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

Emirkan Burak Yılmaz
1901042659

Table of Contents

1 Critical Sections

1.1 stdout and stderr

To provide mutual exclusion for the standard output and standard error, print_threadsafe function is writtern. It takes the file descriptor and the mutex and the print format. An example usage for stdout could be print_threadsafe(STDOUT_FILENO, &mutex_stdout, "Hello Netherlands\n");

```c
int print_threadsafe(int fd, pthread_mutex_t *fd_mutex, const char *format, ...)
{
    va_list args;
    int numbytes;

    if (pthread_mutex_lock(fd_mutex) != 0) {
        fprintf(stderr, "Failed to lock the mutex\n");
        return -1;
    }

    va_start(args, format);
    numbytes = vdprintf(fd, format, args);
    va_end(args);

    if (pthread_mutex_unlock(fd_mutex) != 0) {
        fprintf(stderr, "Failed to unlock the mutex\n");
        return -1;
    }

    return numbytes;
}
```

1.2 Shared buffer

Producer thread opens the source an——d destination files and push these open file descriptors with their file names to the shared job buffer. Consumer threads pop from the job buffer and apply read/write processes to successfully copy source file to destination file. Access to this shared buffer is a critical region between producer and consumers. To prevent race condition and provide mutual exclusion, critical region is protected with a mutex, and the empty/full buffer conditions are handled with two condition variables cond_empty and cond_full.

```c
if (pthread_mutex_lock(&mutex_jobs_buff) != 0) {
    print_threadsafe(STDERR_FILENO, &mutex_stderr, "Failed to lock the mutex\n");
    exit(EXIT_FAILURE);
}

/* Add the copy structure to the producer-consumer buffer */
while (buff_push(&jobs_buff, job) == -1) {
    if (pthread_cond_wait(&cond_empty, &mutex_jobs_buff) != 0) {
        print_threadsafe(STDERR_FILENO, &mutex_stderr, "Failed to make condtional wait\n");
        exit(EXIT_FAILURE);
    }
}

if (pthread_cond_signal(&cond_full) != 0) {
    print_threadsafe(STDERR_FILENO, &mutex_stderr, "Failed to make conditional signal\n");
    exit(EXIT_FAILURE);
}

if (pthread_mutex_unlock(&mutex_jobs_buff) != 0) {
    print_threadsafe(STDERR_FILENO, &mutex_stderr, "Failed to unlock the mutex\n");
    exit(EXIT_FAILURE);
}
```

producer pushes new copy job

2 Special File Types

Producer thread checks type of the file and acts according to its type. For the regular files they are passed to the consumers via the shared job buffer. For the FIFO files, producer creates the files at the destination and continues with the next file/directory. For a symbolic link file, its link is first read and then created a new symbolic link file at the destination with that link.

```c
/**
 * The seven standard Unix file types are regular, directory, symbolic link,
 * FIFO special, block special, character special, and socket as defined by POSIX
 */
You, 47 minutes ago | 1 author (You)
struct cp_stats {
    int num_cons;
    int buff_size;
    int num_dir;
    int num_regfile;
    int num_slink;
    int num_fifo;
    int num_unsupported;
    long totalbytes;
    double elapsed_time;
};
```

3 Open File Descriptor Limit

At the beginning of the main program a counting semaphore is created with the initial value as the maximum number of open file descriptor. This value is gathered by the getrlimit() from sys/resource.h library. The producer thread downs this semaphore before opening a file and the consumers up the semaphore after close calls on source and destination file descriptors. With this way the number of maximum open file never exceeds, and no error will occur. Another possible solution could be resetting the buffer size according to current # of max open file. By doing that we never reach the max

limit. However, this seems to me not user friendly, for that reason I prefer using counting semaphore for handling exceeding fd limit.

