# COEN 146L – Computer Networks

Santa Clara University | Instructor: Behnam Dezfouli

## Lab Project 1: Copying Files

- **Work individually on the project**
- **Use C language**

---

This project consists of building an application to copy a binary file to another file. We also measure file copy performance using various file sizes.

The two operational modes are as follows:

**Option 1:**
Receive the name of source file from user;
Receive the name of destination file from user;
Copy the source file to the destination file;
Write two separate functions for copying;
One function that uses function call;
Another that uses system call

**Option 2:**
Receive the source file name from user;
Receive *maximum_file_size* (in bytes) from user;
Receive *step_size* (in bytes) from user;
current_size = 0;
While ( current_size < maximum_file_size )
{
    Start time measurement;
    Create a file of current_size = current_size + step_size;
    Copy source file to destination file;
    Stop time measurement;
    Report time measurement;
}

---

**Deliverables:**

- **Demo your project to the TA in the lab**
- **Generate a graph (using Excel or Python's matplotlib) showing the impact of file size on copy performance (the x-axis shows file size and y-axis shows time)**
- **Submit your code and report to Camino**

The important file operation functions are as follows:

```
// Important C library functions used in this program:

// -- To open a file:
FILE * fopen ( const char * filename, const char * mode );
// Opens the file whose name is specified in the parameter filename and
associates it with a stream that can be identified in future operations by the
FILE pointer returned.
//

// -- To read a certain number of bytes from a file:
size_t fread ( void * ptr, size_t size, size_t count, FILE * stream );
// ptr: Pointer to a block of memory with a size of at least (size*count)
bytes, converted to a void*.
// size: Size, in bytes, of each element to be read.
// count: Number of elements, each one with a size of size bytes
// stream: Pointer to a FILE object that specifies an input stream.

// -- To write a certain number of bytes to a file:
size_t fwrite ( const void * ptr, size_t size, size_t count, FILE * stream );
// ptr: Pointer to the array of elements to be written, converted to a const
void*.
// size: Size in bytes of each element to be written.
// count: Number of elements, each one with a size of size bytes.
// stream: Pointer to a FILE object that specifies an output stream.

// The following are the equivalent system calls:

// -- To open a file:
int open ( const char * pathname, int flags );
// Opens the file whose name is specified in the parameter pathname and
associates it with a file descriptor that can be identified in future
operations by the integer that is returned.
//

// -- To read a certain number of bytes from a file:
size_t read ( int fd,void * buf, size_t size, size_t count );
// fd: Descriptor to a FILE object that specifies an input stream.
// buf: Pointer to a block of memory with a size of at least (size*count)
bytes, converted to a void*.
// count: Total number of bytes to be read


// -- To write a certain number of bytes to a file:
size_t write ( int fd,void * buf, size_t size, size_t count );
// fd: Descriptor to a FILE object that specifies an output stream.
// buf: Pointer to a block of memory with a size of at least (size*count)
bytes, converted to a void*.
// count: Total number of bytes to be written
```

The difference between **fopen** and **open** is as follows:

1) The **fopen** is a standard C library function whereas **open** is defined by POSIX.
2) The **fopen** series are high-level I/O, and the buffer is used for reading and writing. The **open** series are relatively low-level, closer to the operating system, and there is no buffer for reading and writing.

As a result the **fopen** series are a lot more portable and little bit faster as well since **open** can only be used by those systems that are following the POSIX standard.

Note that system calls are usually provided to the end-users with C library wrapper functions, hence the **fopen** function call essentially calls the **open** system call internally.

To calculate time taken by a process, we can use `clock()` function which is available *time.h*. Using this function, we can get the current value of processor clock ticks.

We can call the clock function at the beginning and end of the code for which we measure time, subtract the values, and then divide by `CLOCKS_PER_SEC` (the number of clock ticks per second) to get processor time, like following.

```
#include <time.h>

clock_t start, end;
double cpu_time_used;

start = clock();
... /* Time consuming process. */
end = clock();
cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
```