Group 2
Swapnil Rustagi B17104
Vipul Sharma B17069

CS307 System Practicum
Assignment 2

Problem 1

Dining Philosophers Problem

Inferences -

Approaches used were -
Trivial Approach -
Each philosopher attempts to pick up the left fork first and then the second fork. Use an array of mutex locks to simulate forks which each philosopher can hold. Then each philosopher drops the left fork and then right. This is prone to deadlock as it is possible that each of the philosophers picks up the left fork and then every philosopher will be waiting for a right fork which will never become available.
In circular fashion order is
P0 F1 P1 F2 P2 F3 P3 F4 P4 F0 // Here, P stands for Philosopher and F for Fork.

Approach 1 (main.cpp)-
Change the trivial approach so that the odd numbered philosophers pick up the left fork first and the even philosophers pick up the right fork first. This will prevent any deadlocks from occurring as at least one philosopher can always acquire both forks. Starvation also does not appear to be a problem here but can occur if say P0 is waiting for P1 to drop his left fork but after dropping P1 again picks it up. This is highly unlikely because of the random think times introduced in the question. The main problem with this solution is that P4 will eat the most in this case. This is because P0 and P1 will fight for F1 and P2 and P3 will fight for F3 but there will always be both forks available for P4 , i.e. F0 and F4 will always be initially available. Thus, this solution gives P4 a slight advantage over others.

Approach 2 (main2.cpp)-
Change the trivial approach so that all philosophers except the last one pick up the right fork first. The last philosopher picks up the left fork first. This will prevent any deadlocks from occurring as at least one philosopher can always acquire both forks. Starvation is also unlikely because of the random times of thinking. So, in our case P0, P1, P2 and P3 pickup the right fork first and P4 picks up the left fork first. Here also, we observe that P0 does the most eating in each case because P4 and P3 will fight for F4 and both F0 and F1 will always be available for P0 initially. Thus, this solution gives P0 a slight advantage over others.

How to fix uneven distribution?
This is highly unlikely to solve in the classical problem. In each solution to this we have to modify the problem by some amount.

1. We can introduce a waiter (queue) which keeps hold of how much food each philosopher has eaten and assign forks accordingly.
2. We can also modify Approach 2 we used above to cycle the philosopher with the opposite fork picking pattern after some interval of time, say every 2 minutes. This will ensure that each philosopher gets a fair chance and will ensure a fair distribution.

Observations -
Approach 1 (main.cpp)-
out1.txt

Philosopher	**Thinking**	**Eating**	**1 Fork Acquired**
P0	616	616	616
P1	634	635	635
P2	649	649	649
P3	673	673	673
P4	692	693	693

out2.txt

Philosopher	**Thinking**	**Eating**	**1 Fork Acquired**
P0	614	614	615
P1	639	639	639
P2	648	648	648
P3	664	665	665
P4	699	699	700

out3.txt

Philosopher	**Thinking**	**Eating**	**1 Fork Acquired**
P0	612	612	613
P1	636	637	637
P2	648	648	648
P3	670	671	671
P4	698	698	698

out4.txt

Philosopher	Thinking	Eating	1 Fork Acquired
P0	614	614	614
P1	640	640	640
P2	650	650	650
P3	664	665	665
P4	697	697	697

out5.txt

Philosopher	Thinking	Eating	1 Fork Acquired
P0	620	620	620
P1	636	636	636
P2	653	654	654
P3	661	661	661
P4	694	695	695

Approach 2 (main2.cpp)-

out2_1.txt

Philosopher	Thinking	Eating	1 Fork Acquired
P0	691	691	691
P1	680	680	680
P2	664	665	665
P3	618	618	618
P4	658	658	658

out2_2.txt

Philosopher	Thinking	Eating	1 Fork Acquired
P0	698	698	699

P1	679	679	679
P2	668	668	668
P3	615	615	615
P4	650	651	651

out2_3.txt

Philosopher	**Thinking**	**Eating**	**1 Fork Acquired**
P0	688	688	688
P1	671	671	672
P2	648	648	648
P3	595	596	596
P4	641	641	641

out2_4.txt

Philosopher	**Thinking**	**Eating**	**1 Fork Acquired**
P0	700	700	700
P1	665	665	665
P2	642	642	642
P3	599	600	600
P4	637	637	637

out2_5.txt

Philosopher	**Thinking**	**Eating**	**1 Fork Acquired**
P0	692	692	693
P1	678	678	678
P2	671	672	672
P3	612	612	613
P4	656	657	657