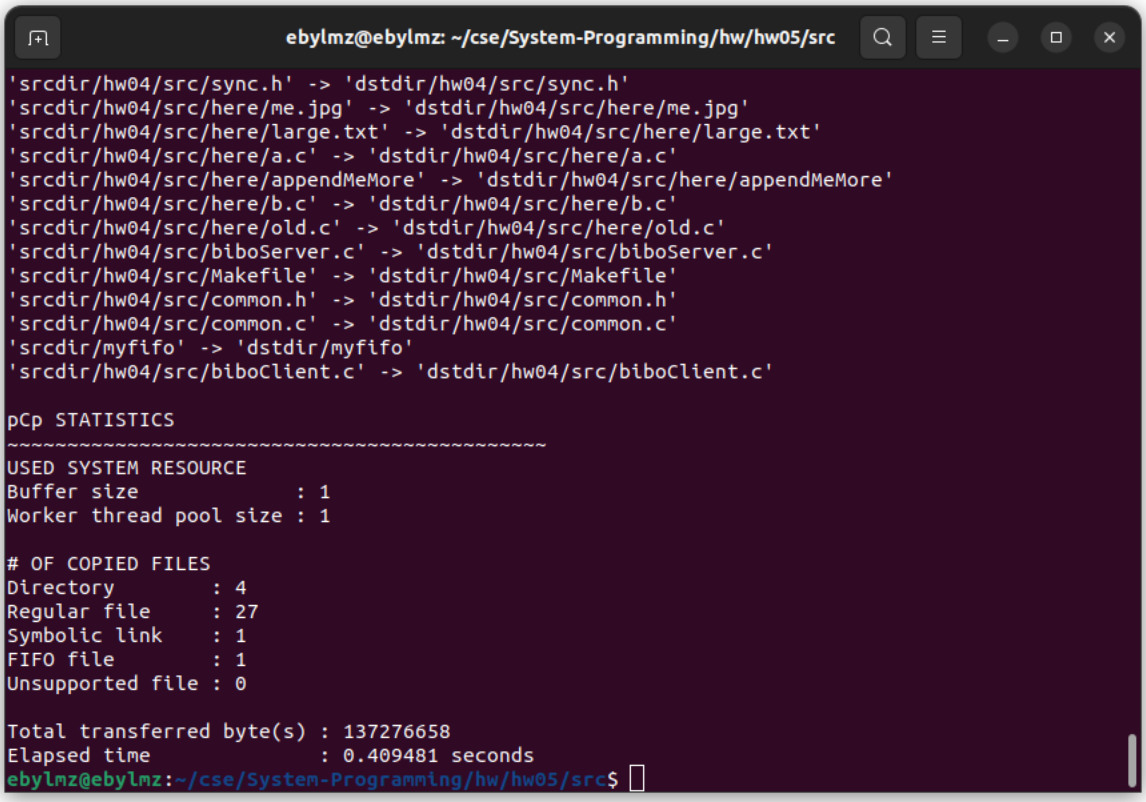
4 Signal Handling

A signal handler written for the signals SIGTERM and SIGINT. The handler sets a global flag variable which are checked on both producer and consumer threads. When the producer thread realizes this flag is up, then it stops reading directory and make conditional broadcast for consumer thread for graceful termination. When the consumer thread realizes it, stop checking the jobs buffer and closes all the resources and exits gracefully.

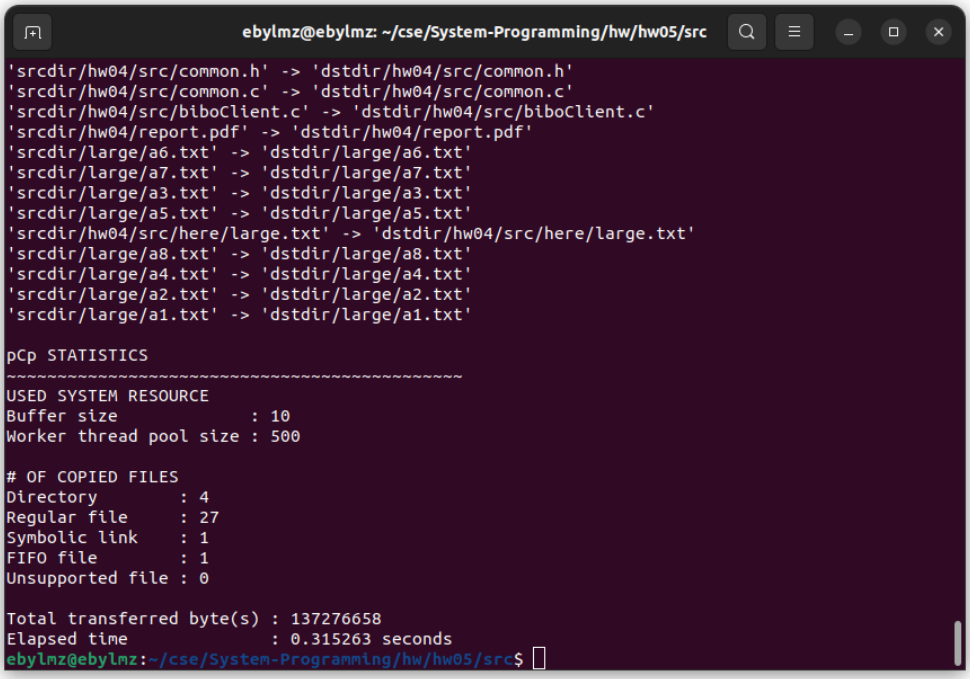
5 Experimenting Buffer and Number of Consumer parameters

