====== Description ======

This is the Readme file to explain how I solve the problem,

an overview of the functions, and how to run(use) my program

====================

— means double dash.

—per-process, indicates that only the process FD table will be displayed

--systemWide, indicates that only the system-wide FD table will be displayed

--Vnodes, indicates that the Vnodes FD table will be displayed

--composite, indicates that only the composed table will be displayed

--threshold=X, where X denotes an integer, indicating that processes which have a number of FD assigned larger than X should be flagged in the output.

For this it will list the PID and number of assigned FDs, e.g. PID (FD)

- and combination of these, i.e. the program can receive multiple of these flags combined.

positional argument:
only one positional argument indicating a particular process id number (PID), if not specified the program will attempt to process all the currently running processes for the user executing the program

—output_TXT: to generate a txt file that contains the composite table.
—output_binary: to generate a bin file that contains the composite table.

For the default behavior, I choose to display the composite table.
For every command that I listed above, if the command contains the positional argument, then for every print line, I need to have the counter to indicate the number of lines.

Now, I will introduce how I solve each command.

In summary, by handout, I need to create my own data structure. Basically, I choose to use the linked list data structure since I do not need to do a lot of searching. Therefore, a linked list is a good choice for me to do. My node in my linked list will the field of pid, FD, file_name, inode, and the pointer that point to next node in my linked list
In order to achieve fully modularization, it is necessary for me to create a header file so that my code will be clearer. So I put my struct into a header file, called a2.h. Also, all function declarations are also in my header file. And importantly, I put a guard in my header file.

Firstly, in my main function, I use the struct passwd to get my current user. And then I pass it into my function of create_save_info so that I can keep checking if it is my own process in the terminal.

Secondly, since the function of create_save_info has the parameter of current use, I can loop through our file and then check.

1. —per-process: opendir to open the directory of /proc but there are a lot of directories in the /proc, filename all digit will be considered to be valid for our program and we need to make sure it is directory. and then we need to get the name of the file by using the struct dirent that is returned by readdir(). For each valid directory, I use strcat to generate a real path and then go into every fd_path by using readdir() again and get the struct. But we need to check if it is symbolic link since /proc/#/fd/# is the symbolic link by looking up in the man page. And then we can directly access the name by using the fd_entry dirent struct.
2. —system-wide: for this part, we need to add one more field to our per-process table, which is the field of pathname. Therefore, we need to use the function of redlink to get the filename since readlink() places the contents of the symbolic link pathname in the buffer buf, which has size bufsiz. readlink() does not append a terminating null byte to buf. It will (silently) truncate the contents (to a length of bufsiz characters), in case the

buffer is too small to hold all of the contents. And we know /proc/#/fd/# is a symbolic link, so for sure, we need to use readlink to get the pathname.

3. —Vnodes: for this part, we need to add the field of inode. Each file descriptor has the inode. I use the stat struct to get the inode since there is a field of inode inside that struct. These functions return information about a file, in the buffer pointed to by statbuf.

4. —composite: for this part, we just put every info inside one table here.

5. —threshold=X: for this part, we need to get the total number of FDs in one specific PID and then based on that number to identify if we need to print it out by comparing it with our threshold. If the count is still less than the threshold, we can continue the loop. If not,

6. For the command of —output_TXT, I use the fopen to get the txt file and then directly apply the sprintf and fprintf to save into the txt.

7. For the command of —output_bin, I use fopen() to open a file, loop over the linked list and used fwrite() to write the nodes in it.

Note: in order to print nicely, I separate two cases to print all the infomation.

====================== function overview ========================

- print_target: It takes an integer pid and prints the message >>> Target PID: %d\n if pid is greater than zero.
- skip_line: Prints two new lines.
- get_inode: It takes a file path as a string and returns the inode of that file.
- create_new_process: It creates a new process structure and initializes its values.
- update: It takes a process structure, pid, FD, inode, and filename and updates the structure with the new values.
- insertTail: It inserts a new process structure to the end of the linked list.
- create_save_info: It takes a pointer to the head of the linked list and a user name as a string. It iterates through the /proc directory and checks the user name of each process. If the user name matches the given name, it adds the process information to the linked list.
- print_for_composite: It takes two integers t_flag and pid. If t_flag is true, it prints the target PID message and the table headers. Otherwise, it only prints the table headers.
- save_txt: It takes a pointer to the head of the linked list and a target PID. It saves the process information in a text file named compositeTable.txt. If the target is -1, it saves all the processes. Otherwise, it only saves the information of the process with the given target PID.
- save_bin: It takes a pointer to the head of the linked list and a target PID. It saves the process information in a binary file named compositeTable.bin. If the target is -1, it saves all the processes. Otherwise, it only saves the information of the process with the given target PID.

How to run my program?

User can use make help command to get the info of makefile.

```
make

make all
```

These two command can get the executable program of my assignment 2 called a2

and then ./a2 (--composite) (threshold=#) (--per-process) (--Vnodes) (--systemWide) (output_TXT) (output_binary) (#) <- position parameter for specific pid