CS 307 - System Practicum

Assignment 2 - Group 17 Report

Aaditya Arora (B17071), Anvay Shah (B17078)

Q1. Dining philosophers problem

Problem Statement :

Simulate the Dining Philosophers problem with **5** philosophers and **5** forks. The possible states are

1. **Thinking**
2. **Eating**
3. **Fork Acquired**

Run the solution for **30** minutes, **5** times and compute statistics. Ensure that no deadlock and starvation takes place.

Solution :

We have used semaphores as locks to solve this problem. One semaphore named “state_lock” is used to ensure that only one thread can change the state at a time. Another array of semaphores “forks” is used to ensure that a particular fork can only be used by one thread at a time.

Each philosopher eats for **1** second and thinks for a random time afterwards. We have chosen the random time to be an integer between **1** and **6** seconds. The strategy used for deadlock prevention is as follows. Each philosopher except the 5th picks up their left forks first. The 5th philosopher picks up his right fork first.

Observations :

Number of times each Philosopher ate						
Philosopher Number	**Run 1**	**Run 2**	**Run 3**	**Run 4**	**Run 5**	**Average**
1	404	406	409	415	424	411.6
2	413	404	423	418	413	414.2
3	422	425	412	407	422	417.2
4	420	433	434	421	431	427.8
5	415	418	417	412	419	416.2

From the observation table we can see that there is no starvation of forks - as each philosopher gets to eat roughly an equal number of times. Each time the code ran to completion - thus showing that it did not run into a deadlock.

Q2. Matrix Multiplication

Problem Statement :

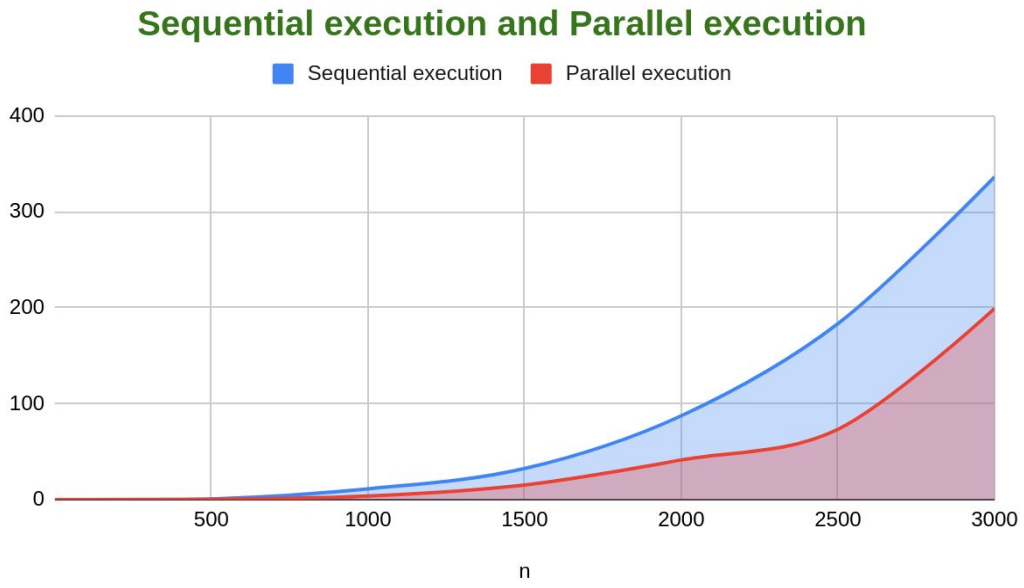
Create two programs for **n**x**n** matrix multiplication - one that sequentially multiplies and one that parallelly uses threads for the same. Compare run times of both programs for values of **n** up to **3000**. Create a graph of input sizes versus run times for both programs.

Solution :

Matrix multiplication uses a triply nested For loop. We are parallelizing the execution of the outermost For loop by creating **8** threads. The loop which was originally supposed to run from **i = 0** to **n-1** is now split into intervals of size **n/8** and each thread runs one of these intervals.

Observations :

Runtimes of both programs vs size of matrix											
n	**1**	**5**	**10**	**50**	**100**	**500**	**1000**	**1500**	**2000**	**2500**	**3000**
Sequential execution	0.001	0.001	0.001	0.003	0.006	0.675	11.213	32.345	87.342	183.32	336.32
Parallel execution	0.002	0.002	0.003	0.005	0.0055	0.350	3.789	15.233	41.234	73.20	199.21



We see that for higher values of **n**, the program with parallel execution provides significantly faster performance. For low values of **n** the difference isn't very noticeable but it becomes more apparent as **n** becomes larger than **500**. We conclude that usage of threads in programs requiring high computation can improve performance. The hyperparameter number of threads must be chosen appropriately as too many threads will cause decrease in performance.

Q3. Compiling the linux kernel from source and building a loadable module

Compiling the linux kernel code took approximately 1 hour for a 4 core machine. The process of booting up and installation was simple. We followed the whole process of installing from this website https://www.cyberciti.biz/tips/compiling-linux-kernel-26.html.

We created 3 linux kernel modules, of which two had runtime errors. As the linux kernel is quite stable, crashing it with runtime error is not possible. But we got logs of **floating point exceptions** in the kern.log file located in '/var/log/kern.log'.

The compilation of linux kernel modules is different than normal c program and for compilation we need to compile with this Makefile.

```
obj-m+=avishkar.o

all:
	make -C /lib/modules/$(shell uname -r)/build/ M=$(PWD) modules
clean:
	make -C /lib/modules/$(shell uname -r)/build/ M=$(PWD) clean
```

The first line of this Makefile is called a goal definition and it defines the module to be built avishkar and the modules target is the default target for external kernel modules.