Name: Anubhav Jana
Roll: 22M2109
Course: CS744

sudo apt install build-essential

1. a . more */proc/*cpuinfo

The file **/proc/cpuinfo** displays what type of processor your system is running including the number of CPUs present.

processor – Provides each processor with an identifying number. If we have 1 processor, then it will be denoted by 0. If you have one processor it will display a 0. More than 1 proc, it will display each proc info separately with their own ids. Proc 0, Proc etc.

processor : 0
vendor_id : GenuineIntel
cpu family : 6
model : 158
model name : Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz
stepping : 10
cpu MHz : 2591.998
cache size : 12288 KB
physical id : 0
siblings : 2
core id : 0
cpu cores : 2
address sizes : 39 bits physical, 48 bits virtual

processor : 1
vendor_id : GenuineIntel
cpu family : 6
model : 158
model name : Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz
stepping : 10
cpu Mhz : 2591.998
cache size : 12288 KB
physical id : 0
siblings : 2
core id : 1
cpu cores : 2
address sizes : 39 bits physical, 48 bits virtual

Lscpu - used to get CPU information of the system. This command fetches the CPU architecture information from the “sysfs” and /proc/cpuinfo files and displays it in a terminal.

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): 2
On-line CPU(s) list: 0,1
Vendor ID: GenuineIntel
Model name: Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz
CPU family: 6 --→type of processor
Model: 158
Thread(s) per core: 1
Core(s) per socket: 2
Virtualization features:
Hypervisor vendor: KVM
Virtualization type: full
Caches (sum of all):
L1d: 64 KiB (2 instances)
L1i: 64 KiB (2 instances)
L2: 512 KiB (2 instances)
L3: 24 MiB (2 instances)
NUMA:
NUMA node(s): 1
NUMA node0 CPU(s): 0,1

b. Number of cores = 2
c. Number of processors =2 – proc 0, proc 1
d. Frequency of each processor = **2.60GHz**
e. Architecture of my CPU = **x86_64**
I ran the command "**more /proc/meminfo**" for memory related information.

f. MemTotal: **7242472 kB - Total physical memory**
g. MemFree: **401540 kB - Free memory**
MemAvailable: 4553468 kB

h. **vmstat -f → 38984 forks**

cat /proc/stat

ctxt 11222665
btime 1659853808
processes 39031
procs_running 1
procs_blocked 0

Number of context switches **– 11222665**

Or command vmstat -s will give the no of forks and context switches.

2. Pid **= 39188**
CPU usage **= 100%**
Mem usage **= 0**

State of the process is **Running (R)** given by the column **"S" in "top"**

Other states are: **Sleeping – S , Zombie – Z , Stopping – T**

3. a. I ran the command **ps -ejH –forest**

gnome shell → gnome-terminal → bash → cpu-print

Bash runs the cpu print executable . So required PID **is of bash = 5013**

b. systemd (752) → gnome shell (992) → gnome-terminal (4953)→ bash (5013) → cpu-print (39443)

ps -f <pid of the cpu-print> will give ppid , and this way we can go back 5 generations

ps -f 11392 – cpu-print

```
UID        PID    PPID  C STIME TTY      STAT   TIME CMD
anubhav    11392    5951  4 12:36 pts/0    T      0:22 ./cpu-print
```

ps -f 5951

```
UID        PID    PPID  C STIME TTY      STAT   TIME CMD
anubhav     5951    5835  0 10:23 pts/0    Ss     0:00 bash
```

ps -f 5835

```
UID        PID    PPID  C STIME TTY      STAT   TIME CMD
anubhav     5835    1340  0 10:23 ?        Ssl    1:11 /usr/libexec/gnome-terminal
```

ps -f 1340

```
UID        PID    PPID  C STIME TTY      STAT   TIME CMD
anubhav     1340       1  0 10:15 ?        Ss     0:00 /lib/systemd/systemd –u
```

ps -f 1

```
UID        PID    PPID  C STIME TTY      STAT   TIME CMD
root         1       0  0 10:15 ?        Ss     0:01 /lib/systemd/systemd spl
```

3.c. ./cpu-print > /tmp/tmp.txt &
[3] 11710
File descriptors help index into the file descriptor table (FDT) which stores pointer to resources such as devices, terminal , pipes etc.
After executing, it shows the file descriptors pointing to which resource.
Following is the output I got.

