Final Project
4.1

4.1.1: Use strace in Linux to learn about echo command or any other command of your choice. Show a list of **system calls** being made, **total number of calls**, **time for running strace** on a particular command. (screenshots of your run should be enough for this task)

i. strace -c ls

```
@htk-leung ➜/workspaces/HW2 $ strace -c ls
flow         graph.flow  parallel_hashtable    parallel_spin.c  source                          test-3-looping-x-layers-down.c
flow-mine.c  linux.txt   parallel_hashtable.c  ph1              test                            test-4-failed-connect-bottom-child-to-process.c
flow.c       num.txt     parallel_mutex.c      pm1              test-1-fork-print.c             test-5-use-dup2-to-pipe-output.c
foo.txt      num2.txt    parallel_mutex_opt.c  ps1              test-2-pipe-two-way-communication.c  test-6-redirect-multi-level-with-2-pipes.c
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 22.51    0.000269          11        24           openat
 15.23    0.000182           4        40           mmap
 15.15    0.000181           6        26           close
 13.47    0.000161           6        25           fstat
  7.70    0.000092          23         4           write
  6.86    0.000082          10         8           mprotect
  4.02    0.000048           5         9           read
  3.01    0.000036          18         2           getdents64
  2.09    0.000025          12         2         2 statfs
  1.59    0.000019           6         3           brk
  1.51    0.000018           9         2           ioctl
  1.34    0.000016          16         1           munmap
  1.26    0.000015           7         2           rt_sigaction
  0.84    0.000010           5         2         2 access
  0.75    0.000009           9         1           rt_sigprocmask
  0.67    0.000008           8         1           futex
  0.67    0.000008           8         1           set_tid_address
  0.67    0.000008           8         1           set_robust_list
  0.67    0.000008           8         1           prlimit64
  0.00    0.000000           0         8           pread64
  0.00    0.000000           0         1           execve
  0.00    0.000000           0         2         1 arch_prctl
------ ----------- ----------- --------- --------- ----------------
100.00    0.001195                   166         5 total
```

ii. strace -c grep "include" parallel_hashtable.c

```
@htk-leung ➜/workspaces/HW2 $ strace -c grep -l "include" parallel_hashtable.c
parallel_hashtable.c
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 30.17    0.000127           4        28         6 openat
 18.76    0.000079           2        35           mmap
 12.83    0.000054           2        24           close
 12.83    0.000054           2        24           fstat
 11.64    0.000049           7         7           read
  4.04    0.000017          17         1           write
  3.33    0.000014           3         4           brk
  2.61    0.000011          11         1           stat
  1.90    0.000008           2         3           rt_sigaction
  1.90    0.000008           8         1           sigaltstack
  0.00    0.000000           0         6           mprotect
  0.00    0.000000           0         1           munmap
  0.00    0.000000           0         1           rt_sigprocmask
  0.00    0.000000           0         8           pread64
  0.00    0.000000           0         1         1 access
  0.00    0.000000           0         1           execve
  0.00    0.000000           0         2         1 arch_prctl
  0.00    0.000000           0         1           futex
  0.00    0.000000           0         1           set_tid_address
  0.00    0.000000           0         1           set_robust_list
  0.00    0.000000           0         1           prlimit64
------ ----------- ----------- --------- --------- ----------------
100.00    0.000421                   152         8 total
```

iii. strace -c cat parallel_hashtable.c > testcat.c

```
@htk-leung ➜/workspaces/HW2 $ strace -c cat parallel_hashtable.c > testcat.c
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
  0.00    0.000000           0         5           read
  0.00    0.000000           0         1           write
  0.00    0.000000           0        21           close
  0.00    0.000000           0        20           fstat
  0.00    0.000000           0        22           mmap
  0.00    0.000000           0         3           mprotect
  0.00    0.000000           0         2           munmap
  0.00    0.000000           0         3           brk
  0.00    0.000000           0         6           pread64
  0.00    0.000000           0         1         1 access
  0.00    0.000000           0         1           execve
  0.00    0.000000           0         2         1 arch_prctl
  0.00    0.000000           0         1           fadvise64
  0.00    0.000000           0        19           openat
------ ----------- ----------- --------- --------- ----------------
100.00    0.000000                   107         2 total
```

