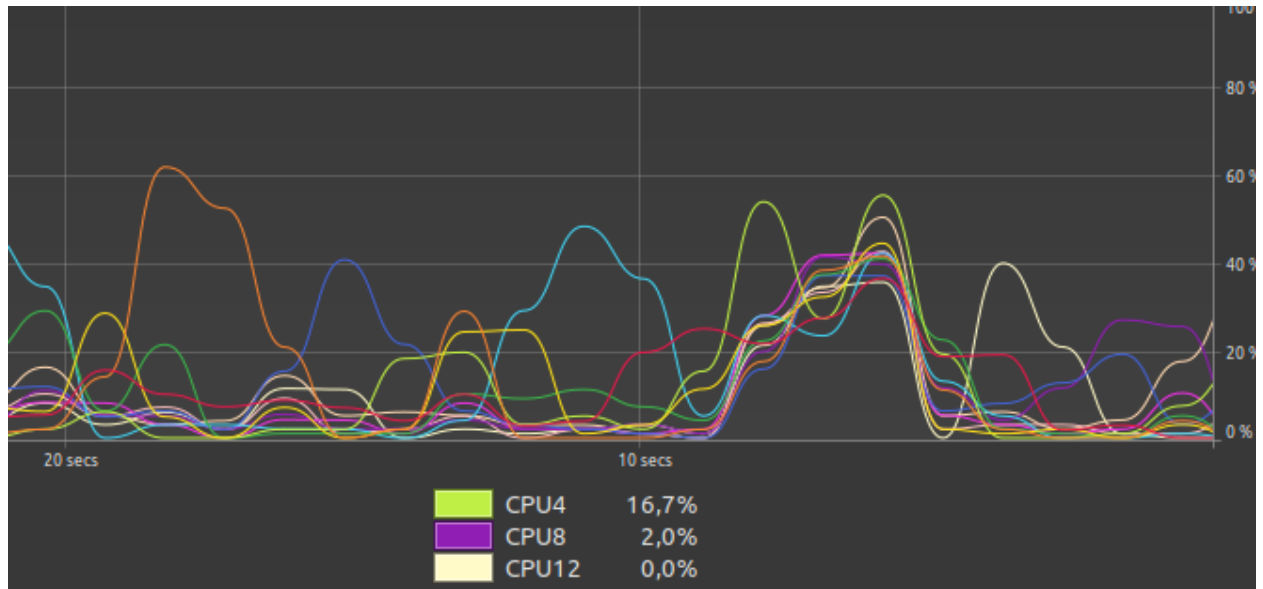
```
-----  
Copying finished!  
-----  
Directories copied: 110  
Regular files copied: 451  
FIFO files copied: 0  
Total bytes copied: 1291944449  
Time taken: 2359107 µs  
-----  
bulut@bulut: ~/Desktop/system5$
```

Consumer = 10, buffer size = 10



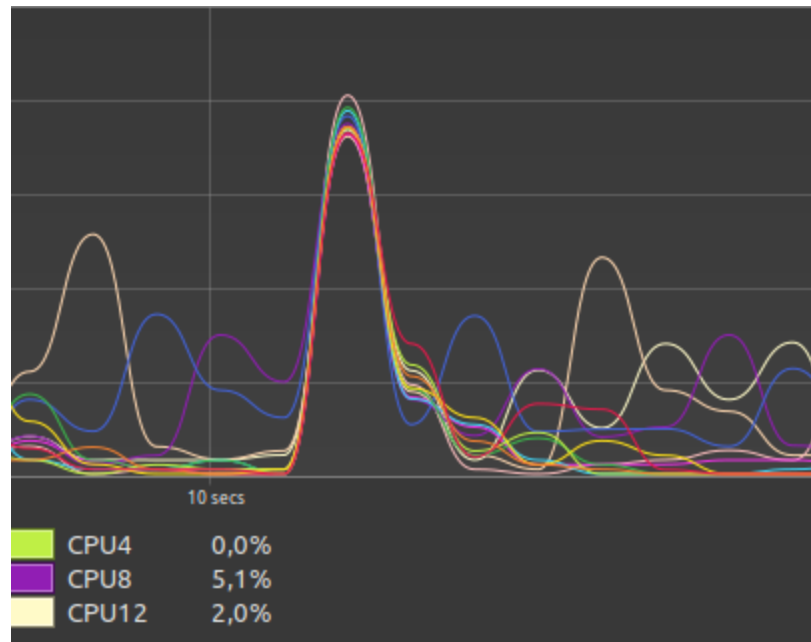
```
-----  
Copying finished!  
-----  
Directories copied: 110  
Regular files copied: 451  
FIFO files copied: 0  
Total bytes copied: 6502801397  
Time taken: 1678479 µs  
-----  
bulut@bulut:~/Desktop/system5$
```

