CSE 344 System Programming
Homework 5 Report

Mehmet Hüseyin YILDIZ
200104004095

1. Introduction

In this homework we are expected to implement a directory copy tool that creates threads. One producer threads traverses the given source directory and generates jobs. Consumer threads will take the jobs from the buffer and read from the source file descriptors and write to destination file descriptors. While designing the project I decided to implement a thread safe specialized queue data structure to ease the operations and to divide into parts. I will firstly explain this queue data structure and then producer and consumer. Let's dive into them.

2. Synchronized and Thread Safe Buffer Queue

This queue is ordinal queue implementation but it's thread safe and synchronized. The header file is shown in the below image.

queue.h > ...

```c
// Queue Implementation

# ifndef QUEUE_H
# define QUEUE_H

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>


typedef struct {
    int max_queue_size;
    void **data;
    int front;
    int rear;
    int size;
    pthread_mutex_t mutex;
    pthread_cond_t not_full;
    pthread_cond_t not_empty;

    short wait;

}queue;

void init_queue(queue *q, int max_queue_size);
void destruct_queue(queue *q);
void enqueue(queue *q, void *value);
void *dequeue(queue *q);
void print_queue(queue *q);
void unblock_queue(queue* q);

# endif
```

2.1 Race Conditions

The queue holds pointers as data. Thus, we can hold everything with its address. It has a pthread_mutex to prevent race conditions while adding and removing elements and also it is used to provide synchronization mechanism with condition variable. Enqueue func takes an element and add the element to the queue. Dequeue element removes an element and return it. In the start of enqueue and dequeu mutex is locked and in the end of them the mutex unlocked to prevent race conditions. Now we have a thread safe queue.

2.2 Producer Consumer Synchronization

We have a thread safe queue but its not enough still. Because we need a synchronization mechanism among the producer and consumers. When the queue is empty the dequeue caller should be blocked until one element added and also when the queue is full the enqueue caller should be blocked until one element is removed. And also the blocked dequeue should be released with a trigger function. Because when the all jobs are done. The consumer threads shouldnt wait any more and terminate. As told in the lecture I set up 2 condition variable not_full and not_empty. not_full is for enqueue and the not_empty is for dequeue. They are used with the same mutex. Thus, Enqueue waits until there is at least one empty space and the dequeu waits until there is at least one element in the queue. So, we have thread safe synchronized queue. The producer and consumers are synchronized.

2.3 Unblock Trigger

Now we have thread safe and synchronized queue but we have still one problem. When the all jobs are done, we must unblock the dequue function calls to terminate the consumer. For this purpose I implemented a function **unblock_queue.** I added a wait flag into queue structure and its true initially. So, the dequeue is only waits if the wait is enabled otherwise if the queue is empty the function return NULL. When the unblock_queue is called the wait flag is set as false and the not_empty condition variable is signaled with broadcast. So, the blocked all dequeue callers are unblocked and returns NULL.

```
// Unblock all waited dequeue func callers and return NULL value to them
void unblock_queue(queue* q) {
    pthread_mutex_lock(&q->mutex);
    q->wait = 0;
    pthread_cond_broadcast(&q->not_empty);
    pthread_mutex_unlock(&q->mutex);
}
```

So we have solved the synchronization problems with this specialized queue. The implementations of the enqueue and dequeue is show in the below.

```c
// Add an element to the queue
void enqueue(queue *q, void *value) {

    // Lock the queue mutex
    pthread_mutex_lock( &(q->mutex) );

    // Wait until the queue is has an empty space
    while (q->size == q->max_queue_size) {
        pthread_cond_wait(&q->not_full, &q->mutex);
    }

    // Add the element to the queue
    q->data[q->rear] = value;
    q->rear = (q->rear + 1) % q->max_queue_size;
    q->size++;

    // Signal the not_empty
    pthread_cond_signal(&q->not_empty);

    // Unlock the queue mutex
    pthread_mutex_unlock( &q->mutex );
}

// Remove element and return it from the queue
void *dequeue(queue *q) {

    // Lock the queue mutex
    pthread_mutex_lock( &q->mutex );

    // If the wait flag is enabled wait until there is a at least one element
    while (q->size == 0 && q->wait) {
        pthread_cond_wait(&q->not_empty, &q->mutex);
    }

    // If the size is 0 and wait is disabled
    if(q->size == 0)
    {
        pthread_mutex_unlock( &q->mutex );
        return NULL;
    }

    void *value = q->data[q->front];
    q->front = (q->front + 1) % q->max_queue_size;
    q->size--;

    pthread_cond_signal(&q->not_full);

    // Unlock the queue mutex
    pthread_mutex_unlock( &q->mutex );

    return value;
}
```

2.4 Queue Constructor and Destructor

The queue is initalized with **init_queue** function. The func takes max_size and initailizes the queue and mutex and condition variables dynamically.

```
void init_queue(queue *q, int max_size){

    q->max_queue_size = max_size;
    q->data = malloc(max_size * sizeof(void*));
    q->front = 0;
    q->rear = 0;
    q->size = 0;
    q->wait = 1;

    pthread_mutex_init(&q->mutex, NULL);
    pthread_cond_init(&q->not_full, NULL);
    pthread_cond_init(&q->not_empty, NULL);
}
```

When all jobs are done the queue can be destruct with destruct_queue function, that deallocates the buffer and destroys mutex and condition variables.

```
void destruct_queue(queue *q){

    free(q->data);
    pthread_mutex_destroy(&q->mutex);
    pthread_cond_destroy(&q->not_full);
    pthread_cond_destroy(&q->not_empty);
}
```

That’s all for the queue part, now let’s review the producer.