In general, using a big/moderate buffer and keeping the size of thread pool moderate gives best performance. Even if the size of thread pool is big, if the buffer size is small then having big thread pool is just a waste of system resources. On the other hand, if the buffer is large and the thread pool is small then the buffer will quickly fill, and the producer blocked until a consumer pop from the buffer and wakes up the producer. Since the blocking chance of producer increase with the small buffer sizes, we cannot reach the optimum performance with these parameters.

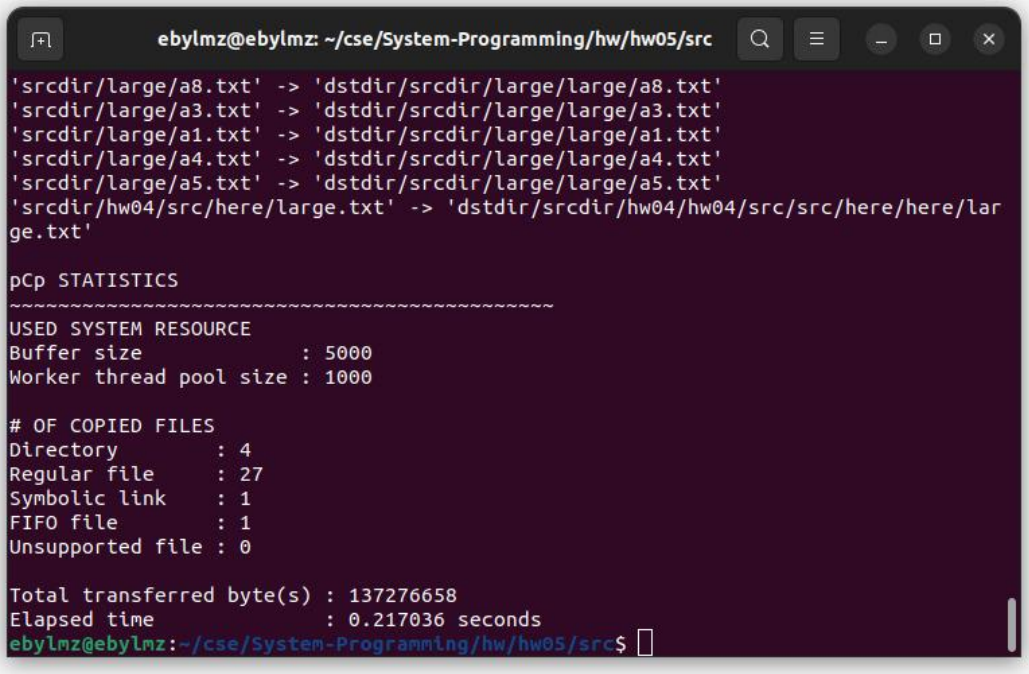
5.1 Small Buffer, Small Consumer Thread Pool



