DECS Assignment 1

- Shivansh Dhiman, 22m2120

Question 1

In this question, we will understand the hardware configuration of your working machine using the /proc filesystem.

(a) Hyperthreading

> - Hyperthreading allows the system to use more number of cores than the socket actually has. A socket with Hypertheading enabled exposes two (or more) logical cores to the system for each single physical core. It allows system to execute two (or more) concurrent threads on each core, thus improving performance.
> - Cores are physical processing units inside a socket.
> - Processors (or CPUs) are logical processing units that are exposed to the system.
> - In my machine, there are two physical cores. Hyperthreading is enabled, so, the number of logical cores exposed to the system are four in number.

Command used: `more /proc/cpuinfo`

```
 sd@ip  ~  more /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 78
model name      : Intel(R) Core(TM) i5-6200U CPU @ 2.30GHz
stepping        : 3
microcode       : 0xf0
cpu MHz         : 1399.999
cache size      : 3072 KB
physical id     : 0
siblings        : 4
core id         : 0
cpu cores       : 2
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 22
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr
sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc art arch_perfmon pebs bts rep_good nopl xtopology no
nstop_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1
 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch cpuid_fault invpc
id_single pti ssbd ibrs ibpb stibp tpr_shadow vnmi flexpriority ept vpid ept_ad fsgsbase tsc_adjust bmi1 avx2 smep b
mi2 erms invpcid mpx rdseed adx smap clflushopt intel_pt xsaveopt xsavec xgetbv1 xsaves dtherm ida arat pln pts hwp
hwp_notify hwp_act_window hwp_epp md_clear flush_l1d arch_capabilities
vmx flags       : vnmi preemption_timer invvpid ept_x_only ept_ad ept_1gb flexpriority tsc_offset vtpr mtf vapic ept
 vpid unrestricted_guest ple pml
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf mds swapgs itlb_multihit srbds mmio_stal
e_data
bogomips        : 4800.00
clflush size    : 64
cache_alignment : 64
address sizes   : 39 bits physical, 48 bits virtual
power management:
```

Command used: `lscpu`

```
sd@ip ~ lscpu
Architecture:            x86_64
  CPU op-mode(s):        32-bit, 64-bit
  Address sizes:         39 bits physical, 48 bits virtual
  Byte Order:            Little Endian
CPU(s):                  4
  On-line CPU(s) list:   0-3
Vendor ID:               GenuineIntel
  Model name:            Intel(R) Core(TM) i5-6200U CPU @ 2.30GHz
    CPU family:          6
    Model:               78
    Thread(s) per core:  2
    Core(s) per socket:  2
    Socket(s):           1
    Stepping:            3
    CPU max MHz:         2800.0000
    CPU min MHz:         400.0000
    BogoMIPS:            4800.00
    Flags:               fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mm
                         x fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc art arch_perfmon peb
                         s bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor ds_
                         cpl vmx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_de
                         adline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch cpuid_fault invpcid_single
                          pti ssbd ibrs ibpb stibp tpr_shadow vnmi flexpriority ept vpid ept_ad fsgsbase tsc_adjust
                         bmi1 avx2 smep bmi2 erms invpcid mpx rdseed adx smap clflushopt intel_pt xsaveopt xsavec xg
                         etbv1 xsaves dtherm ida arat pln pts hwp hwp_notify hwp_act_window hwp_epp md_clear flush_l
                         1d arch_capabilities
Virtualization features:
  Virtualization:        VT-x
Caches (sum of all):
  L1d:                   64 KiB (2 instances)
  L1i:                   64 KiB (2 instances)
  L2:                    512 KiB (2 instances)
  L3:                    3 MiB (1 instance)
NUMA:
  NUMA node(s):          1
  NUMA node0 CPU(s):     0-3
Vulnerabilities:
  Itlb multihit:         KVM: Mitigation: VMX disabled
  L1tf:                  Mitigation; PTE Inversion; VMX conditional cache flushes, SMT vulnerable
  Mds:                   Mitigation; Clear CPU buffers; SMT vulnerable
  Meltdown:              Mitigation; PTI
  Mmio stale data:       Mitigation; Clear CPU buffers; SMT vulnerable
  Spec store bypass:     Mitigation; Speculative Store Bypass disabled via prctl
  Spectre v1:            Mitigation; usercopy/swapgs barriers and __user pointer sanitization
  Spectre v2:            Mitigation; Retpolines, IBPB conditional, IBRS_FW, STIBP conditional, RSB filling
  Srbds:                 Mitigation; Microcode
  Tsx async abort:       Not affected
sd@ip ~
```

(b) Number of `cores`: My laptop has 2 physical cores.

Command: `lscpu`

```
    Core(s) per socket:  2
```

(c) Number of `processors`: My laptop has 4 logical processors.