3. Producer

I implemented a produce function that takes source and destination and paths then create the destination directory. Then It opens the source directory and read all the files in the directory one by one.

```
void* produce(void* paths1){

    paths * path = (paths*) paths1;
    char *src_path = path->src_path;
    char *dest_path = path->dest_path;

    // Create the destination directory
    make_dir(dest_path);

    DIR *dir;
    struct dirent *entry;
    struct stat fileStat;

    // Open the directory
    dir = opendir(src_path);
    if (dir == NULL) {
        perror("Error opening directory");
        return NULL;
    }

    // Read directory entries one by one
    while ((entry = readdir(dir)) != NULL) {
```

The file entries are traversed in a loop and it gets the file information. If the file is a regular file it creates a file with additional O_TRUNC flag. So, it will truncate if the file exist in the destination. Then add the read fd and write fd and name to the buffer_queue by merged under a struct.

```
// Check if it's a regular file
if (S_ISREG(fileStat.st_mode)) {
    printf("File: %s\n", entry->d_name);

    // Increment the regular file count
    atomic_fetch_add(&regular_count,1);

    // Add the file to the buffer
    add_buffer(src_file_path, dest_file_path, entry->d_name);

}
```

```c
// Add the file operation to the buffer
void add_buffer(char *src_path, char *dest_path, char *file_name){

    buffer_entry *entry = malloc(sizeof(buffer_entry));

    strcpy(entry->name, file_name);
    entry->src_fd = open(src_path, O_RDONLY);
    entry->dest_fd = open(dest_path, O_WRONLY | O_CREAT | O_TRUNC, 0666);

    enqueue(&buffer_queue, entry);
}
```

If the file is a directory, then it will make a recursive call.

```c
// Check if it's a directory
else if (S_ISDIR(fileStat.st_mode)) {
    // Skip "." and ".." directories
    if (strcmp(entry->d_name, ".") != 0 && strcmp(entry->d_name, "..") != 0) {

        printf("Directory: %s\n", entry->d_name);

        // Increment the directory count
        atomic_fetch_add(&dir_count,1);

        // Recursive call to the directory
        paths path1 = {.dest_path = dest_file_path, .src_path = src_file_path };
        produce(&path1);

    }
}
```