4.1.2: Pick 4 random system calls (ex: nmap, write, open, etc) of your choice and explain their functionality for the command that you run (ex: echo, ls, cd, etc) in 2-3 sentences

fstat - It is basically the "stat" function but specifically used for files. It is a system call that retrieves information about an open file by using its file descriptor, providing details like file size, access time, modification time, ownership, and file type, which are essentiallythe "status" of the file.

openat - It opens a file the same way open() does, by using the specified directory file descriptor as the starting location for the path search, except that if pathname is absolute, file descriptor returned from dirfd() is ignored.

read - It is a system call used to read data from an open file into system buffer.

access - It is a system call used to check file permissions, including whether a file exists, and if it can be read, written, or executed by the calling process, and can be used for a file or directory.

4.2

4.2.1: Implement **"strace on"** [10 points]
After typing strace on into the terminal, the straceon() system call is used to set the trace_off global variable located in proc.c to 1. When further commands are given to the terminal, the shell will check this variable with checkstrace, and set the process variable strace to 1 in the new process. Processes with the variable strace set to 1 will print out system call logs.

```
$ echo hello
hello
$ strace on
$ echo hello
hTRACE: pid = 9 | command_name = echo | syscall = write | return value = 1
eTRACE: pid = 9 | command_name = echo | syscall = write | return value = 1
lTRACE: pid = 9 | command_name = echo | syscall = write | return value = 1
lTRACE: pid = 9 | command_name = echo | syscall = write | return value = 1
oTRACE: pid = 9 | command_name = echo | syscall = write | return value = 1

TRACE: pid = 9 | command_name = echo | syscall = write | return value = 1
TRACE: pid = 9 | command_name = echo | syscall = exit
```

4.2.2: Implement **"strace off"** [10 points]
After typing strace off into the terminal, the straceoff() system call is used to set trace_off to 0. Consequently, future terminal commands won't have the strace variables in their processes set to 1 and won't print out system call logs.

```
$ echo hello
hello
$ strace on
$ echo hello
hTRACE: pid = 9 | command_name = echo | syscall = write | return value = 1
eTRACE: pid = 9 | command_name = echo | syscall = write | return value = 1
lTRACE: pid = 9 | command_name = echo | syscall = write | return value = 1
lTRACE: pid = 9 | command_name = echo | syscall = write | return value = 1
oTRACE: pid = 9 | command_name = echo | syscall = write | return value = 1

TRACE: pid = 9 | command_name = echo | syscall = write | return value = 1
TRACE: pid = 9 | command_name = echo | syscall = exit
$ strace off
$ echo hello
hello
```

4.2.3: Implementing **"strace run <command>"** [10 points]
For strace run, we call fork in strace.c, use the set_proc_strace() to set the child's strace variable, and then use exec to run the child.

```
$ strace run echo hello
hTRACE: pid = 13 | command_name = echo | syscall = write | return value = 1
eTRACE: pid = 13 | command_name = echo | syscall = write | return value = 1
lTRACE: pid = 13 | command_name = echo | syscall = write | return value = 1
lTRACE: pid = 13 | command_name = echo | syscall = write | return value = 1
oTRACE: pid = 13 | command_name = echo | syscall = write | return value = 1

TRACE: pid = 13 | command_name = echo | syscall = write | return value = 1
TRACE: pid = 13 | command_name = echo | syscall = exit
```

4.2.4: Implementing **strace dump**[10 points]

In syscall.c, we created the global variable char system_call_log[X][MAX_TRACE_ENTRY_SIZE]. X, used instead of N, and MAX_TRACE_ENTRY_SIZE are defined in param.h. When system calls unrelated to strace itself and the shell are made, they are logged in sys_call_log. Through the use of the global variable sys_call_index and modular arithmetic, we are able to use sys_call_log as a circular buffer. This ensures the most recent 10 system calls are always printed in order.