Problem 2

Matrix Multiplication

Observations -

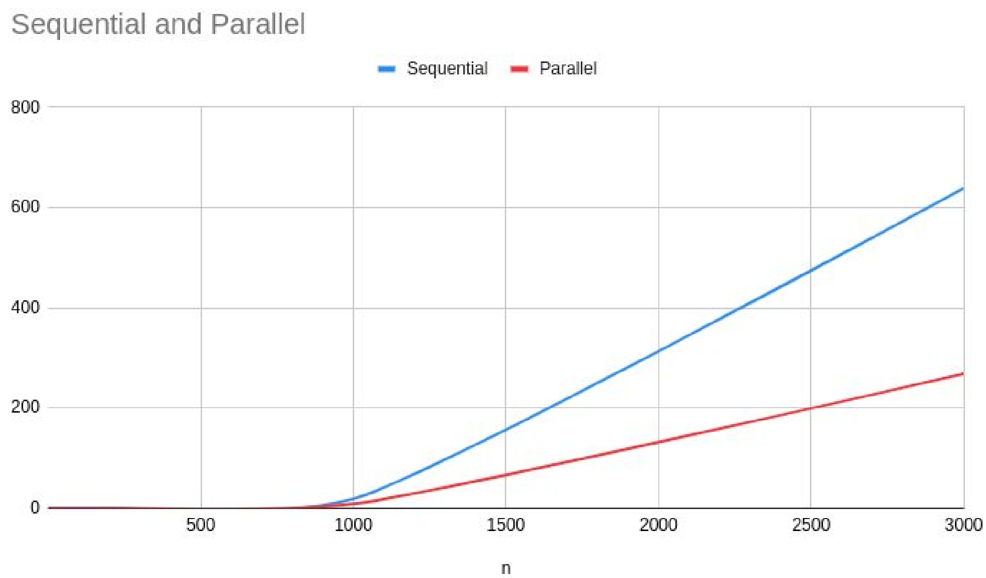
Parallel program runs with 4 threads
All times are in seconds

	n=1		n=50		n=200		n=1000		n=3000	
	Sequential	Parallel	Sequential	Parallel	Sequential	Parallel	Sequential	Parallel	Sequential	Parallel
	0.0000008	0.00057741	0.00750554	0.006976	0.12302338	0.06901864	18.0772234	8.16928457	643.635285	267.5291197
	0.00000082	0.0007222	0.00729521	0.0068415	0.12296388	0.07060032	18.02407869	8.00962116	639.38174	266.0654095
	0.00000096	0.00067246	0.00746023	0.00527879	0.1297082	0.06516734	17.93985098	8.06493118	632.3827295	267.0017897
	0.00000076	0.00055715	0.00754953	0.00695252	0.11847829	0.07076258	18.25651759	8.10387556	635.496553	270.2624048
	0.00000104	0.00051749	0.00724737	0.00668165	0.12440977	0.06415848	18.28275159	8.09542102	641.7354021	267.4019674
Mean:	0.000000876	0.000609342	0.007411576	0.006546092	0.123716704	0.067941472	18.11608445	8.088626698	638.5263419	267.6521382