Consumer = 5, buffer size = 10



```
./source/graduation (7th copy)/Model Oluřturma/data,  
./source/graduation (6th copy)/Model Oluřturma/data,  
  
-----  
Copying finished!  
-----  
Directories copied: 110  
Regular files copied: 451  
FIFO files copied: 0  
Total bytes copied: 12918530641  
Time taken: 2085337 μs  
-----  
bulut@bulut:~/Desktop/system5$
```

Consumer = 10, buffer size = 5



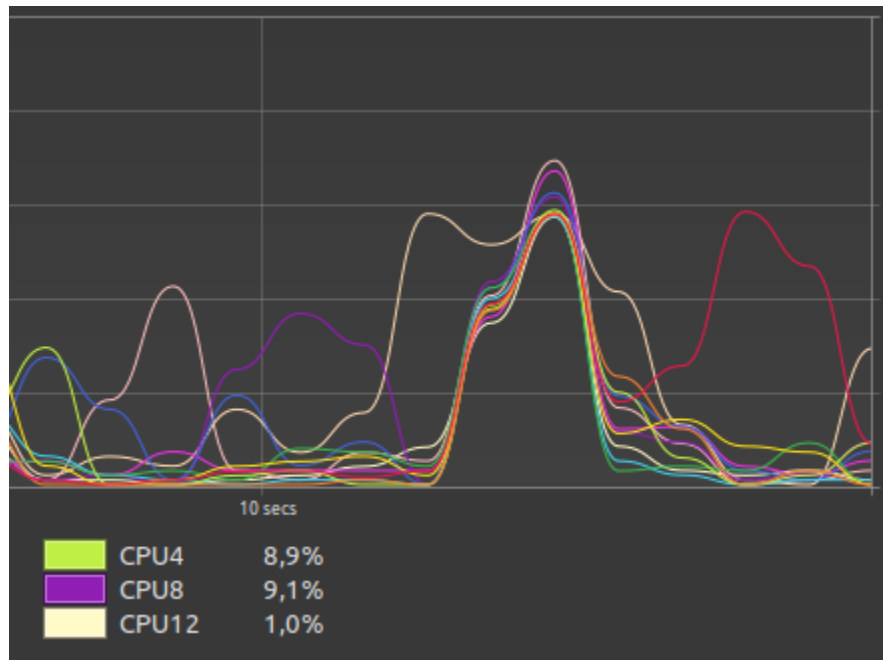
```

./source/graduation (7th copy)/Model Olusturma/Test-data/test4.csv is copied success
./source/graduation (8th copy)/Model Olusturma/data/telemetry.csv is copied successf
./source/graduation (7th copy)/Model Olusturma/data/PdM_errors.csv is copied success
./source/graduation (7th copy)/Model Olusturma/Test-data/test5.csv is copied success
./source/graduation (7th copy)/Model Olusturma/modelme.ipynb is copied successfully!
./source/graduation (6th copy)/Model Olusturma/data\dataexcluded is copied successfu
./source/graduation (4th copy)/Model Olusturma/data/telemetry.csv is copied successf
./source/graduation (6th copy)/Model Olusturma/data_2.csv is copied successfully!
./source/graduation (7th copy)/Model Olusturma/data/telemetry.csv is copied successf
./source/graduation (6th copy)/Model Olusturma/data/telemetry.csv is copied successf
./source/graduation (copy)/Model Olusturma/data/telemetry.csv is copied successfully

-----
Copying finished!
-----
Directories copied: 110
Regular files copied: 451
FIFO files copied: 0
Total bytes copied: 19300604226
Time taken: 1716979 µs
-----

bulut@bulut:~/Desktop/system5$
```