Note: This is the main reason why system calls related to strace and the shell are not printed or logged. If we did log them, most of the last 10 system calls would always be related to the shell or strace itself. Per Aniket, this is not the desired output.

```
$ strace dump
trace log 0 = TRACE: pid = 2 | command_name = sh | syscall = read | return value = 1
trace log 1 = TRACE: pid = 2 | command_name = sh | syscall = read | return value = 1
trace log 2 = TRACE: pid = 2 | command_name = sh | syscall = read | return value = 1
trace log 3 = TRACE: pid = 2 | command_name = sh | syscall = read | return value = 1
trace log 4 = TRACE: pid = 2 | command_name = sh | syscall = strace_selstatus | return value = 0
trace log 5 = TRACE: pid = 2 | command_name = sh | syscall = fork | return value = 5
trace log 6 = TRACE: pid = 5 | command_name = sh | syscall = check_strace | return value = 0
trace log 7 = TRACE: pid = 5 | command_name = sh | syscall = sbrk | return value = 16384
trace log 8 = TRACE: pid = 5 | command_name = sh | syscall = exec
trace log 9 = TRACE: pid = 5 | command_name = strace | syscall = exec | return value = 0
```

Below is the output we get with our current implementation.

```
$ strace off
$ echo howdy there partner
howdy there partner
$ strace dump
trace log 0 = TRACE: pid = 33 | command_name = echo | syscall = write | return value = 1
trace log 1 = TRACE: pid = 33 | command_name = echo | syscall = write | return value = 1
trace log 2 = TRACE: pid = 33 | command_name = echo | syscall = write | return value = 1
trace log 3 = TRACE: pid = 33 | command_name = echo | syscall = write | return value = 1
trace log 4 = TRACE: pid = 33 | command_name = echo | syscall = write | return value = 1
trace log 5 = TRACE: pid = 33 | command_name = echo | syscall = write | return value = 1
trace log 6 = TRACE: pid = 33 | command_name = echo | syscall = write | return value = 1
trace log 7 = TRACE: pid = 33 | command_name = echo | syscall = write | return value = 1
trace log 8 = TRACE: pid = 33 | command_name = echo | syscall = write | return value = 1
trace log 9 = TRACE: pid = 33 | command_name = echo | syscall = exit
```

4.2.5: Trace **child process**[10 points]

