Homework 1

Name: Jiawei Wang

NetID: jw160

StudentID: S01435302

Part I

Task 1

My program consists of two parts. The first one is to run the program for 1000 times and then record the output in a file. And the second part is to read the output file and count how many "f1" before "f2". The program is as following.

```
#!/bin/bash
i=0
while (( $i<1000 ))
do
    ./prog >> task1.out
    i=`expr $i + 1`
done

cnt=1
f1_first_cnt=0
f2_first_cnt=0
cat task1.out | while read file
do
    if (( $cnt%2==1 ))
    then
        if [ $file = "f1" ]
        then
            f1_first_cnt=`expr $f1_first_cnt + 1`
        else
            f2_first_cnt=`expr $f2_first_cnt + 1`
        fi
    fi
    cnt=`expr $cnt + 1`
    echo "f1_first_cnt = ${f1_first_cnt}, f2_first_cnt = ${f2_first_cnt}"
done
```

Task 2

The output of the above program shows that the "f1" is ahead of "f2" every time. I have tested this program for 5 times and results always share the same pattern. The following figure is a part of the final output.

```
f1_first_cnt = 991, f2_first_cnt = 0
f1_first_cnt = 992, f2_first_cnt = 0
f1_first_cnt = 993, f2_first_cnt = 0
f1_first_cnt = 994, f2_first_cnt = 0
f1_first_cnt = 995, f2_first_cnt = 0
f1_first_cnt = 996, f2_first_cnt = 0
f1_first_cnt = 997, f2_first_cnt = 0
f1_first_cnt = 998, f2_first_cnt = 0
f1_first_cnt = 999, f2_first_cnt = 0
f1_first_cnt = 1000, f2_first_cnt = 0
```

Task 3

As far as I am concerned, it seems to be guaranteed because thousands of records show the same results according to the experiment. And the reason is the difference between system call and library call. The library call is usually faster than the system call in terms of file I/O since there is no need to do context switch and just stay at the user mode for library call. Hence, the fprintf function has less trade-off than the write function.

Part II

Task 1

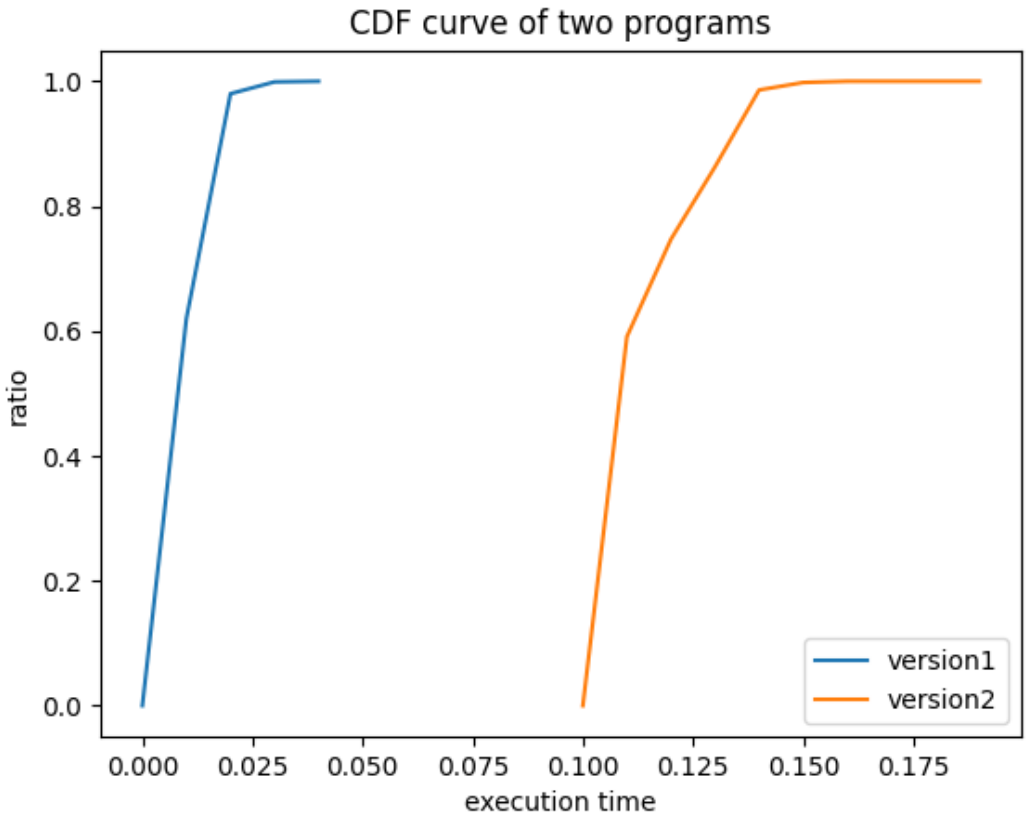
To implement this task, I write the following script to measure the execution time spent on programs each run and record the running time of each execution accordingly.

```
#!/bin/bash
echo "ver1 time"
i=0
while (( $i<1000 ))
do
    /usr/bin/time ./ver1 > /dev/null
    i=`expr $i + 1`
done

echo "ver2 time"
i=0
while (( $i<1000 ))
do
    /usr/bin/time ./ver2 > /dev/null
    i=`expr $i + 1`
done
```