Command: `lscpu`

```
CPU(s):                  4
  On-line CPU(s) list:   0-3
```

(d) Frequency of each core:

Command: `more /proc/cpuinfo`

Processor 0:
```
cpu MHz         : 1342.874
```

Processor 1:
```
cpu MHz         : 1222.149
```

Processor 2:
```
cpu MHz         : 1333.337
```

Processor 3:
```
cpu MHz         : 2400.000
```

(e) Architecture of my CPU: x86_64

Command: `lscpu`

```
Architecture:            x86_64
```

(f) Physical memory of my system:

Command: `cat /proc/meminfo`

```
MemTotal:        7997696 kB
MemFree:         1240536 kB
MemAvailable:    3079392 kB
```

(g) Free memory:

Command: `free`

```
               total        used        free      shared  buff/cache   available
Mem:         7997696     3709816     1211392      918844     3076488     3051384
Swap:       12287480     1661088    10626392
```

(h) Total number of forks and context switches since last boot:

Command for no. of forks: `vmstat -f`

```
       383743 forks
```

Command for no. of context switches: `cat /proc/stat`

```
ctxt 777416199
```

Question 2

In this question, we will understand how to monitor the status of a running process using the top command. Compile the program cpu.c given to you and execute it in the bash or any other shell of your choice as follows.

Command: `top`

```
top - 12:06:39 up 4 days, 21:37,  1 user,  load average: 2.86, 2.11, 1.89
Tasks: 315 total,   2 running, 313 sleeping,   0 stopped,   0 zombie
%Cpu(s): 14.7 us,  5.4 sy, 42.4 ni, 37.3 id,  0.0 wa,  0.0 hi,  0.1 si,  0.0 st
MiB Mem :   7810.2 total,   1179.4 free,   3535.0 used,   3095.8 buff/cache
MiB Swap:  11999.5 total,  10377.6 free,   1621.9 used.   3019.8 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 384371 sd        15  -5    2640    952    864 R  99.3   0.0   1:13.32 cpu
```

(a) PID of the process: `384371`

(b) CPU consumed by the process: 99.3%

Command: `top`

```
%CPU
99.3
```

Memory consumed by the process: 2640kB

Command: `cat /proc/384371/status`

```
VmSize:     2640 kB
```

(c) State of the process: Running

Command: `cat /proc/384371/status`

```
State:  R (running)
```

Question 3

In this question, we will understand how the Linux shell (e.g., the bash shell) runs user commands by spawning new child processes to execute the various commands.

(a) PID of the process: 386348

Command used: `ps -ef | grep cpu-print`

```
sd        386348  386274 51 12:29 pts/1    00:01:51 ./cpu-print
```

(b) Parent PID of the process: 386274

Command used: `ps -ef | grep cpu-print`

```
sd        386348  386274 51 12:29 pts/1    00:01:51 ./cpu-print
```

Ancestors' PID of the process: 1, 2063, 381846, 386274

Command used: `pstree -ps 386348`

```
systemd(1)---systemd(2063)---gnome-terminal-(381846)---zsh(386274)---cpu-print(386348)
```

(c) I/O Redirection

> I/O Redirection into a file is done by using angular brackets `>` and `<`. When `stdin` stream (numbered as 0) is pointed to a file, then the bash shell receives its input from that file. Here, in case of `stdout` (numbered as 1), the program `cpu-print` sends its output to the specified file.

```
 sd@ip  ~/Downloads/intro-code  ./cpu-print > /tmp/tmp.txt &
[1] 87240
```

Command used: `ls -la /proc/87240/fd`

```
 sd@ip  ~/Downloads/intro-code  ls -la /proc/87240/fd
total 0
dr-x------ 2 sd sd  0 Aug 17 11:56 .
dr-xr-xr-x 9 sd sd  0 Aug 17 11:56 ..
lrwx------ 1 sd sd 64 Aug 17 11:56 0 -> /dev/pts/0
l-wx------ 1 sd sd 64 Aug 17 11:56 1 -> /tmp/tmp.txt
lrwx------ 1 sd sd 64 Aug 17 11:56 2 -> /dev/pts/0
```

(d) Pipes

> Pipes are handled by a virtual filesystem called `pipefs` inside kernel. Here, when the shell encountered the symbol `|`, it invoked a system call `pipe()`. The kernel created a pipe with an ID `3389479`, and takes the output (stdout) from `cpu-print` process and feeds it as an input (stdin) to the process `grep`.
>
> `stdin` of `cpu-print`, `stdout` of `grep` and `stderr` of both processes are pointing to pseudo-terminals.

```
 sd@ip  ~/Downloads/intro-code  ./cpu-print | grep hello &
[1] 390026 390027
```

Commands used:

`ls -la /proc/390026/fd`

`ls -la /proc/390027/fd`

```
 sd@ip  ~  ls -la /proc/390026/fd
total 0
dr-x------ 2 sd sd  0 Aug 13 13:10 .
dr-xr-xr-x 9 sd sd  0 Aug 13 13:09 ..
lrwx------ 1 sd sd 64 Aug 13 13:10 0 -> /dev/pts/1
l-wx------ 1 sd sd 64 Aug 13 13:10 1 -> 'pipe:[3389679]'
lrwx------ 1 sd sd 64 Aug 13 13:10 2 -> /dev/pts/1
 sd@ip  ~  ls -la /proc/390027/fd
total 0
dr-x------ 2 sd sd  0 Aug 13 13:10 .
dr-xr-xr-x 9 sd sd  0 Aug 13 13:09 ..
lr-x------ 1 sd sd 64 Aug 13 13:10 0 -> 'pipe:[3389679]'
lrwx------ 1 sd sd 64 Aug 13 13:10 1 -> /dev/pts/1
lrwx------ 1 sd sd 64 Aug 13 13:10 2 -> /dev/pts/1
```

(e) Types of commands

- `cd` & `history` are shell built-in commands. The executables of these commands do not exist and shell implements these commands itself within its code.