For this portion, all we did was modify the fork() function in proc.c so that a child inherits the strace value of its parent. Consequently, when a program is run with strace, all of its children will also print out system call lines. Please see the output below.

```
$ strace on
$ trace_children
TRACE: pid = 18 | command_name = trace_children | syscall = fork | return value = 19
CTRACE: pid = 19 | command_name = trace_children | syscall = write | return value = 1
hTRACE: pid = 19 | command_name = trace_children | syscall = write | return value = 1
TRACE: pid = 18 | command_name = trace_children | syscall = fork | return value = 20
iTRACE: pid = 19 | command_name = trace_children | syscall = write | return value = 1
CTRACE: pid = 20 | command_name = trace_children | syscall = write | return value = 1
hTRACE: pid = 20 | command_name = trace_children | syscall = write | return value = 1
lTRACE: pid = 19 | command_name = trace_children | syscall = write | return value = 1
iTRACE: pid = 20 | command_name = trace_children | syscall = write | return value = 1
dTRACE: pid = 19 | command_name = trace_children | syscall = write | return value = 1
lTRACE: pid = 20 | command_name = trace_children | syscall = write | return value = 1
 TRACE: pid = 19 | command_name = trace_children | syscall = write | return value = 1
TRACE: pid = 18 | command_name = trace_children | syscall = fork | return value = 21
0TRACE: pid = 19 | command_name = trace_children | syscall = write | return value = 1
dTRACE: pid = 20 | command_name = trace_children | syscall = write | return value = 1
 TRACE: pid = 20 | command_name = trace_children | syscall = write | return value = 1
CTRACE: pid = 21 | command_name = trace_children | syscall = write | return value = 1
1TRACE: pid = 18 | command_name = trace_children | syscall = fork | return value = 22
CTRACE: pid = 22 | command_name = trace_children | syscall = write | return value = 1

TRACE: pid = 19 | command_name = trace_children | syscall = write | return value = 1
TRACE: pid = 19 | command_name = trace_children | syscall = exit
TRACE: pid = 20 | command_name = trace_children | syscall = write | return value = 1
hTRACE: pid = 21 | command_name = trace_children | syscall = write | return value = 1

hTRACE: pid = 22 | command_name = trace_children | syscall = write | return value = 1
iTRACE: pid = 21 | command_name = trace_children | syscall = write | return value = 1
lTRACE: pid = 21 | command_name = trace_children | syscall = write | return value = 1
TRACE: pid = 18 | command_name = trace_children | syscall = fork | return value = 23
iTRACE: pid = 22 | command_name = trace_children | syscall = write | return value = 1
TRACE: pid = 20 | command_name = trace_children | syscall = write | return value = 1
```

```
ldTRACE: pid = 21 | command_name = trace_children | syscall = write | return value = 1
TRACE: pid = 22 | command_name = trace_children | syscall = write | return value = 1
dTRACE: pid = 22 | command_name = trace_children | syscall = write | return value = 1
 TRACE: pid = 22 | command_name = trace_children | syscall = write | return value = 1
CTRACE: pid = 23 | command_name = trace_children | syscall = write | return value = 1
TRACE: pid = 20 | command_name = trace_children | syscall = exit
 TRACE: pid = 21 | command_name = trace_children | syscall = write | return value = 1
TRACE: pid = 18 | command_name = trace_children | syscall = fork | return value = 24
3hTRACE: pid = 23 | command_name = trace_children | syscall = write | return value = 1
CTRACE: pid = 24 | command_name = trace_children | syscall = write | return value = 1
2TRACE: pid = 22 | command_name = trace_children | syscall = write | return value = 1
TRACE: pid = 21 | command_name = trace_children | syscall = write | return value = 1

TRACE: pid = 21 | command_name = trace_children | syscall = write | return value = 1
ihTRACE: pid = 24 | command_name = trace_children | syscall = write | return value = 1

iTRACE: pid = 22 | command_name = trace_children | syscall = write | return value = 1
TRACE: pid = 23 | command_name = trace_children | syscall = write | return value = 1
TRACE: pid = 24 | command_name = trace_children | syscall = write | return value = 1
lTRACE: pid = 23 | command_name = trace_children | syscall = write | return value = 1
TRACE: pid = 18 | command_name = trace_children | syscall = fork | return value = 25
lTRACE: pid = 21 | command_name = trace_children | syscall = exit
TRACE: pid = 22 | command_name = trace_children | syscall = exit
TRACE: pid = 24 | command_name = trace_children | syscall = write | return value = 1
dTRACE: pid = 18 | command_name = trace_children | syscall = wait | return value = 19
CTRACE: pid = 25 | command_name = trace_children | syscall = write | return value = 1
TRACE: pid = 18 | command_name = trace_children | syscall = wait | return value = 20
TRACE: pid = 23 | command_name = trace_children | syscall = write | return value = 1
TRACE: pid = 18 | command_name = trace_children | syscall = wait | return value = 21
dTRACE: pid = 24 | command_name = trace_children | syscall = write | return value = 1
  TRACE: pid = 24 | command_name = trace_children | syscall = write | return value = 1
5hTRACE: pid = 24 | command_name = trace_children | syscall = write | return value = 1
TRACE: pid = 25 | command_name = trace_children | syscall = write | return value = 1
TRACE: pid = 18 | command_name = trace_children | syscall = wait | return value = 22
```

```
TRACE: pid = 23 | command_name = trace_children | syscall = write | return value = 1

4TRACE: pid = 24 | command_name = trace_children | syscall = write | return value = 1
iTRACE: pid = 25 | command_name = trace_children | syscall = write | return value = 1
TRACE: pid = 24 | command_name = trace_children | syscall = exit
lTRACE: pid = 18 | command_name = trace_children | syscall = wait | return value = 24
TRACE: pid = 23 | command_name = trace_children | syscall = write | return value = 1
TRACE: pid = 25 | command_name = trace_children | syscall = write | return value = 1
dTRACE: pid = 25 | command_name = trace_children | syscall = write | return value = 1
 TRACE: pid = 25 | command_name = trace_children | syscall = write | return value = 1
6TRACE: pid = 25 | command_name = trace_children | syscall = write | return value = 1

TRACE: pid = 23 | command_name = trace_children | syscall = write | return value = 1

TRACE: pid = 23 | command_name = trace_children | syscall = exit
TRACE: pid = 25 | command_name = trace_children | syscall = write | return value = 1
TRACE: pid = 18 | command_name = trace_children | syscall = wait | return value = 23
TRACE: pid = 25 | command_name = trace_children | syscall = exit
TRACE: pid = 18 | command_name = trace_children | syscall = wait | return value = 25
TRACE: pid = 18 | command_name = trace_children | syscall = exit
```