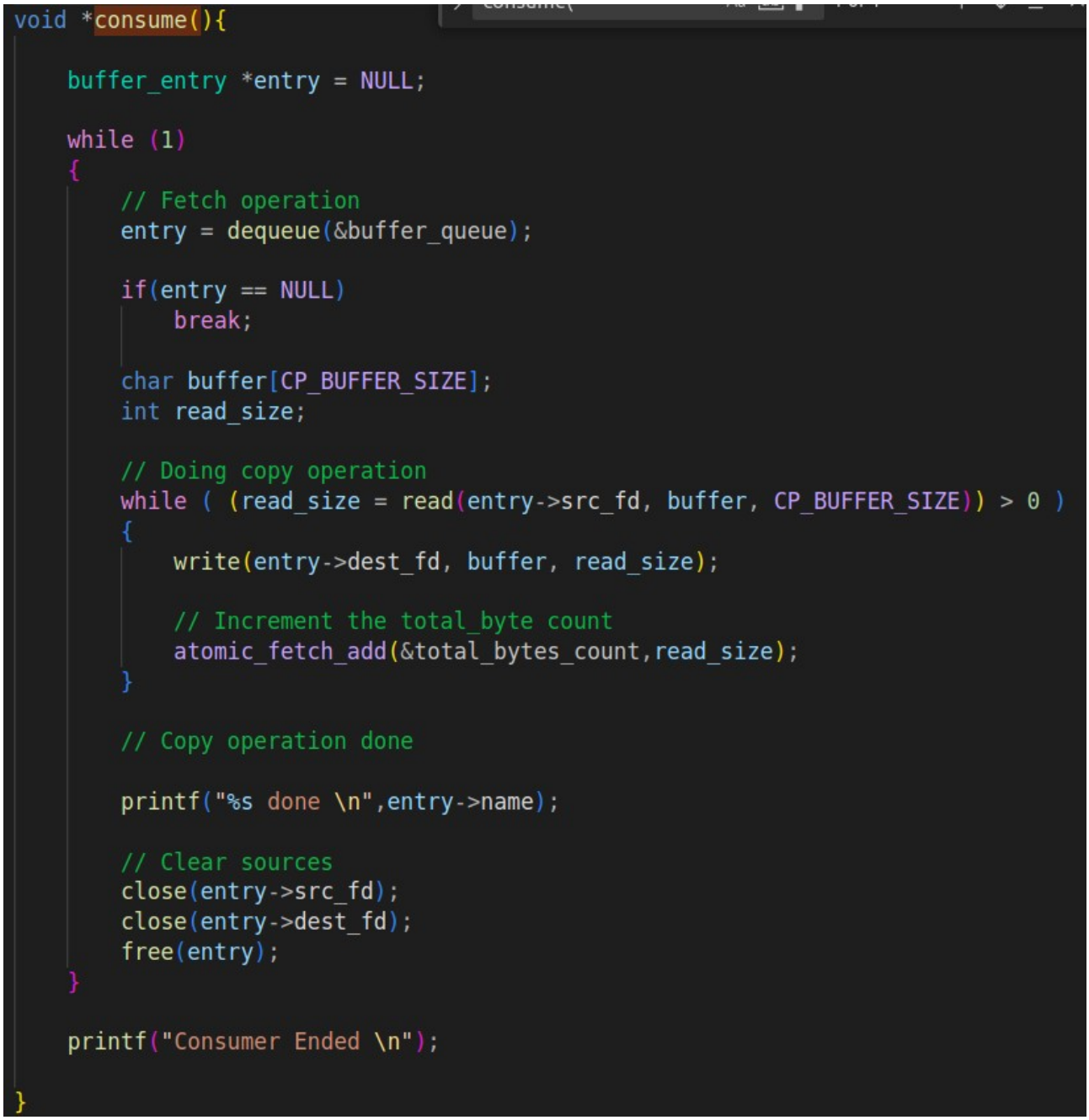
If the file is a fifo, then it will create same named fifo in the destination path.

```c
// Check if it's a fifo
else if (S_ISFIFO(fileStat.st_mode)) {
    printf("Fifo: %s\n", entry->d_name);

        // Increment the fifo count
        atomic_fetch_add(&fifo_count,1);

    mkfifo(dest_file_path, 0666);

}
```

We can extend these for other special file types too. And In the end of the produce function the open directory is closed and return.

4. Consumer

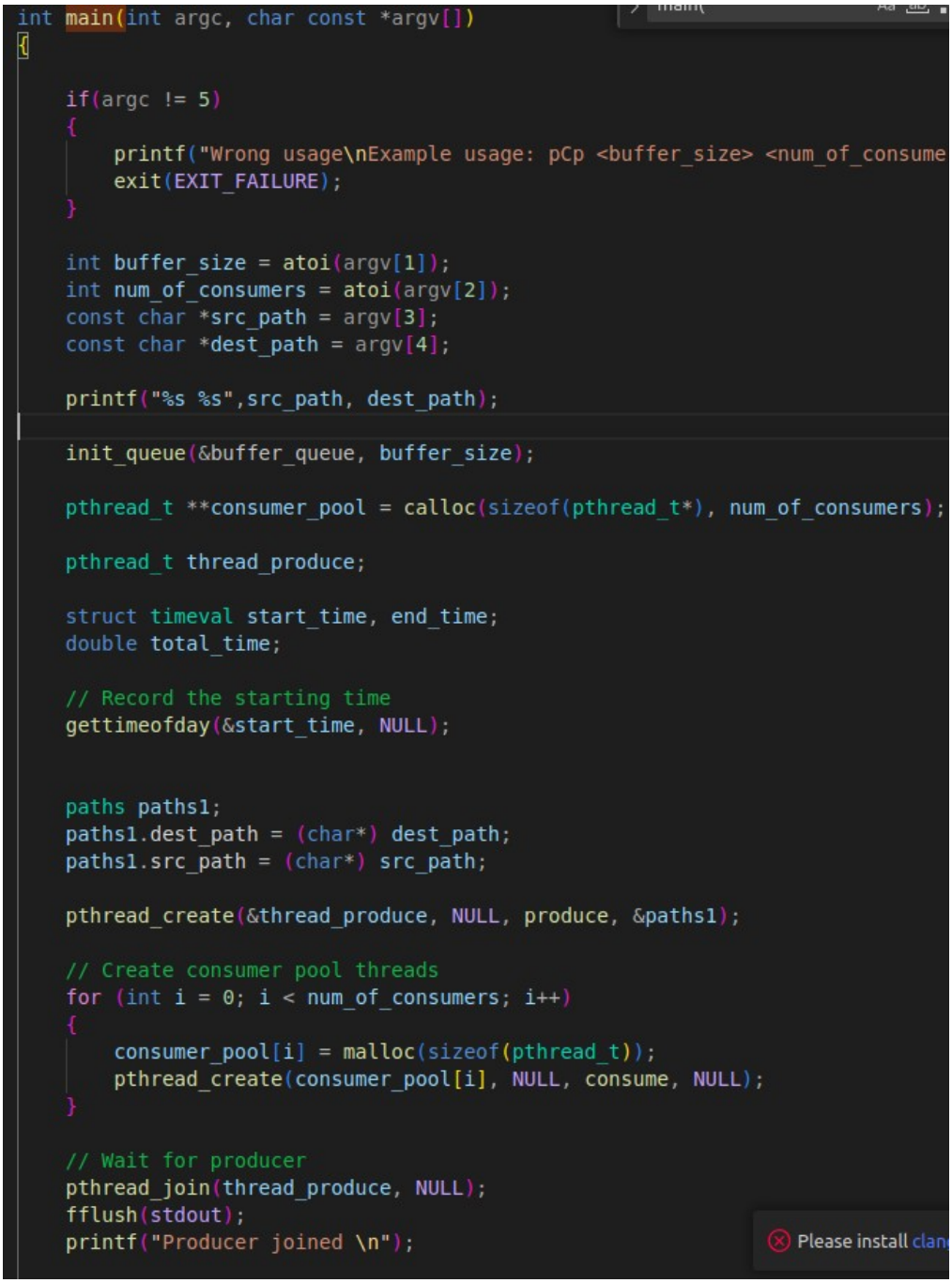
There is consume function for the consumer thread. It fetches a copy entry from the queue and if the entry is null breaks the loop and returns. If its not null, then it does the copy operation by reading from source fd and write to the destination fd. At the end of the copy operation, it close the 2 fd and deallocate the entry struct and returns.



```
void *consume(){

    buffer_entry *entry = NULL;

    while (1)
    {
        // Fetch operation
        entry = dequeue(&buffer_queue);

        if(entry == NULL)
            break;

        char buffer[CP_BUFFER_SIZE];
        int read_size;

        // Doing copy operation
        while ( (read_size = read(entry->src_fd, buffer, CP_BUFFER_SIZE)) > 0 )
        {
            write(entry->dest_fd, buffer, read_size);

            // Increment the total_byte count
            atomic_fetch_add(&total_bytes_count,read_size);
        }

        // Copy operation done

        printf("%s done \n",entry->name);

        // Clear sources
        close(entry->src_fd);
        close(entry->dest_fd);
        free(entry);
    }

    printf("Consumer Ended \n");

}
```

