# Learn diary

## To fulfill the requirements in the assignment:

1. Redirect the standard output to a separate output process or thread.
2. This output process should continue to post-process the output of programs by, for example, writing it out to a log file.
3. Name of the log file should be able to be given in the old program.
4. The implementation should be as simple as possible to add to existing programs, and it must also support child processes spawned by the programs.
5. If the file already exists, new messages should be appended to the end.
6. The stderr stream must not be captured but is to be left untouched.

## Problems and solutions during programming:

### 1. How to redirect the standard output.

Functions like dup2 can be used to redirect between the file descriptors.

### 2. How to communicate between different processes.

Pipes are used to communicate between processes in the program.

### 3. How to make it simple to redirect the stdout to the file with given name.

I made a library to offer the function redirect (char *file_name). File name is passed by the parameter file_name to the function. Use open with O_APPEND to make sure that if the file already exists, new messages should be appended to the end.

### 4. How to make stderr stream not captured but left untouched.

Only redirect the STDOUT_FILENO to the pipe when use dup2 function. And leave STDERR_FILENO untouched.

### 5. How to use mutex in the program.

Mutex is used in the test function to synchronize the different threads created in the test program. For every second, one thread takes charge of the resource foo.

### 6. After adding the redirect function, the stdout is printed on the screen and also write in the file.

The problem is because the fork in the redirect function caused two processes, in which one process uses dup2 to redirect the stdout, and the other process prints the message to stdout. The solution is to close the STDOUT_FILENO before fork in the redirect function.

### 7. When there is a stderr in the main function, it is printed twice.

This problem is also because the fork in redirect function creates 2 processes, both of which print the error message. The solution is to close STDOUT_FILENO in the child process.

### 8. When use thread to read from pipe and write to file, there is a warning: ISO C forbids passing arg 3 of `pthread_create' between function pointer and `void *'.

This warning is because the usage of pthread_create is that if there is a parameter to pass to the thread, it should be a pointer and put into the 4th argument of pthread_create. The solution is to define a structure and the pass the pointer of the structure to the thread, the content of structure carries the parameter.

## Function test:

| Features work | Features don't work |
|---|---|
| Communication between processes or programs<br>-pipe is used between processes. | Sending the output over network sockets |
| Thread creation<br>-thread is created to write the content from pipe to a file. | |
| Use of mutex<br>-mutex is used to synchronize threads created by the test program. | |
| Thread synchronization<br>-mutex is used to synchronize threads created by the test program. | |
| Write to file<br>-write the stdout to a separate file. The file name is given by the old program. | |
| Support child processes and threads spawned by the programs. | |
| The stderr stream is not captured but is to be left untouched. | |
| If the file already exists, new messages are appended to the end. | |

## Future development:

1. In the redirect function, use two threads to post-process the captured stdout, for example, one thread write the content to a file while the other thread send the content to other programs via socket.
2. Other kind of thread synchronization like semaphore, rwlock can be implemented in the program.