- `ls` & `ps` is an executable in Linux kernel directory tree. Since these are not implemented inside the shell, the shell simply finds and invokes the executable.

Commands used:

`type cd`

`type ls`

`type history`

`type ps`

```
 sd@ip  ~  type cd
cd is a shell builtin
 sd@ip  ~  type ls
ls is an alias for ls --color=tty
 sd@ip  ~  type history
history is an alias for omz_history
 sd@ip  ~  type omz_history
omz_history is a shell function from /home/sd/.oh-my-zsh/lib/history.zsh
 sd@ip  ~  type ps
ps is /usr/bin/ps
```

Question 4

Consider the two programs memory1.c and memory2.c given to you. Compile and run them one after the other. Both programs allocate a large array in memory. One of them accesses the array and the other doesn't. Both programs pause before exiting to let you inspect their memory usage. You can inspect the memory used by a process with the ps command. In particular, the output will tell you what the total size of the "virtual" memory of the process is, and how much of this is actually physically resident in memory. You will learn later that the virtual memory of the process is the memory the process thinks it has, while the OS only allocates a subset of this memory physically in RAM.
Compare the virtual and physical memory usage of both programs, and explain your observations. You can also inspect the code to understand your observations.

> Virtual Memory is total memory the process thinks it has. It is not necessary that all of this memory is present in the RAM. OS usually swaps out the inactive pages out of the RAM to make more space for other programs. Here, the `memory1` program is just initializing a large array, whereas, `memory2` is initializing and looping over a large array. This causes the OS to store more pages in the RAM. Thats why, `memory2` has a larger RSS (Resident Set Size), i.e., size of the process actually in RAM.

Command used: `ps -aux --sort -vsz | grep memory`

```
sd        435217  0.0  0.0   6560  4824 pts/1    S<+  11:11   0:00 ./memory1
sd        435305  0.2  0.0   6560  5404 pts/2    S<+  11:12   0:00 ./memory2
```

Question 5

In this question, you will compile and run the programs disk.c and disk1.c given to you. These programs read a large number of files from disk, and you must first create these files as follows. Create a folder disk-files and place the file foo.pdf in that folder. Then use the script make-copies.sh to make 5000 copies of the same file in that folder, with different filenames. The disk programs will read these files. Now, run the disk programs one after the other. For each program, measure the utilization of the disk while the program is running. Report and explain your observations. You will find a tool like iostat useful for measuring disk utilization. Also read through the code of the programs to help explain your observations.
Note that for this exercise to work correctly, you must be reading from a directory on the local disk. If your disk-files directory is not on a local disk (but, say, mounted via NFS), then you must alter the location of the files in the code provided to you to enable reading from a local disk. Also, modern operating systems store recently read files in a cache in memory (called disk buffer cache) for faster access to the same files in the future. In order to ensure that you are making observations while actually reading from disk, you must clear your disk buffer cache between multiple runs of disk.c. If you do not clear the disk buffer cache between successive runs of disk.c, you will be reading the files not from disk but from

memory. Look up online for commands on how to clear your disk buffer cache, and note that you will need superuser permissions to execute these commands.

> According to the observations, `disk.c` read the disk at a rate of about 200 MB/s. Whereas, `disk1.c` doesn't use the disk at all, except just once.
> This is due to the fact that `disk.c` randomly chooses a file from the copies of `foo.pdf`, reads it and overwrites it with zeroes in memory.
> On the other hand, `disk1.c` only reads one file and repeteadly overwrites it with zeroes in memory. Due to this, the OS caches the file in memory and uses it for this process. Hence, there is almost no disk activity in this case.

Command for clearing disk buffer cache: `free && sync && echo 3 > /proc/sys/vm/drop_caches && free`

```
 sd@ip  ~/Downloads/intro-code  sudo su
[sudo] password for sd:
root@ip:/home/sd/Downloads/intro-code# free && sync && echo 3 > /proc/sys/vm/drop_caches && free
               total        used        free      shared  buff/cache   available
Mem:         7997696     1547796     5637528      349292      812372     5728540
Swap:       12287480     2067664    10219816
               total        used        free      shared  buff/cache   available
Mem:         7997696     1535104     5701232      340844      761360     5770964
Swap:       12287480     2067664    10219816
root@ip:/home/sd/Downloads/intro-code# exit
exit
```