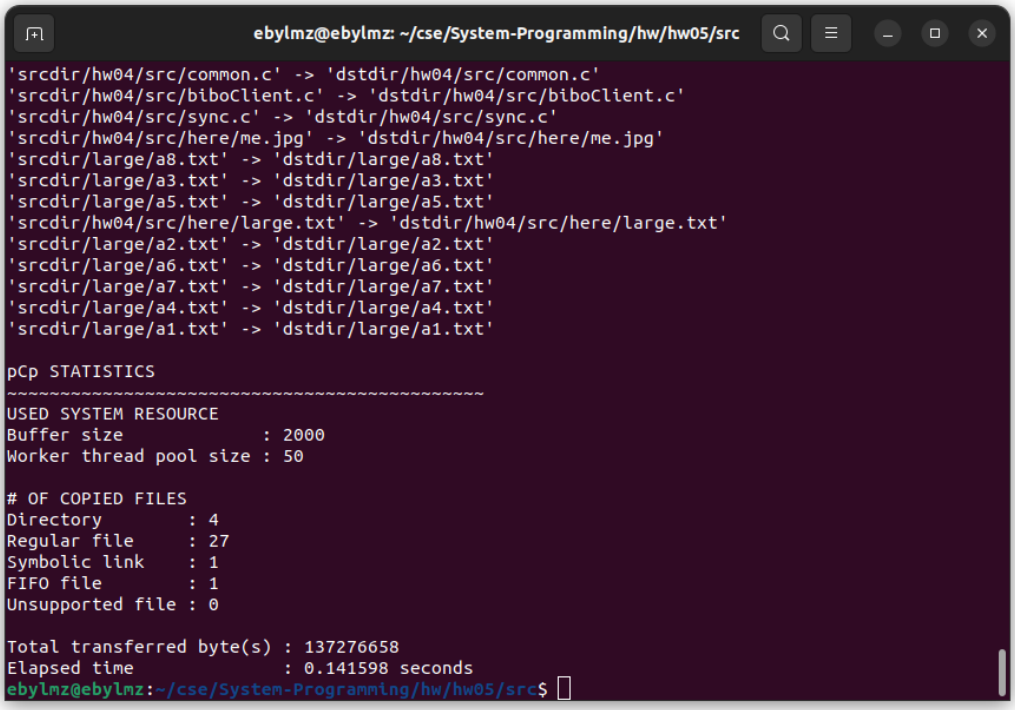
5.2 Small Buffer, Large Consumer Thread Pool