5. Main Function

In the main function, first arguments are checked then, queue is initialized, start time is fetched and consumer pool is created and producer thread and consumer threads created and then waits for the producer thread to join. After producer join, it will send the queue to unblock all blocked threads by calling unblock_queue function. After it, it waits all consumer threads to join and deallocate the consumer pool. Lastly it destruct the queue to deallocate all resources and record the end time and calculate the total time and prints all counters and elapsed time.



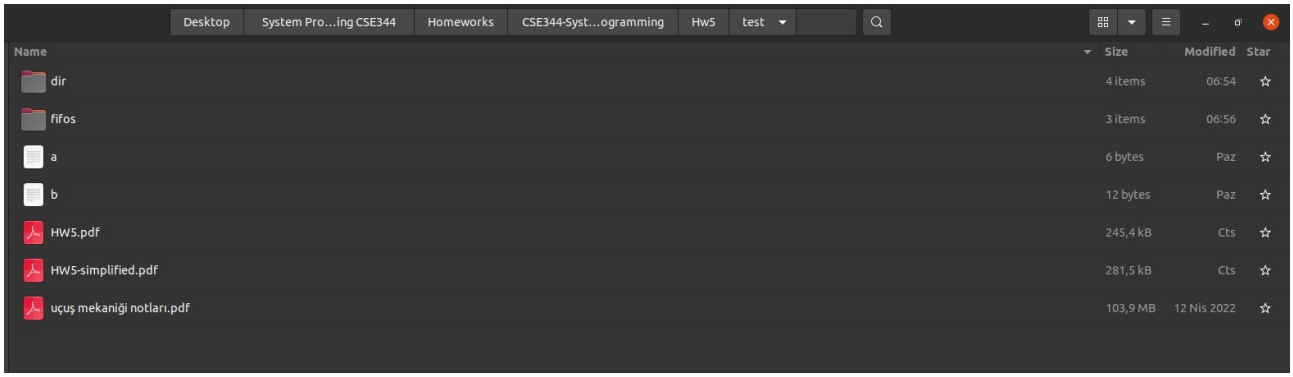
```c
int main(int argc, char const *argv[])
{

    if(argc != 5)
    {
        printf("Wrong usage\nExample usage: pCp <buffer_size> <num_of_consume
        exit(EXIT_FAILURE);
    }

    int buffer_size = atoi(argv[1]);
    int num_of_consumers = atoi(argv[2]);
    const char *src_path = argv[3];
    const char *dest_path = argv[4];

    printf("%s %s",src_path, dest_path);

    init_queue(&buffer_queue, buffer_size);

    pthread_t **consumer_pool = calloc(sizeof(pthread_t*), num_of_consumers);

    pthread_t thread_produce;

    struct timeval start_time, end_time;
    double total_time;

    // Record the starting time
    gettimeofday(&start_time, NULL);


    paths paths1;
    paths1.dest_path = (char*) dest_path;
    paths1.src_path = (char*) src_path;

    pthread_create(&thread_produce, NULL, produce, &paths1);

    // Create consumer pool threads
    for (int i = 0; i < num_of_consumers; i++)
    {
        consumer_pool[i] = malloc(sizeof(pthread_t));
        pthread_create(consumer_pool[i], NULL, consume, NULL);
    }

    // Wait for producer
    pthread_join(thread_produce, NULL);
    fflush(stdout);
    printf("Producer joined \n");
```

```c
    // Producing is end so unblock all waiting consumers
    unblock_queue(&buffer_queue);

    // Wait all consumers to join
    for (int i = 0; i < num_of_consumers; i++)
    {
        pthread_join(*consumer_pool[i], NULL);
        free(consumer_pool[i]);
    }

    free(consumer_pool);

    destruct_queue(&buffer_queue);

    printf("\nOperation done \n\n");

    // Record the ending time
    gettimeofday(&end_time, NULL);

    // Calculate the total time in milliseconds
    total_time = (end_time.tv_sec - start_time.tv_sec) * 1000.0 + (end_time.tv_usec - start_time.tv_u

    printf("Total elapsed time           : %.2f ms \n", total_time);
    printf("Total copied fifo files      : %d \n", fifo_count);
    printf("Total copied regular files   : %d \n", regular_count);
    printf("Total copied sub-directories : %d \n", dir_count);
    printf("Total copied number of bytes : %lld \n", total_bytes_count);

    return 0;
}
```