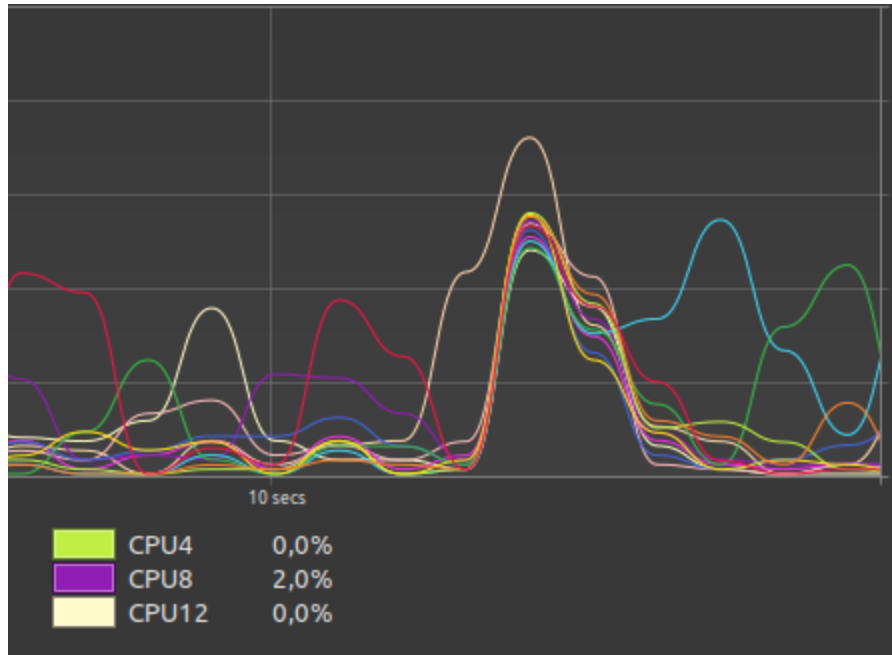
Consumer = 15, buffer size = 15



```
./source/graduation (7th copy)/Model Oluřturma/dat
./source/graduation (6th copy)/Model Oluřturma/dat
./source/graduation (7th copy)/Model Oluřturma/dat
./source/graduation (6th copy)/Model Oluřturma/dat
./source/graduation (copy)/Model Oluřturma/data/te
./source/graduation (8th copy)/Model Oluřturma/dat
./source/graduation (4th copy)/Model Oluřturma/dat
./source/graduation (7th copy)/Model Oluřturma/dat
./source/graduation (6th copy)/Model Oluřturma/dat

-----
| Copying finished!
|-----
| Directories copied: 110
| Regular files copied: 451
| FIFO files copied: 0
| Total bytes copied: 122329173984
| Time taken: 1787641 µs
|-----
bulut@bulut:~/Desktop/system5$
```

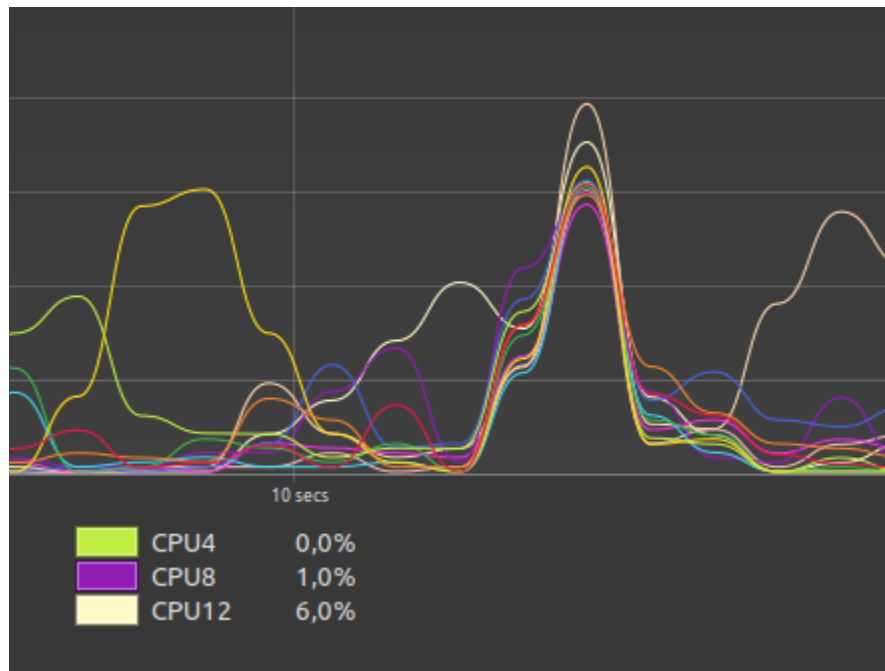
Consumer = 100, buffer size = 100



```
-----  
Copying finished!  
-----  
Directories copied: 110  
Regular files copied: 451  
FIFO files copied: 0  
Total bytes copied: 12917005843  
Time taken: 1681960 µs  
-----
```