4.3

4.3.1: Option: **-e <system call name>** [5 points]

When option flag -e is provided followed by a system call name (ex: write), we will print only that system call. If no such system call is made in the command, print nothing.

```
$ strace -e pipe
$ ls | wc
TRACE: pid = 15 | command_name = sh | syscall = pipe | return value = 0
23 92 575
$ ls | wc
23 92 575
$ strace -e pipe
$ echo hello
hello
$
```

4.3.2: Option: -s [5 points]

When option flag -s is provided, print only successful system calls.

```
$ strace -s
$ echo hello
hTRACE: pid = 7 | command_name = echo | syscall = write | return value = 1
eTRACE: pid = 7 | command_name = echo | syscall = write | return value = 1
lTRACE: pid = 7 | command_name = echo | syscall = write | return value = 1
lTRACE: pid = 7 | command_name = echo | syscall = write | return value = 1
oTRACE: pid = 7 | command_name = echo | syscall = write | return value = 1
```

4.3.3: Option: -f [5 points]

When option flag -f is provided, print only failed system calls.

4.3.4: Extra credits: Combine options [5 points]

Implement these 2 commands:

“strace -s -e <system call name>”: print only successful system call name.

“strace -f -e <system call name>”: print only failed system call name.

4.5: Application of strace [15 points]

Strace in Linux has additional information about system call input argument values, which helps to see which system call failed upon which input, and makes it easy to match the line with the corresponding code.In both printouts, immediately after the clone, close() executes, then wait(). Which hints at a similar execution sequence in Linux and xv6. However, while the file descriptor in xv6 stays intact, the one in Linux becomes corrupted. This hints at differences in file descriptor handling methods when cloning and protection mechanisms. This helps to navigate the system while debugging. Rather than trying to find syntax errors, based on this information, we can see that the problem is deeper in the system.

Behavior in xv6 :

```
$ test 10
num = 0
num = 1
num = 1
num = 1
num = 1
num = 1
num = 1
num = 1
num = 1
num = 1
num = 1
$
```