5.3 Large Buffer, Large Thread Pool



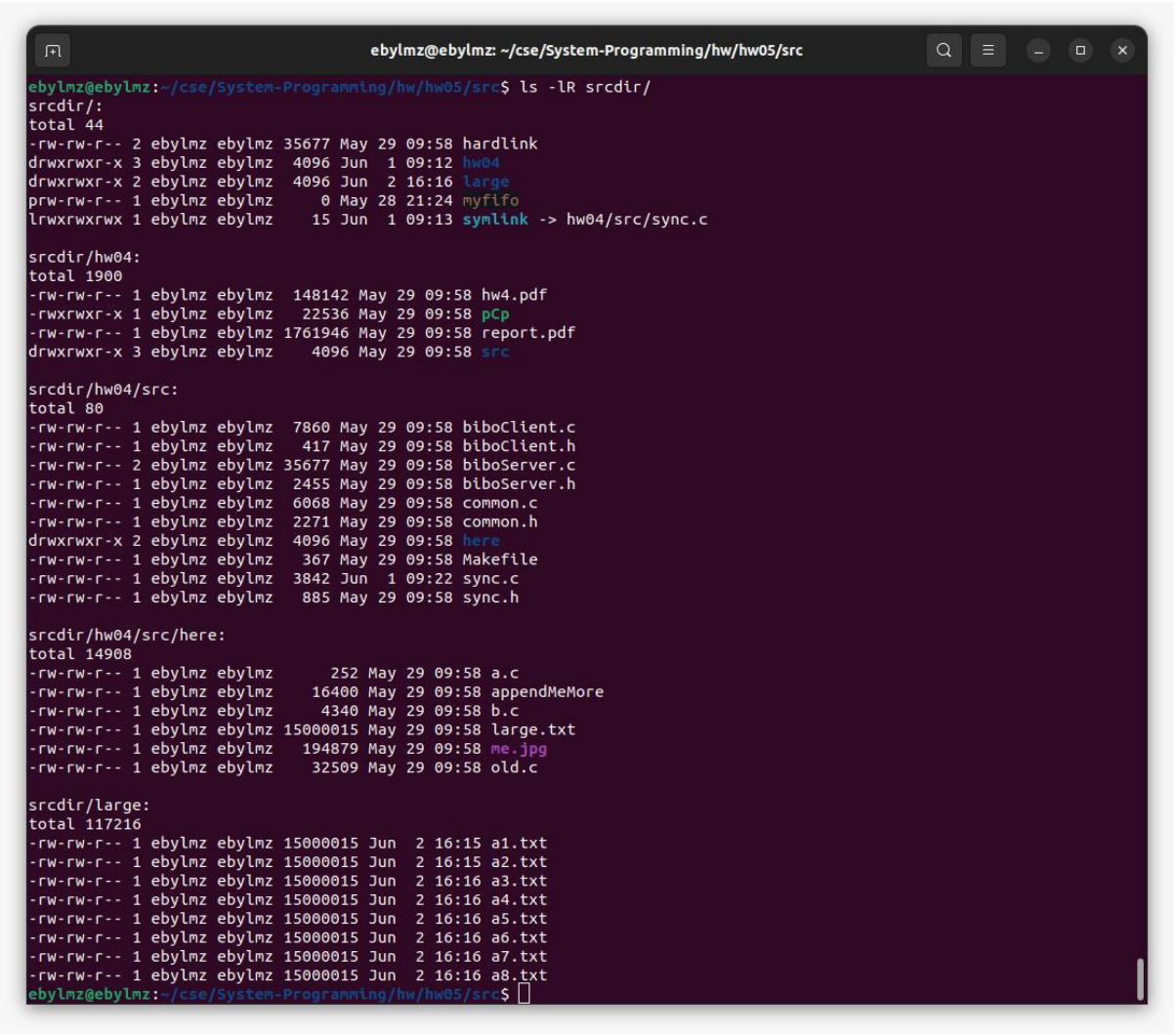
5.4 Large Buffer, Moderate Thread Pool



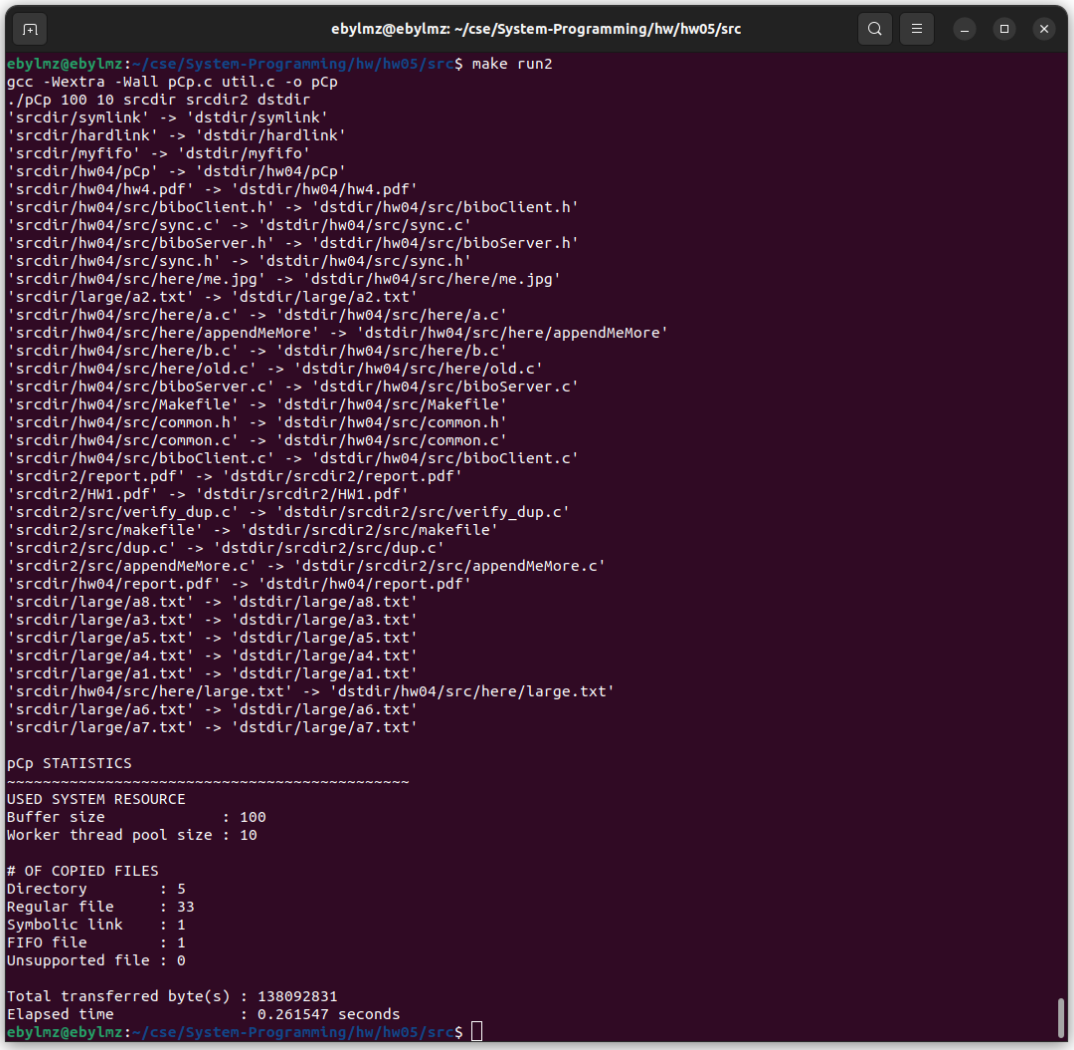
6 Test Cases & Results