Commands used for creating duplicates: `./make-copies.sh` and `ls disk-files`

```
 sd@ip  ~/Downloads/intro-code  ls disk-files
foo0.pdf     foo1500.pdf  foo2000.pdf  foo2501.pdf  foo3001.pdf  foo3502.pdf  foo4002.pdf  foo4503.pdf  foo502.pdf
foo1000.pdf  foo1501.pdf  foo2001.pdf  foo2502.pdf  foo3002.pdf  foo3503.pdf  foo4003.pdf  foo4504.pdf  foo503.pdf
foo1001.pdf  foo1502.pdf  foo2002.pdf  foo2503.pdf  foo3003.pdf  foo3504.pdf  foo4004.pdf  foo4505.pdf  foo504.pdf
foo1002.pdf  foo1503.pdf  foo2003.pdf  foo2504.pdf  foo3004.pdf  foo3505.pdf  foo4005.pdf  foo4506.pdf  foo505.pdf
foo1003.pdf  foo1504.pdf  foo2004.pdf  foo2505.pdf  foo3005.pdf  foo3506.pdf  foo4006.pdf  foo4507.pdf  foo506.pdf
foo1004.pdf  foo1505.pdf  foo2005.pdf  foo2506.pdf  foo3006.pdf  foo3507.pdf  foo4007.pdf  foo4508.pdf  foo507.pdf
foo1005.pdf  foo1506.pdf  foo2006.pdf  foo2507.pdf  foo3007.pdf  foo3508.pdf  foo4008.pdf  foo4509.pdf  foo508.pdf
foo1006.pdf  foo1507.pdf  foo2007.pdf  foo2508.pdf  foo3008.pdf  foo3509.pdf  foo4009.pdf  foo450.pdf   foo509.pdf
foo1007.pdf  foo1508.pdf  foo2008.pdf  foo2509.pdf  foo3009.pdf  foo350.pdf   foo400.pdf   foo4510.pdf  foo50.pdf
foo1008.pdf  foo1509.pdf  foo2009.pdf  foo250.pdf   foo300.pdf   foo3510.pdf  foo4010.pdf  foo4511.pdf  foo510.pdf
foo1009.pdf  foo150.pdf   foo200.pdf   foo2510.pdf  foo3010.pdf  foo3511.pdf  foo4011.pdf  foo4512.pdf  foo511.pdf
foo100.pdf   foo1510.pdf  foo2010.pdf  foo2511.pdf  foo3011.pdf  foo3512.pdf  foo4012.pdf  foo4513.pdf  foo512.pdf
foo1010.pdf  foo1511.pdf  foo2011.pdf  foo2512.pdf  foo3012.pdf  foo3513.pdf  foo4013.pdf  foo4514.pdf  foo513.pdf
foo1011.pdf  foo1512.pdf  foo2012.pdf  foo2513.pdf  foo3013.pdf  foo3514.pdf  foo4014.pdf  foo4515.pdf  foo514.pdf
foo1012.pdf  foo1513.pdf  foo2013.pdf  foo2514.pdf  foo3014.pdf  foo3515.pdf  foo4015.pdf  foo4516.pdf  foo515.pdf
foo1013.pdf  foo1514.pdf  foo2014.pdf  foo2515.pdf  foo3015.pdf  foo3516.pdf  foo4016.pdf  foo4517.pdf  foo516.pdf
foo1014.pdf  foo1515.pdf  foo2015.pdf  foo2516.pdf  foo3016.pdf  foo3517.pdf  foo4017.pdf  foo4518.pdf  foo517.pdf
foo1015.pdf  foo1516.pdf  foo2016.pdf  foo2517.pdf  foo3017.pdf  foo3518.pdf  foo4018.pdf  foo4519.pdf  foo518.pdf
foo1016.pdf  foo1517.pdf  foo2017.pdf  foo2518.pdf  foo3018.pdf  foo3519.pdf  foo4019.pdf  foo451.pdf   foo519.pdf
foo1017.pdf  foo1518.pdf  foo2018.pdf  foo2519.pdf  foo3019.pdf  foo351.pdf   foo401.pdf   foo4520.pdf  foo51.pdf
foo1018.pdf  foo1519.pdf  foo2019.pdf  foo251.pdf   foo301.pdf   foo3520.pdf  foo4020.pdf  foo4521.pdf  foo520.pdf
foo1019.pdf  foo151.pdf   foo201.pdf   foo2520.pdf  foo3020.pdf  foo3521.pdf  foo4021.pdf  foo4522.pdf  foo521.pdf
foo101.pdf   foo1520.pdf  foo2020.pdf  foo2521.pdf  foo3021.pdf  foo3522.pdf  foo4022.pdf  foo4523.pdf  foo522.pdf
foo1020.pdf  foo1521.pdf  foo2021.pdf  foo2522.pdf  foo3022.pdf  foo3523.pdf  foo4023.pdf  foo4524.pdf  foo523.pdf
foo1021.pdf  foo1522.pdf  foo2022.pdf  foo2523.pdf  foo3023.pdf  foo3524.pdf  foo4024.pdf  foo4525.pdf  foo524.pdf
foo1022.pdf  foo1523.pdf  foo2023.pdf  foo2524.pdf  foo3024.pdf  foo3525.pdf  foo4025.pdf  foo4526.pdf  foo525.pdf
foo1023.pdf  foo1524.pdf  foo2024.pdf  foo2525.pdf  foo3025.pdf  foo3526.pdf  foo4026.pdf  foo4527.pdf  foo526.pdf
foo1024.pdf  foo1525.pdf  foo2025.pdf  foo2526.pdf  foo3026.pdf  foo3527.pdf  foo4027.pdf  foo4528.pdf  foo527.pdf
foo1025.pdf  foo1526.pdf  foo2026.pdf  foo2527.pdf  foo3027.pdf  foo3528.pdf  foo4028.pdf  foo4529.pdf  foo528.pdf
foo1026.pdf  foo1527.pdf  foo2027.pdf  foo2528.pdf  foo3028.pdf  foo3529.pdf  foo4029.pdf  foo452.pdf   foo529.pdf
foo1027.pdf  foo1528.pdf  foo2028.pdf  foo2529.pdf  foo3029.pdf  foo352.pdf   foo402.pdf   foo4530.pdf  foo52.pdf
foo1028.pdf  foo1529.pdf  foo2029.pdf  foo252.pdf   foo302.pdf   foo3530.pdf  foo4030.pdf  foo4531.pdf  foo530.pdf
foo1029.pdf  foo152.pdf   foo202.pdf   foo2530.pdf  foo3030.pdf  foo3531.pdf  foo4031.pdf  foo4532.pdf  foo531.pdf
foo102.pdf   foo1530.pdf  foo2030.pdf  foo2531.pdf  foo3031.pdf  foo3532.pdf  foo4032.pdf  foo4533.pdf  foo532.pdf
```