All counters are atomic so there can’t be any race condition while increasing them.

6. Tests

I created a test directory that contains regular files directories and fifos recursively. So the results of the tests are given in the images.



```
huseyin@huseyin-Inspiron-7577:~/Desktop/System Programming CSE344/Homeworks/CSE344-System-Programming/Hw5$ make clean
rm -f pCp
huseyin@huseyin-Inspiron-7577:~/Desktop/System Programming CSE344/Homeworks/CSE344-System-Programming/Hw5$ make
cc main.c queue.c -lpthread -o pCp
huseyin@huseyin-Inspiron-7577:~/Desktop/System Programming CSE344/Homeworks/CSE344-System-Programming/Hw5$ ./pCp
Wrong usage
Example usage: pCp <buffer_size> <num_of_consumers> <src_path> <dest_path>
huseyin@huseyin-Inspiron-7577:~/Desktop/System Programming CSE344/Homeworks/CSE344-System-Programming/Hw5$
```

Buffer size: 10 Consumer size: 5 Time : 539.4 ms

```
huseyin@huseyin-Inspiron-7577: ~/Desktop/System Programming CSE344/Homeworks/CSE344-System-Programming/Hw5
huseyin@huseyin-Inspiron-7577:~/Desktop/System Programming CSE344/Homeworks/CSE344-System-Programming/Hw5$ make clean
rm -f pCp
huseyin@huseyin-Inspiron-7577:~/Desktop/System Programming CSE344/Homeworks/CSE344-System-Programming/Hw5$ make
cc main.c queue.c -lpthread -o pCp
huseyin@huseyin-Inspiron-7577:~/Desktop/System Programming CSE344/Homeworks/CSE344-System-Programming/Hw5$ ./pCp 10 5 ./test/ ./testCP
Fifo: fifo2 done
Fifo: fifo1 done
Fifo: fifo3 done
Directory: fifos done
Regular File: b2 done
Regular File: a2 done
Directory: dir2 done
Regular File: b1 done
Directory: dir done
Regular File: a1 done
Regular File: b done
Regular File: a done
Regular File: HW5.pdf done
Regular File: HW5-simplified.pdf done
Regular File: uçuş mekaniği notları2.pdf done
Regular File: uçuş mekaniği notları.pdf done
Regular File: uçuş mekaniği notları1.pdf done

Operation done

Total elapsed time           : 539.39 ms
Total copied fifo files      : 3
Total copied regular files   : 11
Total copied sub-directories : 3
Total copied number of bytes : 312259176
huseyin@huseyin-Inspiron-7577:~/Desktop/System Programming CSE344/Homeworks/CSE344-System-Programming/Hw5$
```

Buffer size: 1 Consumer size: 5 Time : 574.7 ms

```
huseyin@huseyin-Inspiron-7577:~/Desktop/System Programming CSE344/Homeworks/CSE344-System-Programming/Hw5$ ./pCp 1 5 ./test/ ./testCP
Fifo: fifo2 done
Fifo: fifo1 done
Fifo: fifo3 done
Directory: fifos done
Regular File: b2 done
Regular File: a2 done
Directory: dir2 done
Regular File: b1 done
Directory: dir done
Regular File: a1 done
Regular File: b done
Regular File: a done
Regular File: HW5.pdf done
Regular File: HW5-simplified.pdf done
Regular File: uçuş mekaniği notları.pdf done
Regular File: uçuş mekaniği notları2.pdf done
Regular File: uçuş mekaniği notları1.pdf done

Operation done

Total elapsed time           : 574.69 ms
Total copied fifo files      : 3
Total copied regular files   : 11
Total copied sub-directories : 3
Total copied number of bytes : 312259176
```