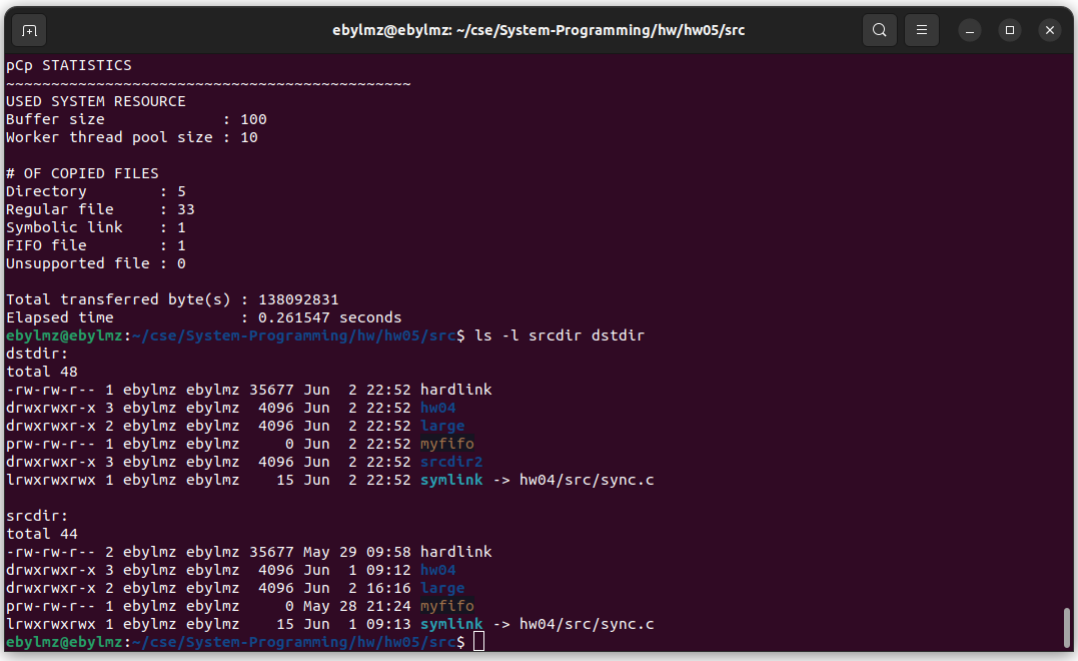
6.1 Content of Source Directory

Source directory contains 1 FIFO file, 1 symbolic link and several regular files in different format (text/binary) with different sizes.