Running `disk.c`

Disk usage using `iostat`

```
 sd@ip  ~/Downloads/intro-code  iostat
Linux 5.18.10-76051810-generic (ip)     14/08/22        _x86_64_        (4 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          36.04   11.29   13.74    0.09    0.00   38.83

Device             tps    kB_read/s    kB_wrtn/s    kB_dscd/s    kB_read    kB_wrtn    kB_dscd
dm-0              4.82         4.05        15.22         0.00    2065389    7772424          0
sda              10.54       177.00       224.16         0.00   90370857  114450462          0
sdb               0.01         0.41         0.00         0.00     211636        176          0


 sd@ip  ~/Downloads/intro-code  iostat
Linux 5.18.10-76051810-generic (ip)     14/08/22        _x86_64_        (4 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          36.04   11.29   13.74    0.09    0.00   38.83

Device             tps    kB_read/s    kB_wrtn/s    kB_dscd/s    kB_read    kB_wrtn    kB_dscd
dm-0              4.82         4.05        15.22         0.00    2065389    7772424          0
sda              10.56       178.43       224.16         0.00   91103305  114450590          0
sdb               0.01         0.41         0.00         0.00     211636        176          0
```

Disk usage using `pidstat`

Command: `pidstat -dl 1`

```
 sd@ip  ~/Downloads/intro-code  pidstat -dl 1
Linux 5.18.10-76051810-generic (ip)     14/08/22        _x86_64_        (4 CPU)

12:20:32 PM IST   UID       PID   kB_rd/s   kB_wr/s kB_ccwr/s iodelay  Command
12:20:33 PM IST  1000    447099 253512.38      0.00      0.00       0  ./disk

12:20:33 PM IST   UID       PID   kB_rd/s   kB_wr/s kB_ccwr/s iodelay  Command
12:20:34 PM IST  1000    447099 251152.00      0.00      0.00       0  ./disk

12:20:34 PM IST   UID       PID   kB_rd/s   kB_wr/s kB_ccwr/s iodelay  Command
12:20:35 PM IST  1000    447099 248408.00      0.00      0.00       0  ./disk

12:20:35 PM IST   UID       PID   kB_rd/s   kB_wr/s kB_ccwr/s iodelay  Command
12:20:36 PM IST  1000    447099 250832.00      0.00      0.00       0  ./disk

12:20:36 PM IST   UID       PID   kB_rd/s   kB_wr/s kB_ccwr/s iodelay  Command
12:20:37 PM IST  1000    447099 237320.00      0.00      0.00       0  ./disk

12:20:37 PM IST   UID       PID   kB_rd/s   kB_wr/s kB_ccwr/s iodelay  Command
12:20:38 PM IST  1000    447099 216652.00      0.00      0.00       0  ./disk

12:20:38 PM IST   UID       PID   kB_rd/s   kB_wr/s kB_ccwr/s iodelay  Command
12:20:39 PM IST  1000    447099 203964.00      0.00      0.00       0  ./disk

12:20:39 PM IST   UID       PID   kB_rd/s   kB_wr/s kB_ccwr/s iodelay  Command
12:20:40 PM IST  1000    447099 200444.00      0.00      0.00       0  ./disk

12:20:40 PM IST   UID       PID   kB_rd/s   kB_wr/s kB_ccwr/s iodelay  Command
12:20:41 PM IST  1000    447099 168592.00      0.00      0.00       0  ./disk

12:20:41 PM IST   UID       PID   kB_rd/s   kB_wr/s kB_ccwr/s iodelay  Command
12:20:42 PM IST  1000    447099 173748.00      0.00      0.00       0  ./disk

12:20:42 PM IST   UID       PID   kB_rd/s   kB_wr/s kB_ccwr/s iodelay  Command
12:20:43 PM IST  1000    447099 154184.00      0.00      0.00       0  ./disk

12:20:43 PM IST   UID       PID   kB_rd/s   kB_wr/s kB_ccwr/s iodelay  Command
12:20:44 PM IST  1000    447099 137752.00      0.00      0.00       0  ./disk

12:20:44 PM IST   UID       PID   kB_rd/s   kB_wr/s kB_ccwr/s iodelay  Command
12:20:45 PM IST  1000    447099 115136.00      0.00      0.00       0  ./disk
^C

Average:      UID       PID   kB_rd/s   kB_wr/s kB_ccwr/s iodelay  Command
Average:     1000    447099 201101.30      0.00      0.00       0  ./disk
 sd@ip  ~/Downloads/intro-code
```