ls l /proc/4333/fd {Here, 4333 is the process id of the spawned process “cpu-print”)
total 0
lrwx------ 1 anubhav anubhav 64 Aug 12 19:42 0 -> /dev/pts/0 --→ Standard input
l-wx------ 1 anubhav anubhav 64 Aug 12 19:42 1 -> /tmp/tmp.txt --→ Standard output
lrwx------ 1 anubhav anubhav 64 Aug 12 19:42 2 -> /dev/pts/0 --→ Standard error

3.d. ./cpu-print | grep hello &

The id of the newly spawned process is **4487**
This time stdout is pointing to resource **“pipe”** unlike the previous one which pointed to a **file tmp.txt**

Output :

ls -l /proc/4487/fd
total 0
lrwx------ 1 anubhav anubhav 64 Aug 12 19:49 0 -> /dev/pts/0
l-wx------ 1 anubhav anubhav 64 Aug 12 19:49 1 -> 'pipe:[45521]'
lrwx------ 1 anubhav anubhav 64 Aug 12 19:49 2 -> /dev/pts/0

3. e. Here , we need to check which commands are built in (internal commands) and which are external **(i.e. the shell has to look for its PATH and a new process has to be spawned and then the command gets executed)**
Execution of the internal commands is fast since the shell does not have to search for its PATH and no new process has to be spawned to run the executable.

type cat
cat is hashed (/usr/bin/cat) → This means "cat" command is external command.

type cd
cd is a shell builtin → INTERNAL COMMAND

anubhav@anubhav-VirtualBox:~/Desktop/intro-code$ **type history**
history is a shell builtin → INTERNAL COMMAND

anubhav@anubhav-VirtualBox:~/Desktop/intro-code$ **type ps**
ps is hashed (/usr/bin/ps) → EXTERNAL COMMAND

ls is an external command

4. Virtual memory assigned to both memory1 and memory2 **is 6556 (VIR) ,and physical memory is 4824 B (PHY) → memory1 and 4828 B (PHY) for memory2.**
I used "htop" to check the fields.
Memory1.c is allocating a large memory array but not using it

5. I checked on which disk the current directory is mounted on
anubhav@anubhav-VirtualBox:~/Desktop/intro-debug-code$ **df .**

Filesystem	**1K-blocks**	**Used**	**Available**	**Use%**	**Mounted on**
/dev/sda3	**83296520**	**18190984**	**60828288**	**24%**	**/**

>> iostat -p sda (After running disk.c)

avg-cpu:	**%user**	**%nice**	**%system**	**%iowait**	**%steal**	**%idle**
	30.13	**3.13**	**33.25**	**0.04**	**0.00**	**33.45**

Device	tps	kB_read/s	kB_wrtn/s	kB_dscd/s	kB_read	kB_wrtn	kB_dscd
sda3	3.61	103.13	151.59	0.00	10670341	15684652	0

After running disk1.c :

sda3	3.62	103.63	150.89	0.00	10778753	15694652	0

LAB 2 – Debugging using GDB

1. I first generated the executable using **g++ pointers.cpp -g -o pointers followed by gdb pointers**
There was an error at line number 13 . Segmentation Fault and this was because the pointer variable "q" was assigned NULL and we tried to dereference a null pointer since we did p=q. So, I commented out the line and then the program executed successfully. If we try to access a memory location which is not assigned to user, then that is out of priviledge and segmentation fault will occurs

2. Breakpoint applied at line 11
Starting program: /home/anubhav/Desktop/intro-debug-code/fib
[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=0x7fffffffe0e8) at fibonacci.cpp:11
11 cout <<
second_last << endl << last << endl;

(gdb) info locals → will give the values of variables till that breakpoint
n = 10
second_last = 1
last = 1

(gdb) break 13
Breakpoint 2 at 0x555555555219: file fibonacci.cpp, line 13.
(gdb) info locals
n = 10
second_last = 1
last = 1

(gdb) next (next statement afterbreak)
14 **int next =**
second_last + last;
(gdb) info locals
next = 0
i = 1
n = 10

second_last = 1
last = 1

(gdb) info locals
next = 4
i = 2
n = 10
second_last = 2
last = 4

n
13 for(int i=1;
i<=10; i++) {
(**gdb) info locals**
i = 3
n = 10
second_last = 8
last = 8

Running intermediate values, got it
Now, I will run the entire programme using “run”

(gdb) run
Starting program: /home/anubhav/Desktop/intro-debug-code/fib
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
1
1
2
4
8
16
32
64
128
256
512
1024
[Inferior 1 (process 7556) exited normally]

Also its printing 12 values instead of 10 terms because of loop terminate condition.