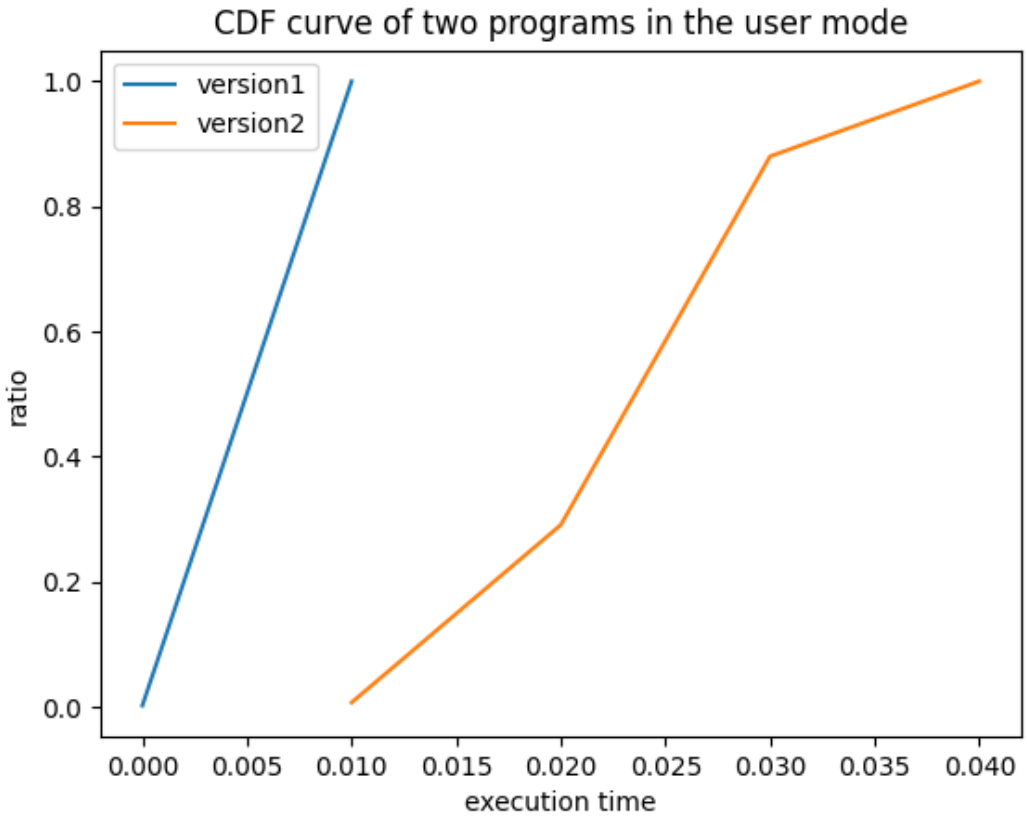
Task 2

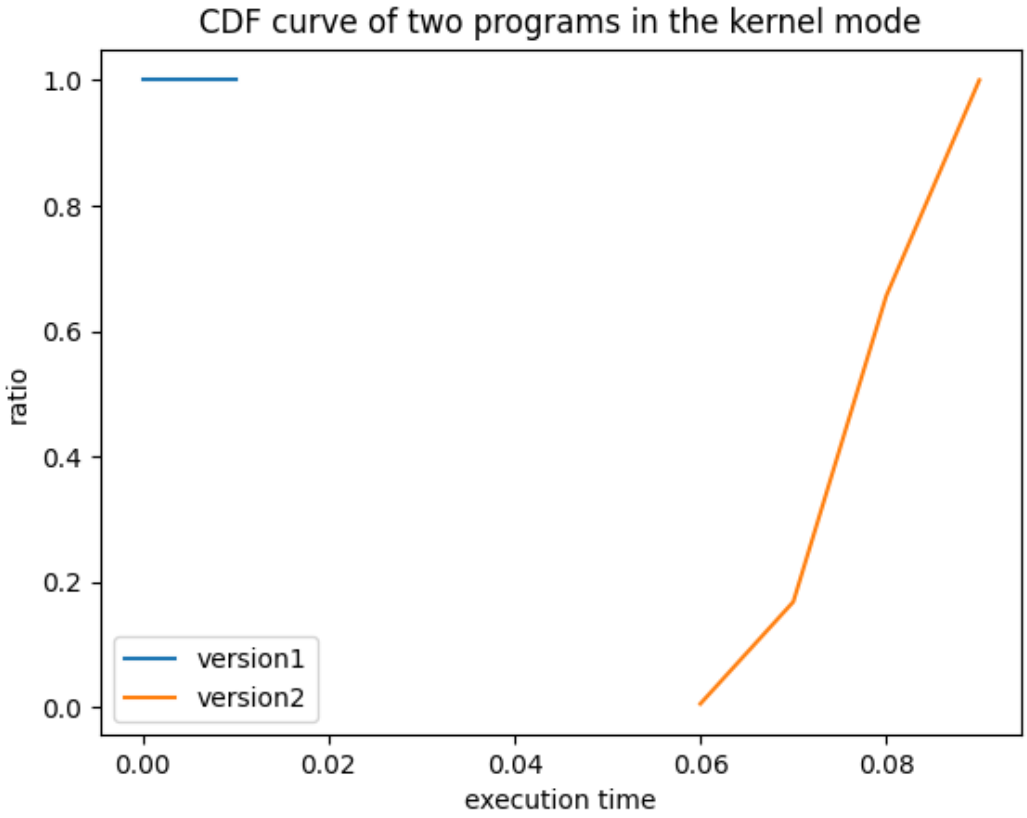
The program version1 is faster and less variable than program version2. The difference in the speed is explained in the above passage. Since the version2 needs more time to execute, the operating system may switch among different contexts during the execution more possibly. However, some threads may not suffer this kind of performance loss. Hence, the time on the program version2 is more variable.



Task 3

Program version1 almost spends no time in the kernel mode since library call doesn't need to switch mode from user to system and hardly cause context switch.


CDF curve of two programs in the user mode
version1
version2
ratio
1.0
0.8
0.6
0.4
0.2
0.0
0.000
0.005
0.010
0.015
0.020
0.025
0.030
0.035
0.040
execution time


CDF curve of two programs in the kernel mode
version1
version2
ratio
1.0
0.8
0.6
0.4
0.2
0.0
0.00
0.02
0.04
0.06
0.08
execution time

Task 4

Both of them don't have the exactly same execution time. However, the distribution of execution time of the program version2 is more dispersive. Methods for stabilize the execution time include

- use library call instead of system call
- minimize the number of processes on the machine during the execution
- avoid triggering system interrupt in the program
- minimize the number of file I/O operations in the program
- accumulate more information to send via network rather than send them respectively
- avoid triggering exception in the program