Running `disk1.c`

Disk usage using `iostat`

```
 sd@ip  ~/Downloads/intro-code  iostat
Linux 5.18.10-76051810-generic (ip)     17/08/22        _x86_64_        (4 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          33.76    6.43   12.42    0.09    0.00   47.30

Device             tps    kB_read/s    kB_wrtn/s    kB_dscd/s    kB_read    kB_wrtn    kB_dscd
dm-0              1.46         0.76         5.11         0.00     138006     925752          0
sda               6.51        50.06       150.24         0.00    9071135   27221450          0
sdb               0.01         0.68         0.00         0.00     122496          0          0
```

Disk usage using `pidstat`

Command: `pidstat -dl 1 -G ./disk1`

```
 sd@ip  ~/Downloads/intro-code  pidstat -dl 1 -G ./disk1
Linux 5.18.10-76051810-generic (ip)     17/08/22        _x86_64_        (4 CPU)

04:07:53 PM IST   UID       PID   kB_rd/s   kB_wr/s kB_ccwr/s iodelay  Command

04:07:54 PM IST   UID       PID   kB_rd/s   kB_wr/s kB_ccwr/s iodelay  Command
04:07:55 PM IST  1000    101989   1188.00      0.00      0.00       0  ./disk1

04:07:55 PM IST   UID       PID   kB_rd/s   kB_wr/s kB_ccwr/s iodelay  Command

04:07:56 PM IST   UID       PID   kB_rd/s   kB_wr/s kB_ccwr/s iodelay  Command

04:07:57 PM IST   UID       PID   kB_rd/s   kB_wr/s kB_ccwr/s iodelay  Command

04:07:58 PM IST   UID       PID   kB_rd/s   kB_wr/s kB_ccwr/s iodelay  Command

04:07:59 PM IST   UID       PID   kB_rd/s   kB_wr/s kB_ccwr/s iodelay  Command

04:08:00 PM IST   UID       PID   kB_rd/s   kB_wr/s kB_ccwr/s iodelay  Command
^C

Average:      UID       PID   kB_rd/s   kB_wr/s kB_ccwr/s iodelay  Command
```

DECS Assignment 2

- Shivansh Dhiman, 22m2120

Question 1

Debug the pointers.cpp program given to you using GDB. The program contains some pointer related operations. A bug has been deliberately introduced in the code so that it generates a segmentation fault. Your task is to use GDB to find the line number of the wrong statement. Please make use of the GDB commands provided above in order to find the bug.

- Compiled using `g++ pointers.cpp -g -o pointers` and ran `pointers`.
- After debugging, the bug was found at line number `13`. This caused a segmentation fault, because the code tried to dereference a NULL pointer.

```
 sd@ip  ~/Downloads/intro-debug-code  ./pointers
45
[1]    448855 segmentation fault (core dumped)  ./pointers
```

```
 sd@ip  ~/Downloads/intro-debug-code  gdb pointers
GNU gdb (Ubuntu 12.0.90-0ubuntu1) 12.0.90
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from pointers...
```

```
(gdb) break pointers.cpp:main
Breakpoint 1 at 0x11dc: file pointers.cpp, line 4.
(gdb) break pointers.cpp:7
Breakpoint 2 at 0x11f9: file pointers.cpp, line 7.
(gdb) run
Starting program: /home/sd/Downloads/intro-debug-code/pointers
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main (argc=1, argv=0x7fffffffe0a8) at pointers.cpp:4
4       int main(int argc, char* argv[]) {
(gdb)
```

```
(gdb) run
Starting program: /home/sd/Downloads/intro-debug-code/pointers
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main (argc=1, argv=0x7fffffffe0a8) at pointers.cpp:4
4       int main(int argc, char* argv[]) {
(gdb) bt
#0  main (argc=1, argv=0x7fffffffe0a8) at pointers.cpp:4
(gdb) print a
$1 = -136146744
```

```
(gdb) f
#0  main (argc=1, argv=0x7fffffffe0a8) at pointers.cpp:4
4       int main(int argc, char* argv[]) {
```

```
(gdb) c
Continuing.

Breakpoint 2, main (argc=1, argv=0x7fffffffe0a8) at pointers.cpp:7
7         int *p = &a;
(gdb) print a
$8 = 45
```

```
(gdb) f
#0  main (argc=1, argv=0x7fffffffe0a8) at pointers.cpp:7
7         int *p = &a;
(gdb) next
8         int *q = NULL;
(gdb) next
10        cout << *p << endl;
```

```
(gdb) f
#0  main (argc=1, argv=0x7fffffffe0a8) at pointers.cpp:12
12        p = q;
(gdb) step
13        cout << *p << endl;
```

```
(gdb) print p
$10 = (int *) 0x0
(gdb) print q
$11 = (int *) 0x0
```

```
(gdb) list
8         int *q = NULL;
9
10        cout << *p << endl;
11
12        p = q;
13        cout << *p << endl;
14
15        p = &b;
16        cout << *p << endl;
17
(gdb) list main
1       #include<iostream>
2       using namespace std;
3
4       int main(int argc, char* argv[]) {
5
6         int a = 45; int b = 22;
7         int *p = &a;
8         int *q = NULL;
9
10        cout << *p << endl;
```

```
(gdb) next

Program received signal SIGSEGV, Segmentation fault.
0x000055555555523e in main (argc=1, argv=0x7fffffffe0a8) at pointers.cpp:13
13        cout << *p << endl;
```

```
(gdb) help c
continue, fg, c
Continue program being debugged, after signal or breakpoint.
Usage: continue [N]
If proceeding from breakpoint, a number N may be used as an argument,
which means to set the ignore count of that breakpoint to N - 1 (so that
the breakpoint won't break until the Nth time it is reached).

If non-stop mode is enabled, continue only the current thread,
otherwise all the threads in the program are continued.  To
continue all stopped threads in non-stop mode, use the -a option.
Specifying -a and an ignore count simultaneously is an error.
```

```
(gdb) quit
 sd@ip  ~/Downloads/intro-debug-code
```