File copy with not existing source directory

```
huseyin@huseyin-Inspiron-7577:~/Desktop/System Programming CSE344/Homeworks/CSE344-System-Programming/HW5$ ./pCp 1 5 ./te ./testCP
Error opening directory: No such file or directory

Operation done

Total elapsed time           : 0.84 ms
Total copied fifo files      : 0
Total copied regular files   : 0
Total copied sub-directories : 0
Total copied number of bytes : 0
huseyin@huseyin-Inspiron-7577:~/Desktop/System Programming CSE344/Homeworks/CSE344-System-Programming/HW5$
```

Buffer size: 1 Consumer size: 20 Time : 537.9 ms

```
huseyin@huseyin-Inspiron-7577:~/Desktop/System Programming CSE344/Homeworks/CSE344-System-Programming/HW5$ ./pCp 1 20 ./test ./testCP
Fifo: fifo2 done
Fifo: fifo1 done
Fifo: fifo3 done
Directory: fifos done
Regular File: b2 done
Regular File: a2 done
Directory: dir2 done
Regular File: b1 done
Regular File: a1 done
Directory: dir done
Regular File: b done
Regular File: a done
Regular File: HW5.pdf done
Regular File: HW5-simplified.pdf done
Regular File: uçuş mekaniği notları2.pdf done
Regular File: uçuş mekaniği notları1.pdf done
Regular File: uçuş mekaniği notları.pdf done

Operation done

Total elapsed time           : 537.89 ms
Total copied fifo files      : 3
Total copied regular files   : 11
Total copied sub-directories : 3
Total copied number of bytes : 312259176
huseyin@huseyin-Inspiron-7577:~/Desktop/System Programming CSE344/Homeworks/CSE344-System-Programming/HW5$
```

Buffer size: 10 Consumer size: 20 Time : 564.9 ms

```
huseyin@huseyin-Inspiron-7577:~/Desktop/System Programming CSE344/Homeworks/CSE344-System-Programming/HW5$ ./pCp 10 20 ./test ./testCP
Fifo: fifo2 done
Fifo: fifo1 done
Fifo: fifo3 done
Directory: fifos done
Regular File: b2 done
Regular File: a2 done
Directory: dir2 done
Regular File: b1 done
Regular File: a1 done
Directory: dir done
Regular File: HW5.pdf done
Regular File: b done
Regular File: a done
Regular File: HW5-simplified.pdf done
Regular File: uçuş mekaniği notları2.pdf done
Regular File: uçuş mekaniği notları1.pdf done
Regular File: uçuş mekaniği notları.pdf done

Operation done

Total elapsed time           : 564.94 ms
Total copied fifo files      : 3
Total copied regular files   : 11
Total copied sub-directories : 3
Total copied number of bytes : 312259176
```

Buffer size: 10 Consumer size: 200 Time : 565.6 ms

```
huseyin@huseyin-Inspiron-7577:~/Desktop/System Programming CSE344/Homeworks/CSE344-System-Programming/HW5$ ./pCp 10 200 ./test ./testCP
Fifo: fifo2 done
Fifo: fifo1 done
Fifo: fifo3 done
Directory: fifos done
Regular File: b2 done
Regular File: a2 done
Directory: dir2 done
Regular File: b1 done
Regular File: a1 done
Directory: dir done
Regular File: HW5.pdf done
Regular File: a done
Regular File: b done
Regular File: HW5-simplified.pdf done
Regular File: uçuş mekaniği notları2.pdf done
Regular File: uçuş mekaniği notları1.pdf done
Regular File: uçuş mekaniği notları.pdf done

Operation done

Total elapsed time           : 565.58 ms
Total copied fifo files      : 3
Total copied regular files   : 11
Total copied sub-directories : 3
Total copied number of bytes : 312259176
```

Buffer size: 2 Consumer size: 20 Time : 578.4 ms

```
huseyin@huseyin-Inspiron-7577:~/Desktop/System Programming CSE344/Homeworks/CSE344-System-Programming/HW5$ ./pCp 2 20 ./test ./testCP
Fifo: fifo2 done
Fifo: fifo1 done
Fifo: fifo3 done
Directory: fifos done
Regular File: b2 done
Regular File: a2 done
Directory: dir2 done
Regular File: b1 done
Regular File: a1 done
Directory: dir done
Regular File: HW5.pdf done
Regular File: b done
Regular File: a done
Regular File: HW5-simplified.pdf done
Regular File: uçuş mekaniği notları2.pdf done
Regular File: uçuş mekaniği notları1.pdf done
Regular File: uçuş mekaniği notları.pdf done

Operation done

Total elapsed time           : 578.43 ms
Total copied fifo files      : 3
Total copied regular files   : 11
Total copied sub-directories : 3
Total copied number of bytes : 312259176
```