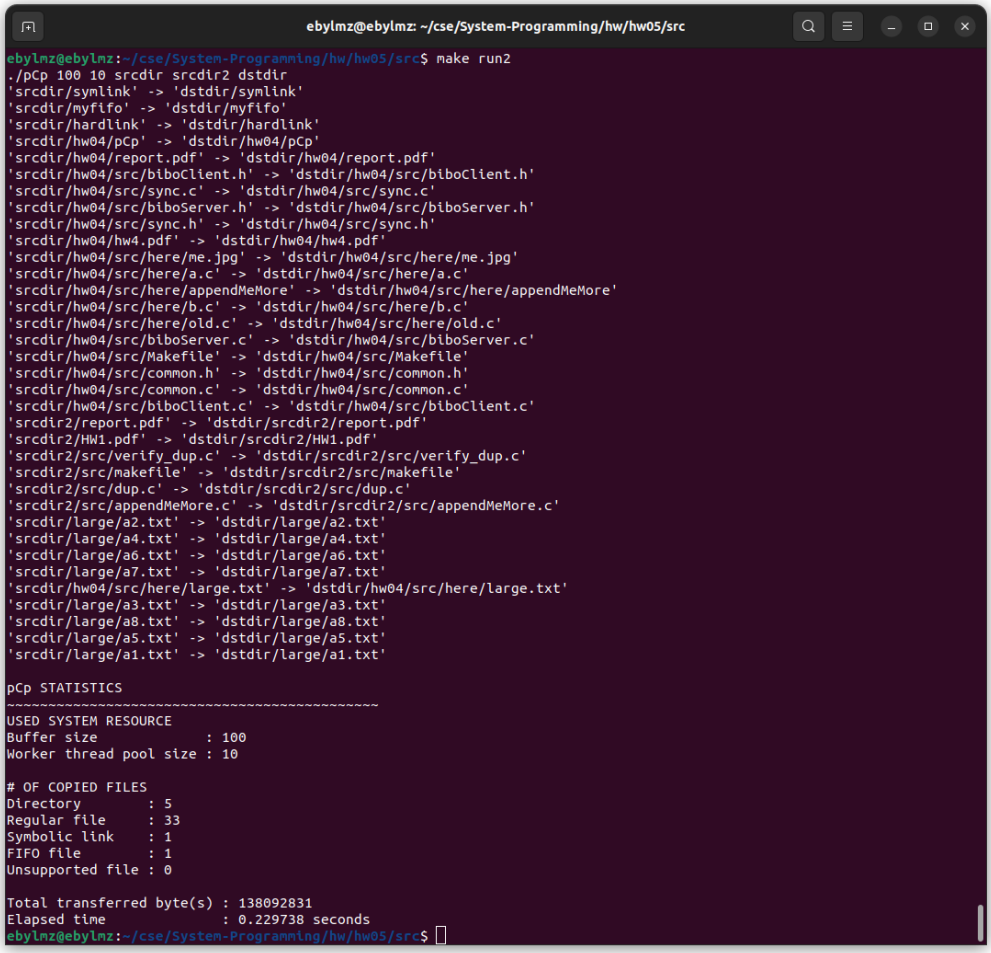


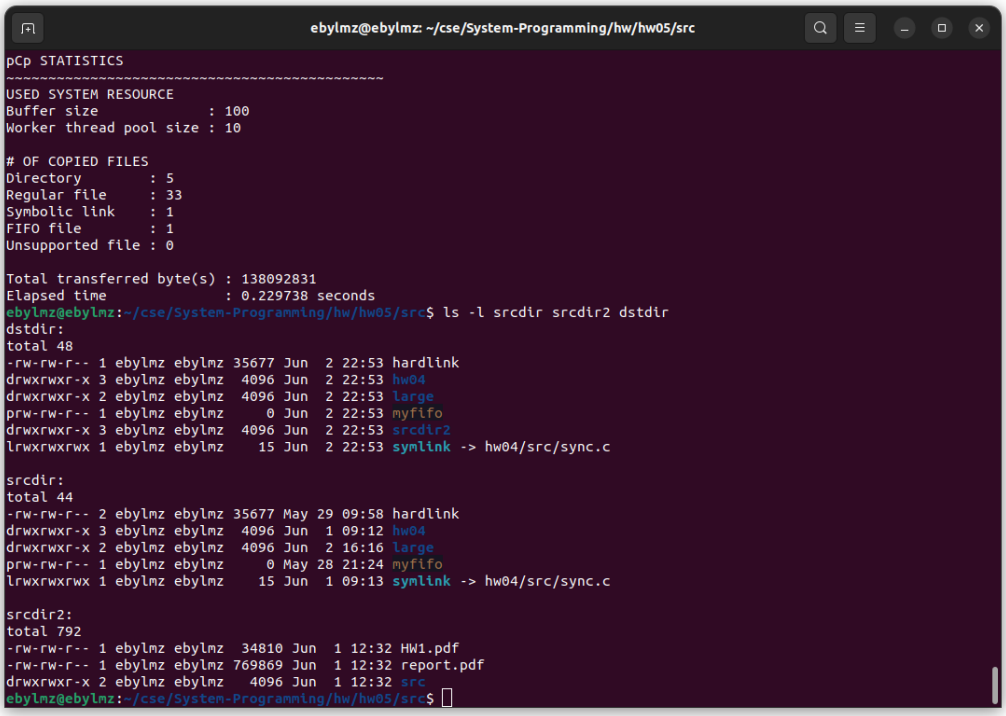
6.2 Single Source to Destination



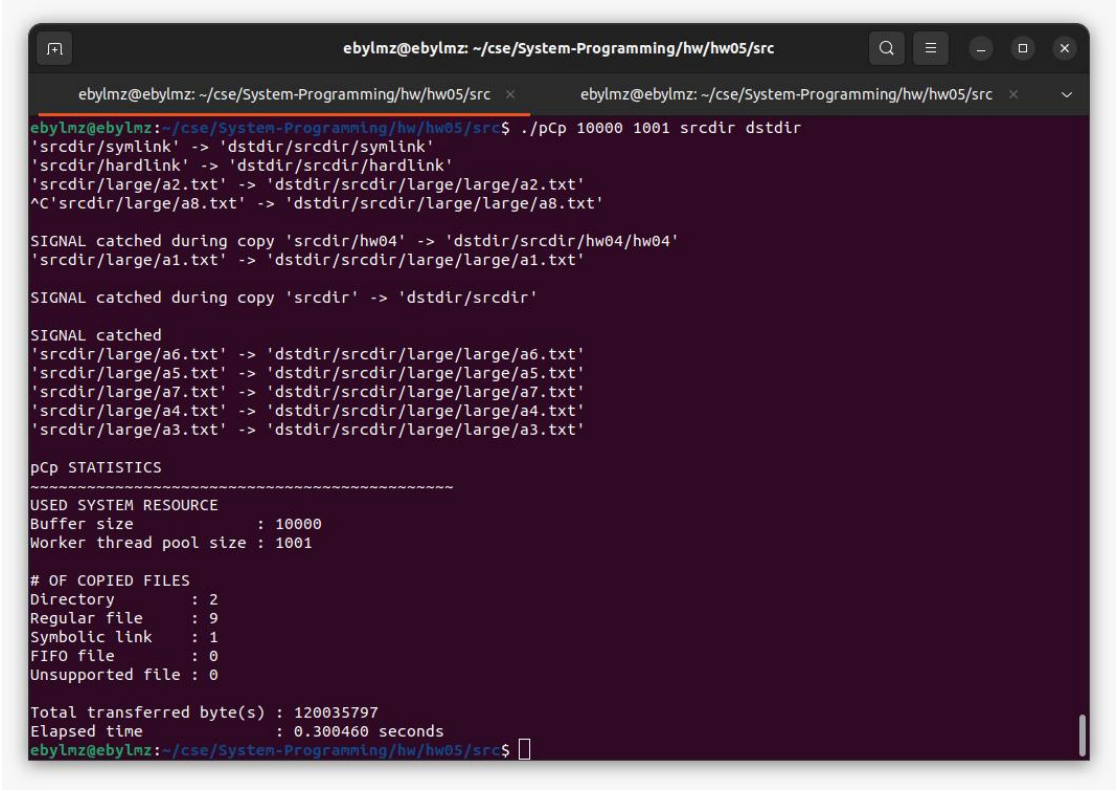


6.3 Multiple Source to Destination





6.4 Signal Handling



Signal is cached, the copy operation is stopped, and exited gracefully.

6.5 Memory Leak Check