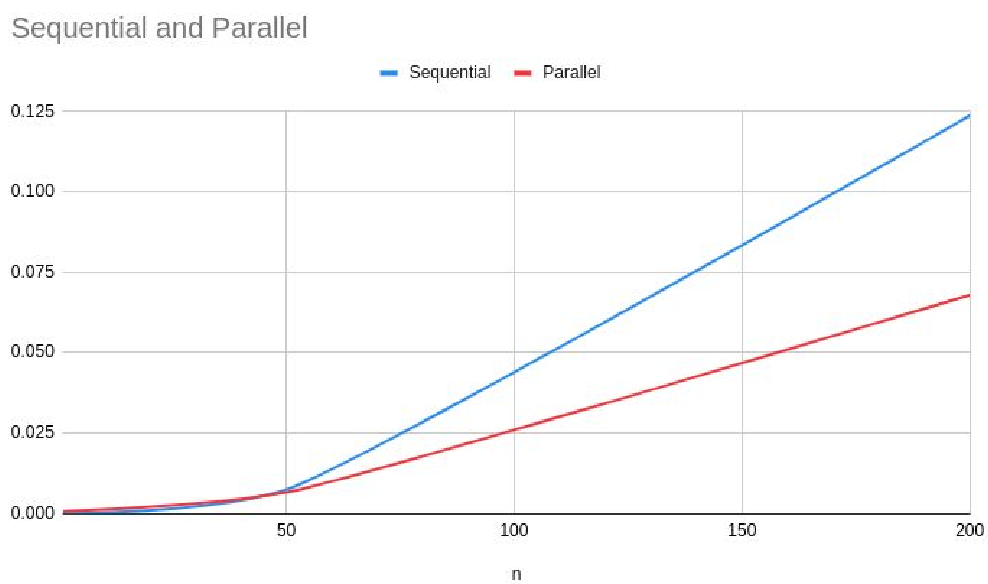
n	Sequential	Parallel
1	0.000000876	0.000609342
50	0.007411576	0.006546092
200	0.123716704	0.067941472
1000	18.11608445	8.088626698
3000	638.5263419	267.6521382

Inferences -

For small matrices (size < 50x50), the sequential program executes faster since the thread creation overhead (time taken to create and join threads) is more than multiplication time. For larger matrices, dividing the work among the cores finishes the multiplication faster.



Graph between time taken and n for n=1 to n=3000



Magnified Above Graph for n=1 to n=200. We can observe that sequential programs run faster for smaller values of n.

Problem 3

Linux Kernel

Kernel compilation and installation

- Linux kernel version - 5.9.1
- After downloading and extracting the source code, I used `make gconfig` (similar to `make menuconfig` but provides GUI) to configure what to include in kernel. I removed some drivers like GPU drivers and manufacturer-specific drivers which are not required because it is running inside a VM.
- Then used `make` and `make modules_install` to compile modules, and `sudo make install` to install the compiled kernel.
- Rebooted into new kernel, and installed `linux-headers-5.9.1` package and dependencies from the Debian repository.

Kernel modules

- Module NULLdereference dereferences a NULL pointer. It causes a kernel oops - an error that the kernel can recover from and the system can continue working.

- Module panic calls the panic() function defined in the kernel libraries. This simply causes a kernel panic - which is a non-recoverable error and freezes the system. The kernel can be configured to reboot in case of a kernel panic, but by default nothing is specified, so the only option in default settings, after a kernel panic is to power off the machine.