Buffer size: 2 Consumer size: 3 Time : 578.55 ms

```
huseyin@huseyin-Inspiron-7577:~/Desktop/System Programming CSE344/Homeworks/CSE344-System-Programming/HW5$ ./pCp 2 3 ./test ./testCP
Fifo: fifo2 done
Fifo: fifo1 done
Fifo: fifo3 done
Directory: fifos done
Regular File: b2 done
Regular File: a2 done
Directory: dir2 done
Regular File: b1 done
Directory: dir done
Regular File: uçuş mekaniği notları2.pdf done
Regular File: uçuş mekaniği notları1.pdf done
Regular File: uçuş mekaniği notları.pdf done
Regular File: a1 done
Regular File: HW5.pdf done
Regular File: b done
Regular File: a done
Regular File: HW5-simplified.pdf done

Operation done

Total elapsed time           : 570.55 ms
Total copied fifo files      : 3
Total copied regular files   : 11
Total copied sub-directories : 3
Total copied number of bytes : 312259176
```

Buffer size: 1 Consumer size: 1 Time : 1244.56 ms

```
huseyin@huseyin-Inspiron-7577:~/Desktop/System Programming CSE344/Homeworks/CSE344-System-Programming/HW5$ ./pCp 1 1 ./test ./testCP
Fifo: fifo2 done
Fifo: fifo1 done
Fifo: fifo3 done
Directory: fifos done
Regular File: uçuş mekaniği notları.pdf done
Regular File: b2 done
Regular File: a2 done
Directory: dir2 done
Regular File: uçuş mekaniği notları2.pdf done
Regular File: b1 done
Directory: dir done
Regular File: uçuş mekaniği notları1.pdf done
Regular File: a1 done
Regular File: HW5.pdf done
Regular File: b done
Regular File: a done
Regular File: HW5-simplified.pdf done

Operation done

Total elapsed time           : 1244.56 ms
Total copied fifo files      : 3
Total copied regular files   : 11
Total copied sub-directories : 3
Total copied number of bytes : 312259176
```

Buffer size: 1 Consumer size: 2 Time : 961.4 ms

```
huseyin@huseyin-Inspiron-7577:~/Desktop/System Programming CSE344/Homeworks/CSE344-System-Programming/HW5$ ./pCp 1 2 ./test ./testCP
Fifo: fifo2 done
Fifo: fifo1 done
Fifo: fifo3 done
Directory: fifos done
Regular File: b2 done
Regular File: a2 done
Directory: dir2 done
Regular File: uçuş mekaniği notları2.pdf done
Regular File: uçuş mekaniği notları.pdf done
Regular File: b1 done
Directory: dir done
Regular File: a1 done
Regular File: HW5.pdf done
Regular File: b done
Regular File: a done
Regular File: HW5-simplified.pdf done
Regular File: uçuş mekaniği notları1.pdf done

Operation done

Total elapsed time           : 961.39 ms
Total copied fifo files      : 3
Total copied regular files   : 11
Total copied sub-directories : 3
Total copied number of bytes : 312259176
```

Buffer size: 2 Consumer size: 1 Time : 1229.2 ms

```
huseyin@huseyin-Inspiron-7577:~/Desktop/System Programming CSE344/Homeworks/CSE344-System-Programming/HW5$ ./pCp 2 1 ./test ./testCP
Fifo: fifo2 done
Fifo: fifo1 done
Fifo: fifo3 done
Directory: fifos done
Regular File: uçuş mekaniği notları.pdf done
Regular File: b2 done
Directory: dir2 done
Regular File: a2 done
Regular File: uçuş mekaniği notları2.pdf done
Regular File: b1 done
Directory: dir done
Regular File: uçuş mekaniği notları1.pdf done
Regular File: a1 done
Regular File: HW5.pdf done
Regular File: b done
Regular File: a done
Regular File: HW5-simplified.pdf done

Operation done

Total elapsed time           : 1229.23 ms
Total copied fifo files      : 3
Total copied regular files   : 11
Total copied sub-directories : 3
Total copied number of bytes : 312259176
```

Buffer size: 2 Consumer size: 2 Time : 852.9 ms

```
huseyin@huseyin-Inspiron-7577:~/Desktop/System Programming CSE344/Homeworks/CSE344-System-Programming/Hw5$ ./pCp 2 2 ./test ./testCP
Fifo: fifo2 done
Fifo: fifo1 done
Fifo: fifo3 done
Directory: fifos done
Regular File: b2 done
Regular File: a2 done
Directory: dir2 done
Regular File: uçuş mekaniği notları2.pdf done
Regular File: uçuş mekaniği notları.pdf done
Directory: dir done
Regular File: b1 done
Regular File: a1 done
Regular File: HW5.pdf done
Regular File: b done
Regular File: a done
Regular File: HW5-simplified.pdf done
Regular File: uçuş mekaniği notları1.pdf done

Operation done

Total elapsed time           : 852.91 ms
Total copied fifo files      : 3
Total copied regular files   : 11
Total copied sub-directories : 3
Total copied number of bytes : 312259176
```