Question 2

Debug the fibonacci.cpp program given to you using GDB. This program is supposed to print fibonacci numbers until a certain value of n. However, there is a logical error introduced in the code which causes it to print wrong output. Your task is to use GDB to debug the program. You must insert suitable breakpoints, pause program execution, print intermediate values of variables from GDB, and monitor the execution step by step in order to find the logical error. Even if you can identify the error without stepping through the code, you must be able to demonstrate the process of debugging using GDB during your evaluation.

- Compiled using `g++ fibonacci.cpp -g -o fibonacci` and ran `./fibonacci`
- After debugging, the bug was found at line numbers `16 and 17`. The `last` variable should first be copied to `second_last`, and then `next` should be copied to `last`. Here, the order of these operations is reversed, thus leading to incorrect output.

```
 sd@ip  ~/Downloads/intro-debug-code  ./fibonacci
1
1
2
4
8
16
32
64
128
256
512
1024
```

Debugging using `gdb fibonacci`

```
GNU gdb (Ubuntu 12.0.90-0ubuntu1) 12.0.90
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from fibonacci...
```

```
(gdb) break fibonacci.cpp:main
Breakpoint 1 at 0x11bc: file fibonacci.cpp, line 6.
(gdb) break fibonacci.cpp:11
Breakpoint 2 at 0x11d1: file fibonacci.cpp, line 11.
(gdb) break fibonacci.cpp:14
Breakpoint 3 at 0x1222: file fibonacci.cpp, line 14.
```

```
(gdb) run
Starting program: /home/sd/Downloads/intro-debug-code/fibonacci
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main (argc=1, argv=0x7fffffffe078) at fibonacci.cpp:6
6         int n = 10;
(gdb) bt
#0  main (argc=1, argv=0x7fffffffe078) at fibonacci.cpp:6
```

```
(gdb) c
Continuing.

Breakpoint 2, main (argc=1, argv=0x7fffffffe078) at fibonacci.cpp:11
11        cout << second_last << endl << last << endl;
(gdb) step
1
1
13        for(int i=1; i<=10; i++) {
(gdb) c
Continuing.

Breakpoint 3, main (argc=1, argv=0x7fffffffe078) at fibonacci.cpp:14
14          int next = second_last + last;
(gdb) print last
$1 = 1
(gdb) print second_last
$2 = 1
```

```
(gdb) c
Continuing.
2

Breakpoint 3, main (argc=1, argv=0x7fffffffe078) at fibonacci.cpp:14
14          int next = second_last + last;
(gdb) print last
$3 = 2
(gdb) print second_last
$4 = 2
```

```
(gdb) step
15          cout << next << endl;
(gdb) print next
$5 = 4
(gdb) next
4
16          last = next;
(gdb) next
17          second_last = last;
(gdb) print last
$6 = 4
(gdb) print second_last
$7 = 2
(gdb) step
13        for(int i=1; i<=10; i++) {
(gdb) print last
$8 = 4
(gdb) print second_last
$9 = 4
```

```
(gdb) step

Breakpoint 3, main (argc=1, argv=0x7fffffffe078) at fibonacci.cpp:14
14          int next = second_last + last;
(gdb) step
15          cout << next << endl;
(gdb) print next
$10 = 8
```

```
(gdb) list
10
11        cout << second_last << endl << last << endl;
12
13        for(int i=1; i<=10; i++) {
14          int next = second_last + last;
15          cout << next << endl;
16          last = next;
17          second_last = last;
18        }
19
(gdb) help list
list, l
List specified function or line.
With no argument, lists ten more lines after or around previous listing.
"list -" lists the ten lines before a previous ten-line listing.
One argument specifies a line, and ten lines are listed around that line.
Two arguments with comma between specify starting and ending lines to list.
Lines can be specified in these ways:
  LINENUM, to list around that line in current file,
  FILE:LINENUM, to list around that line in that file,
  FUNCTION, to list around beginning of that function,
  FILE:FUNCTION, to distinguish among like-named static functions.
  *ADDRESS, to list around the line containing that address.
With two args, if one is empty, it stands for ten lines away from
the other arg.

By default, when a single location is given, display ten lines.
This can be changed using "set listsize", and the current value
can be shown using "show listsize".
(gdb) quit
A debugging session is active.

        Inferior 1 [process 13073] will be killed.

Quit anyway? (y or n) y
```