Strace in xv6 with `strace run test 10`

```
TRACE: pid = 4 | command_name = test | syscall = write | return value = 1
TRACE: pid = 4 | command_name = test | syscall = fork | return value = 13
TRACE: pid = 4 | command_name = test | syscall = close | return value = 0
TRACE: pid = 4 | command_name = test | syscall = wait | return value = 13
TRACE: pid = 4 | command_name = test | syscall = open | return value = 3
TRACE: pid = 4 | command_name = test | syscall = read | return value = 20
nTRACE: pid = 4 | command_name = test | syscall = write | return value = 1
uTRACE: pid = 4 | command_name = test | syscall = write | return value = 1
mTRACE: pid = 4 | command_name = test | syscall = write | return value = 1
 TRACE: pid = 4 | command_name = test | syscall = write | return value = 1
=TRACE: pid = 4 | command_name = test | syscall = write | return value = 1
 TRACE: pid = 4 | command_name = test | syscall = write | return value = 1
1TRACE: pid = 4 | command_name = test | syscall = write | return value = 1

TRACE: pid = 4 | command_name = test | syscall = write | return value = 1
TRACE: pid = 4 | command_name = test | syscall = fork | return value = 14
TRACE: pid = 4 | command_name = test | syscall = close | return value = 0
TRACE: pid = 4 | command_name = test | syscall = wait | return value = 14
TRACE: pid = 4 | command_name = test | syscall = exit
$
```

Actual behavior in Linux :

```
@htk-leung →/workspaces/HW2 $ ./apptest 10
num = 0
num = 1
read error
```

Strace in Linux with `strace ./apptest 10 | grep -e write -e read -e open -e close`

```
@htk-leung →/workspaces/HW2 $ strace ./apptest 10 | grep -e write -e read -e open -e close
execve("./apptest", ["./apptest", "10"], 0x7fff8209b318 /* 102 vars */) = 0
brk(NULL)                               = 0x5c3167b09000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffc036dd2c0) = -1 EINVAL (Invalid argument)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=44733, ...}) = 0
mmap(NULL, 44733, PROT_READ, MAP_PRIVATE, 3, 0) = 0x708e7acf7000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\300A\2\0\0\0\0"..., 832) = 832
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0", 32, 848) = 32
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\7\2C\n\357_\243\335\2449\206V>\237\374\304"..., 68, 880) = 68
fstat(3, {st_mode=S_IFREG|0755, st_size=2029592, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x708e7acf5000
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0", 32, 848) = 32
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\7\2C\n\357_\243\335\2449\206V>\237\374\304"..., 68, 880) = 68
mmap(NULL, 2037344, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x708e7ab03000
mmap(0x708e7ab25000, 1540096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x22000) = 0x708e7ab25000
mmap(0x708e7ac9d000, 319488, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x19a000) = 0x708e7ac9d000
mmap(0x708e7aceb000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7000) = 0x708e7aceb000
mmap(0x708e7acf1000, 13920, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x708e7acf1000
close(3)                                = 0
arch_prctl(ARCH_SET_FS, 0x708e7acf6540) = 0
mprotect(0x708e7aceb000, 16384, PROT_READ) = 0
mprotect(0x5c31668f6000, 4096, PROT_READ) = 0
mprotect(0x708e7ad2f000, 4096, PROT_READ) = 0
munmap(0x708e7acf7000, 44733)           = 0
```

```
openat(AT_FDCWD, "data.txt", O_RDWR|O_CREAT|O_TRUNC, 0644) = 3
fstat(1, {st_mode=S_IFIFO|0600, st_size=0, ...}) = 0
brk(NULL)                               = 0x5c3167b09000
brk(0x5c3167b2a000)                     = 0x5c3167b2a000
write(3, "0\n", 2)                      = 2
close(3)                                = 0
openat(AT_FDCWD, "data.txt", O_RDWR)    = 3
read(3, "0\n", 512)                     = 2
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x708e7acf6810) = 16482
close(3)                                = 0
wait4(-1, NULL, 0, NULL)                = 16482
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=16482, si_uid=1000, si_status=1, si_utime=0, si_stime=0} ---
read(3, 0x5c31668f7040, 512)            = -1 EBADF (Bad file descriptor)
write(1, "num = 0\nnum = 1\nread error\n", 27read error
) = 27
exit_group(1)                           = ?
+++ exited with 1 +++
@htk-leung →/workspaces/HW2 $
```