PART-1

A1:

If Forks[i]== 0 means it’s free and can be used by philosopher else it’s occupied.

Only one philosopher can eat at a time.

No synchronization used

Fork put using Forks[i]==0

A2:

For synchronization and preventing deadlock Semaphore is used

Philosphers are picking forks first from left then from right

Philosphers executed using threads

Forks are defined globally

No two philosphers will pick up same fork and can’t eat with just one fork

Semaphore created using sem_open()

Semaphore deleted using sem_unlink()

One philosopher can eat at a time as there is only one bowl and to prevent deadlock each philosopher can’t access left fork at same time.

No synchronization used

Fork put using Forks[i]==0

If Forks[i]== 0 means it’s free and can be used by philosopher else it’s occupied.

B1:

Without synchronization, two philosphers can eat at a same time.

No two adjacent philosphers can eat at same time

Philosphers executed using threads

Two philosphers can’t pick up same fork and can’t eat with just one fork

Semaphore created using sem_open()

Semaphore deleted using sem_unlink()

Two philosophers can eat at a time as there are two bowls and to prevent deadlock each philosopher can’t access left fork at same time.

Bowl available checked using if Bowl[i]==0

No two adjacent philosphers can eat at same time as they can’t access same fork at same time.

Forks put using sem_post()

B2:

For synchronization and preventing deadlock Semaphore is used

Philosphers are picking forks first from left then from right

Philosphers executed using threads

Two philosphers can’t pick up same fork and can’t eat with just one fork

Semaphore created using sem_open()

Semaphore deleted using sem_unlink()

Two philosophers can eat at a time as there are two bowls and to prevent deadlock each philosopher can’t access left fork at same time.

No two adjacent philosphers can eat at same time as they can’t access same fork at same time.

Forks put using sem_post()

REFERENCE:

https://beej.us/guide/bgipc/html/single/bgipc.html#unixsock

PART-2

FIFO:

Two processes created speak and tick.

Process P1 writes 50 randomly generated strings of size 5 in an array

Process P2 reads 5 strings at a time with their index and prints them to the console

P1 will blick until P2 starts in another terminal.

Opened using: fd = open(FIFO_NAME, O_RDONLY | **O_NDELAY**);

To combine strings strcat is used to make a combined string of format "string1 index string2 index….."

File used is Stringfile which acts as intermediate file between two processes.

REFERENCE:

https://beej.us/guide/bgipc/html/single/bgipc.html#unixsock

https://codereview.stackexchange.com/questions/29198/random-string-generator-in-c

SOCKET:

Two processes created client and server

Process P1 writes 50 randomly generated strings of size 5 in an array

Process P2 reads 5 strings at a time with their index and prints them to the console

Socket(domain, type, protocol) used to create socket

Bind(): used to bind socket to address and port number specified in address.

Listen(): used to wait the client to approach server so that it can make a connection.

To combine strings strcat is used to make a combined string of format "string1 index string2 index….."

REFERENCE:

https://www.geeksforgeeks.org/socket-programming-cc/

https://codereview.stackexchange.com/questions/29198/random-string-generator-in-c

https://beej.us/guide/bgipc/html/single/bgipc.html#unixsock

SHARED_MEMORY:

Used to create a new shared memory object.

Ftok(): for generating a unique key

Signature used: int shm_open (const char *name, int oflag, mode_t mode);

For unlinking: int shm_unlink (const char *name);

Shmget(): used to get identifier for shared memory segment

P1 process is reading the string and printing out in console.

P2 process is generating 50 random strings of size 5 in sending it to a file further read by P1 process.

To combine strings strcat is used to make a combined string of format “string1 index string2 index…..”

REFERENCE:

https://codereview.stackexchange.com/questions/29198/random-string-generator-in-c

https://www.geeksforgeeks.org/ipc-shared-memory/#:~:text=So%2C%20shared%20memory%20provides%20a,to%20generate%20a%20unique%20key.

PART-3

The module file is saved with the .ko extension

Specify licence as MODULE_LICENSE (“GPL”)

Code is executed for loading and unloading (__init and __exit)

Add your kernel object file to obj-m variable

Use the kernel makefile for building the module

To load a kernel module use insmod utility.

-insmod kernel_module.ko

To unload a kernel module use rmmod utility.

-rmmod kernel_module.ko

To list all the modules currently loaded.

-lsmod

To see the output of printk

-dmesg

Reference:

https://linux-kernel-labs.github.io/refs/heads/master/labs/kernel_modules.html

https://www.geeksforgeeks.org/linux-kernel-module-programming-hello-world-program/