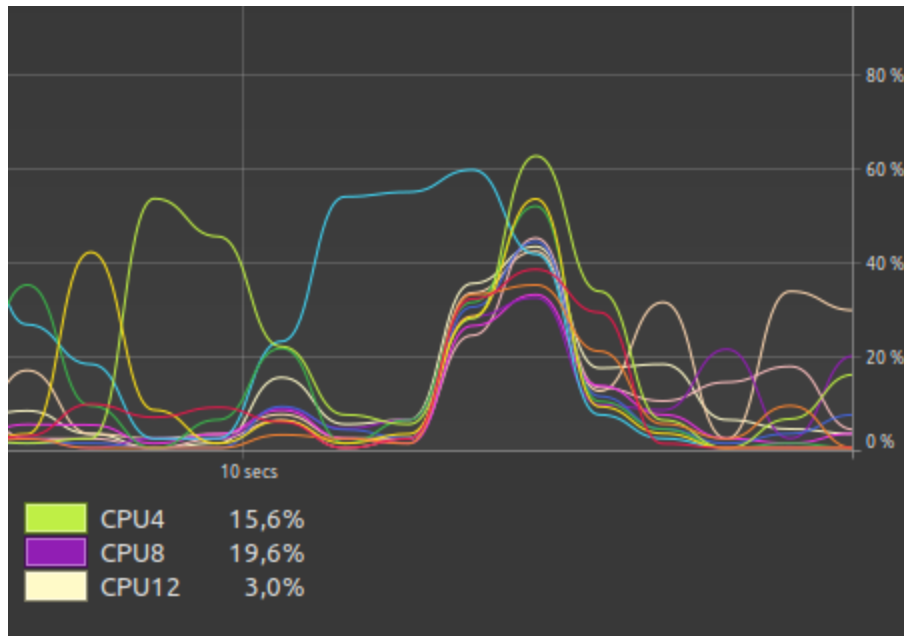
Consumer = 10, buffer size = 100





```
-----  
Copying finished!  
-----  
Directories copied: 110  
Regular files copied: 452  
FIFO files copied: 1  
Total bytes copied: 123729198434  
Time taken: 1630495 µs  
-----  
bulut@bulut:~/Desktop/system5$
```

Consumer = 100, buffer size = 10



```

./source/graduation (6th copy)/model-0.0.0.0/data/0
-----
Copying finished!
-----
Directories copied: 110
Regular files copied: 451
FIFO files copied: 0
Total bytes copied: 6502162557
Time taken: 1641703 µs
-----
bulut@bulut:~/Desktop/system5$

```

Consumer = 5, buffer size = 5

The buffer size and number of consumers are two important factors that can affect the performance of a program that uses multiple threads to copy files. A higher buffer size can reduce the number of times the producer thread needs to block, while a higher number of consumers can reduce the amount of time that the producer thread needs to spend waiting for all of the files to be read. However, both a large buffer size and a high number of consumers can have negative impacts on performance, such as increased memory usage or overhead. The ideal combination of buffer size and number of consumers will vary depending on the specific application, but it is important to find a balance that will improve performance without sacrificing other factors.