Question 3

A program memory bugs.c is provided in the folder. This program is riddled with memory bugs. You may be able to find some bugs just by looking at the code also. Valgrind can help you find these bugs automatically. Your job is to compile the program using the command provided above and use Valgrind to find the possible issues present in the program. You should first understand the different issues Valgrind can detect, and then use the command given above to find the issues present in the program. You might be asked to provide possible reasons and fixes for those issues during your evaluation.

> Valgrind a collection of tools for debugging and memory profiling. Memcheck is a tool that detects memory management problems. It checks many issues in a program like illegal memory accesses, memory leaks, using uninitialized variables and more.

Issues found:

- In line number **19**, `write` function writes the first 10 bytes of array `arr` to `stdout`, but the array is not yet initialized.
- In line number **26**, the code is trying to write a character to the memory address `p`, but the address that `p` was pointing to, has already been freed.
- In line number **29**, the code is trying to read from the memory address `p`, but the address that `p` was pointing to, has already been freed.
- In line number **35**, the code is trying to free a statically initialized array. This is not valid as lifetime of statically allocated variables is the entire execution of the program. If it were dynamically allocated, then there had been no errors.
- `p` on line number **16** and `q` on line number **32** are dynamically allocated and hence need to be freed before `main()` function finishes. Since these are not freed here, it causes two memory leaks in the program.

```
✘ sd@ip ~/Downloads/intro-debug-code valgrind --tool=memcheck --leak-check=yes --show-reachable=yes --num-callers=20 ./memory_bugs
==16466== Memcheck, a memory error detector
==16466== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==16466== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==16466== Command: ./memory_bugs
==16466==
==16466== Syscall param write(buf) points to uninitialised byte(s)
==16466==    at 0x4982A37: write (write.c:26)
==16466==    by 0x109235: main (memory_bugs.c:19)
==16466==  Address 0x1ffefffef0 is on thread 1's stack
==16466==  in frame #1, created by main (memory_bugs.c:9)
==16466==
==16466== Invalid write of size 1
==16466==    at 0x109254: main (memory_bugs.c:26)
==16466==  Address 0x4a990a0 is 0 bytes inside a block of size 12 free'd
==16466==    at 0x484B27F: free (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==16466==    by 0x10924F: main (memory_bugs.c:23)
==16466==  Block was alloc'd at
==16466==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==16466==    by 0x10923F: main (memory_bugs.c:22)
==16466==
==16466== Invalid read of size 1
==16466==    at 0x10925B: main (memory_bugs.c:29)
==16466==  Address 0x4a990a0 is 0 bytes inside a block of size 12 free'd
==16466==    at 0x484B27F: free (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==16466==    by 0x10924F: main (memory_bugs.c:23)
==16466==  Block was alloc'd at
==16466==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==16466==    by 0x10923F: main (memory_bugs.c:22)
==16466==
A
==16466== Invalid free() / delete / delete[] / realloc()
==16466==    at 0x484B27F: free (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==16466==    by 0x109290: main (memory_bugs.c:35)
==16466==  Address 0x1ffefffef0 is on thread 1's stack
==16466==  in frame #1, created by main (memory_bugs.c:9)
==16466==
==16466==
```

```
==16466== HEAP SUMMARY:
==16466==     in use at exit: 80 bytes in 2 blocks
==16466==   total heap usage: 4 allocs, 3 frees, 1,116 bytes allocated
==16466==
==16466== 30 bytes in 1 blocks are definitely lost in loss record 1 of 2
==16466==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==16466==    by 0x10920D: main (memory_bugs.c:16)
==16466==
==16466== 50 bytes in 1 blocks are definitely lost in loss record 2 of 2
==16466==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==16466==    by 0x109280: main (memory_bugs.c:32)
==16466==
==16466== LEAK SUMMARY:
==16466==    definitely lost: 80 bytes in 2 blocks
==16466==    indirectly lost: 0 bytes in 0 blocks
==16466==      possibly lost: 0 bytes in 0 blocks
==16466==    still reachable: 0 bytes in 0 blocks
==16466==         suppressed: 0 bytes in 0 blocks
==16466==
==16466== Use --track-origins=yes to see where uninitialised values come from
==16466== For lists of detected and suppressed errors, rerun with: -s
==16466== ERROR SUMMARY: 6 errors from 6 contexts (suppressed: 0 from 0)
```