# Operating Systems
# HW Sheet # 1

Submitted by: Neeha Hammad
Date: 23-09-2019

## Problem 1.1:

a) Find the source code in file “scat.c” in folder 1a. To test the code, you can run the following command:

**./scat -l < some-large-file > testing**

where testing will be a new file with content copied successfully.

b) I used a file of 6MBs to measure times for system calls (by using -s command) and library calls (by using -l command). Here’s some of the time data I collected (in seconds).

Library Calls:

| real | user | sys |
|---|---|---|
| 0.040 | 0.035 | 0.005 |
| 0.039 | 0.039 | 0.001 |
| 0.068 | 0.063 | 0.005 |
| 0.058 | 0.058 | 0.000 |
| 0.038 | 0.031 | 0.008 |
| 0.038 | 0.034 | 0.004 |

System Calls:

| real | user | sys |
|---|---|---|
| 3.793 | 1.000 | 2.793 |
| 3.867 | 0.988 | 2.879 |
| 5.214 | 1.224 | 3.990 |

| | | |
|---|---|---|
| 4.134 | 1.059 | 3.075 |
| 4.861 | 1.104 | 3.757 |
| 4.405 | 1.204 | 3.201 |

From the data, it is quite obvious that system calls take way more time than library calls. This applies to all: real, user, and sys times.

During system calls, the system shifts from the user mode to the kernel mode to access certain resources. This is why it takes more time. In case of library calls (putc, getc) all characters are first written into a buffer before the system goes into the kernel (privileged mode) where it only spends a small fraction of time. However, in case of pure system calls, the calls read, write cause the system to shift from the user mode to the kernel mode with each separate call. A greater fraction of total time is spent in the kernel here. This makes it way more time consuming and complex.

I used the following patterned command to evaluate number of read/write calls in both variants of my program.

**strace -e trace=write ./scat -l < some-large-file > /dev/null**

In case of library calls, there's a maximum of 3 read() calls and just 1 write() call. However, for the system call option -s, there are way more system calls. We could say that there's a separate read()/write() call for each and every character. As obvious, this is more time consuming and resource consuming.

c) Find the source code in folder c with the name scat.c.

If I use the standard call i.e sendfile(1,0,NULL,1), I'll get results that are something like this:
Linux System Calls:

| real | user | sys |
|---|---|---|
| 3.491 | 0.468 | 3.023 |
| 3.622 | 0.598 | 2.997 |
| 3.468 | 0.444 | 3.024 |
| 3.490 | 0.557 | 2.934 |

From this, we can deduce that linux specific system call sendfile() is a bit faster than system calls like read/write. However, it is still more time consuming than library calls.

Now, the question says that we have to set the amount of data that is copied in each call of sendfile()such that it matches the amount of bytes read and written by the C library copy loop. This was equivalent to 4096. For this, we change the sendfile() command to: sendfile(1,0,NULL,4096) and increase the buffer value. Now, when I compile the program and run the given command a few times, I get the following results:

| real | user | sys |
|---|---|---|
| 0.016 | 0.000 | 0.010 |
| 0.007 | 0.000 | 0.007 |
| 0.006 | 0.000 | 0.007 |
| 0.006 | 0.001 | 0.005 |
| 0.006 | 0.000 | 0.006 |
| 0.006 | 0.001 | 0.006 |

From this, we can clearly see that after increasing the buffer size, this Linux specific system call takes less time than both, library calls as well as system calls.

## Problem 1.2:

Find the source code in file “watch.c”.

To check if the terminal bell rings in case of -b option, try the following command after uncommenting the line where I have written status = - 1:

./watch -b asd

where asd is any invalid command.