CORRECTIONS:

second_last can be initialized with 0 instead of 1.
loop variable i=2 and I<10 because we are already printing first 2 values outside the loop
swap statements should be changed to:

second_last = last;
last=next;

INSTEAD OF

last = next;
second_last = last;

3. anubhav@anubhav-VirtualBox:~/Desktop/intro-debug-code$ **valgrind --tool=memcheck --leak-check=yes --show-reachable=yes --num-callers=20 ./memory_bugs**
==11208== Memcheck, a memory error detector
==11208== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==11208== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==11208== Command: ./memory_bugs
==11208==
==11208== Syscall param write(buf) points to uninitialised byte(s)
==11208== at 0x497EA37: write (write.c:26)
==11208== by 0x109235: main (memory_bugs.c:19)
==11208== Address 0x1ffeffff30 is on thread 1's stack
==11208== in frame #1, created by main (memory_bugs.c:9)
==11208==
==11208== Invalid write of size 1
==11208== at 0x109254: main (memory_bugs.c:26)
==11208== Address 0x4a950a0 is 0 bytes inside a block of size 12 free'd
==11208== at 0x484B27F: free (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==11208== by 0x10924F: main (memory_bugs.c:23)
==11208== Block was alloc'd at
==11208== at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==11208== by 0x10923F: main (memory_bugs.c:22)
==11208==
==11208== Invalid read of size 1
==11208== at 0x10925B: main (memory_bugs.c:29)
==11208== Address 0x4a950a0 is 0 bytes inside a block of size 12 free'd
==11208== at 0x484B27F: free (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==11208== by 0x10924F: main (memory_bugs.c:23)
==11208== Block was alloc'd at
==11208== at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==11208== by 0x10923F: main (memory_bugs.c:22)
==11208==
A
==11208== Invalid free() / delete / delete[] / realloc()
==11208== at 0x484B27F: free (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==11208== by 0x109290: main (memory_bugs.c:35)
==11208== Address 0x1ffeffff30 is on thread 1's stack
==11208== in frame #1, created by main (memory_bugs.c:9)
==11208==
==11208==

LEAK SUMMARY:
==21916== definitely lost: 80 bytes in 2 blocks
==21916== indirectly lost: 0 bytes in 0 blocks
==21916== possibly lost: 0 bytes in 0 blocks
==21916== still reachable: 0 bytes in 0 blocks
==21916== suppressed: 0 bytes in 0 blocks

ISSUES:

Seeing the above output from valgrind, I have summarized the issues as follows**.**

1. Line 26 *p='A': Write system call points to uninitialized bytes **. This is because pointer** "p" is not initialized because it was freed in line 23.

2. **Line 26:** Since, p was already freed, that memory is not anymore under user priviledge, and hence the error **"Invalid write of size 1"** (Size 1 is because we are writing a data of type char which is 1 byte.)

3. Line 29: printf("%c\n",*p) : We are trying to read 1 byte from a memory that is not in the range of user. Since after freeing, its not allocated any memory using malloc(). Therefore, the error is **"Invalid read of size 1"** .

4. Line 35 – free(arr) : Improper use of free(). It is used for dynamically allocated arrays and not for static arrays. Therefore , "**Invalid free() / delete / delete[] / realloc()"** error message .
5. Line 16: memory of **30 bytes** is allocated to char *p but is used again in allocating **12 bytes at line 22 without freeing it**. So, this 30 block of data is definitely lost. It is called memory leak.
Valgrind error: 30 bytes in 1 blocks are definitely lost in loss record 1 of 2

6. Line 32 allocates 50 bytes to *q but is not freed. So, this is also a loss of block of data of 50 bytes.
Valgrind error: 50 bytes in 1 blocks are definitely lost in loss record 2 of 2.

LEAK SUMMARY:
==21916== definitely lost: 80 bytes in 2 blocks

HEAP SUMMARY:
==21916== in use at exit: 80 bytes in 2 blocks
==21916== total heap usage: 4 allocs, 3 frees, 1,116 bytes allocated

since , from above given summary, it is evident that **leak occurs because the number of allocs is NOT equal to number of frees.**