Output from sudo dmesg after sudo insmod NULLdereference

```
[ 9847.880983] Loading custom kernel module.
[ 9847.881004] BUG: kernel NULL pointer dereference, address:
0000000000000000
[ 9847.881014] #PF: supervisor read access in kernel mode
[ 9847.881020] #PF: error_code(0x0000) - not-present page
[ 9847.881024] PGD 0 P4D 0
[ 9847.881035] Oops: 0000 [#1] SMP PTI
[ 9847.881046] CPU: 0 PID: 86252 Comm: insmod Tainted: G           OE
5.9.1 #1
[ 9847.881052] Hardware name: innotek GmbH VirtualBox/VirtualBox, BIOS
VirtualBox 12/01/2006
[ 9847.881066] RIP: 0010:init_module+0x15/0x2c [NULLdereference]
[ 9847.881074] Code: Bad RIP value.
```

```
[ 9847.881081] RSP: 0018:ffffad7506397c50 EFLAGS: 00010246
[ 9847.881087] RAX: 000000000000001d RBX: 0000000000000000 RCX:
0000000000000000
[ 9847.881093] RDX: 0000000000000000 RSI: ffff8de3dd418cc0 RDI:
ffff8de3dd418cc0
[ 9847.881097] RBP: ffffad7506397c50 R08: ffff8de3dd418cc0 R09:
0000000000000004
[ 9847.881102] R10: 0000000000000000 R11: 0000000000000001 R12:
ffffffffc08bd000
[ 9847.881107] R13: ffff8de3a79e0070 R14: ffffffffc08bf000 R15:
0000000000000002
[ 9847.881113] FS:  00007f607f79b540(0000) GS:ffff8de3dd400000(0000)
knlGS:0000000000000000
[ 9847.881120] CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
[ 9847.881125] CR2: ffffffffc08bcfeb CR3: 00000000052d4004 CR4:
00000000000706f0
[ 9847.881140] Call Trace:
[ 9847.881160]  do_one_initcall+0x4a/0x1fa
[ 9847.881193]  ? do_init_module+0x28/0x240
[ 9847.881214]  ? kmem_cache_alloc_trace+0x17e/0x2f0
[ 9847.881225]  do_init_module+0x62/0x240
[ 9847.881234]  load_module+0x280c/0x2b40
[ 9847.881248]  __do_sys_finit_module+0xbe/0x120
[ 9847.881255]  ? __do_sys_finit_module+0xbe/0x120
[ 9847.881267]  __x64_sys_finit_module+0x1a/0x20
[ 9847.881291]  do_syscall_64+0x38/0x90
[ 9847.881302]  entry_SYSCALL_64_after_hwframe+0x44/0xa9
[ 9847.881309] RIP: 0033:0x7f607f8e089d
[ 9847.881320] Code: 00 c3 66 2e 0f 1f 84 00 00 00 00 00 90 f3 0f 1e fa 48
89 f8 48 89 f7 48 89 d6 48 89 ca 4d 89 c2 4d 89 c8 4c 8b 4c 24 08 0f 05
<48> 3d 01 f0 ff ff 73 01 c3 48 8b 0d c3 f5 0c 00 f7 d8 64 89 01 48
[ 9847.881326] RSP: 002b:00007ffc31fb00a8 EFLAGS: 00000246 ORIG_RAX:
0000000000000139
[ 9847.881334] RAX: ffffffffffffffda RBX: 000055c84dc177b0 RCX:
00007f607f8e089d
[ 9847.881339] RDX: 0000000000000000 RSI: 000055c84cb38358 RDI:
0000000000000003
[ 9847.881344] RBP: 0000000000000000 R08: 0000000000000000 R09:
00007f607f9b4260
```

```
[ 9847.881349] R10: 0000000000000003 R11: 0000000000000246 R12:
000055c84cb38358
[ 9847.881354] R13: 0000000000000000 R14: 000055c84dc17750 R15:
0000000000000000
[ 9847.881362] Modules linked in: NULLdereference(OE+) btrfs
blake2b_generic xor zstd_compress raid6_pq ufs qnx4 hfsplus hfs minix ntfs
msdos jfs xfs libcrc32c cpuid vboxvideo(OE) nls_iso8859_1 vmwgfx
snd_intel8x0 snd_ac97_codec ac97_bus snd_pcm snd_seq_midi ttm
snd_seq_midi_event intel_rapl_msr drm_kms_helper snd_rawmidi
intel_rapl_common snd_seq cec crct10dif_pclmul snd_seq_device
ghash_clmulni_intel fb_sys_fops aesni_intel snd_timer syscopyarea
crypto_simd snd sysfillrect cryptd glue_helper sysimgblt soundcore rapl
joydev input_leds video mac_hid serio_raw sch_fq_codel parport_pc ppdev lp
drm parport ip_tables x_tables autofs4 hid_generic usbhid hid psmouse
e1000 pata_acpi crc32_pclmul ahci libahci i2c_piix4
[ 9847.881461] CR2: 0000000000000000
[ 9847.881470] ---[ end trace 543e4fc2a480d1d6 ]---
```