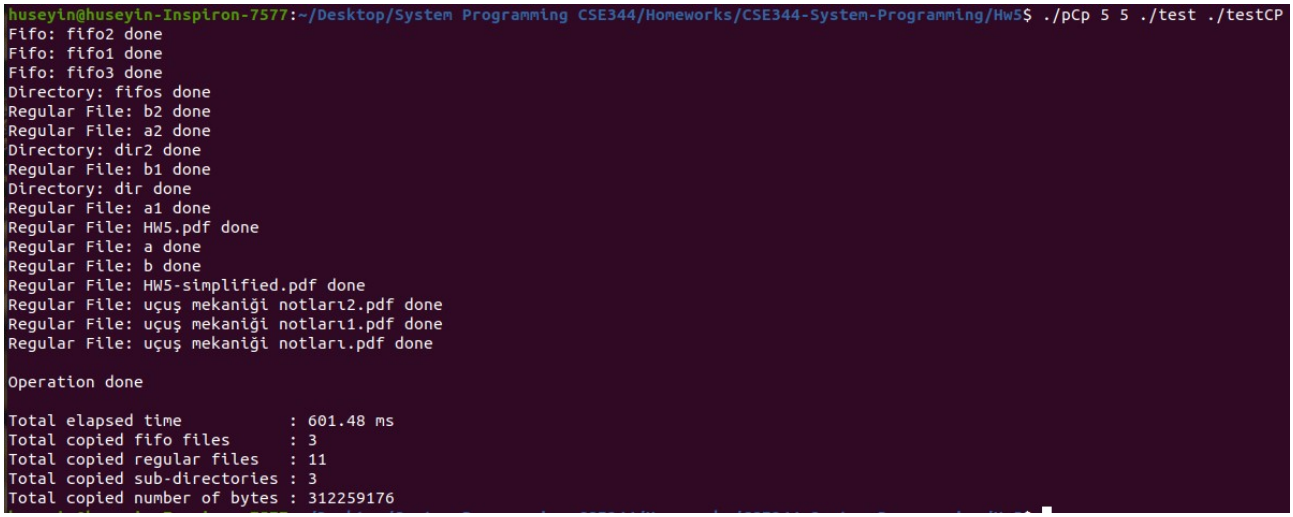
Buffer size: 3 Consumer size: 3 Time : 533.85 ms

```
huseyin@huseyin-Inspiron-7577:~/Desktop/System Programming CSE344/Homeworks/CSE344-System-Programming/Hw5$ ./pCp 3 3 ./test ./testCP
Fifo: fifo2 done
Fifo: fifo1 done
Fifo: fifo3 done
Directory: fifos done
Regular File: b2 done
Regular File: a2 done
Directory: dir2 done
Regular File: b1 done
Directory: dir done
Regular File: uçuş mekaniği notları2.pdf done
Regular File: uçuş mekaniği notları1.pdf done
Regular File: uçuş mekaniği notları.pdf done
Regular File: a1 done
Regular File: HW5.pdf done
Regular File: b done
Regular File: a done
Regular File: HW5-simplified.pdf done

Operation done

Total elapsed time           : 533.85 ms
Total copied fifo files      : 3
Total copied regular files   : 11
Total copied sub-directories : 3
Total copied number of bytes : 312259176
```

Buffer size: 4 Consumer size: 4 Time : 556.97 ms

```
huseyin@huseyin-Inspiron-7577:~/Desktop/System Programming CSE344/Homeworks/CSE344-System-Programming/Hw5$ ./pCp 4 4 ./test ./testCP
Fifo: fifo2 done
Fifo: fifo1 done
Fifo: fifo3 done
Directory: fifos done
Regular File: a2 done
Regular File: b2 done
Directory: dir2 done
Regular File: b1 done
Directory: dir done
Regular File: a1 done
Regular File: HW5.pdf done
Regular File: b done
Regular File: a done
Regular File: HW5-simplified.pdf done
Regular File: uçuş mekaniği notları2.pdf done
Regular File: uçuş mekaniği notları1.pdf done
Regular File: uçuş mekaniği notları.pdf done

Operation done

Total elapsed time           : 556.97 ms
Total copied fifo files      : 3
Total copied regular files   : 11
Total copied sub-directories : 3
Total copied number of bytes : 312259176
```

Buffer size: 5 Consumer size: 5 Time : 601.5 ms



Conclusion

So, the tests showed that, When we increase the number of consumers the performance is increased until file size. After number of file size it doesn't increase the performance better. The best scenario is when we increase the buffer size and consumer size proportionally. Because lack of one side will be bottleneck for the other one.

That's all I done in this homework. I tried to do best and I cover everything I done in this report. Thanks.