**Hungry Philosopher Problem**

**Programing Project2**

**Akshaykumar Parmar(L20529771)**

**Dining philosopher problem:**

The Dining philosopher problem is an illustrative example of a common computing problem in concurrency. It is introduced in 1965. The dining philosopher’s problem describes a group of philosophers sitting at a table doing one of two things eating or thinking. While eating, they are not thinking, and while thinking, they are not eating. The philosophers sit at a circular table each with a bowl of spaghetti/rice. A chopstick/fork is placed in between each philosopher; thus, each philosopher has one chopstick/fork to his or her left and one chopstick to his or her right.

As spaghetti is difficult to serve and eat with a single chopstick, it is assumed that a philosopher must eat

with two chopsticks/forks. The philosopher can only use the fork on his/her immediate left or right. The usage of a semaphore to symbolize a fork is one solution to the dining philosopher problem. A fork can be picked up by performing a wait operation on the semaphore and released by performing a signal operation on the semaphore. Here is the fork structure is mentioned below.

In this program, we could see the below-mentioned file

1)Server File 2)Printer File 3)Fork File 4)Philosopher File

Here we need more than one server as per the suggestion. Here I took 3 machines to execute it.

**Server File**: we have to execute the server golang code to start the server. For that the code is

“go run server.go -n 5 -host sigma25@bmt.lamar.edu:2080”

We can set any port and use it in the multiple place or use in the child process or in print file to track and we can see the machine updates

We must declare 3 struct types for server, child process, and for printing the state of the philosopher

Text

Description automatically generated

We have to manage the server host address and then when we create the child process, we need to add the host id and port followed by go run. There is a function called to read the message from all the child process to show and run on the server.

Here, it has to check for left and right forks to eat . So, initiated two calls here left and right fork to check the address and id of it. We will display the value (1,0) or (1,2) in the server when it’s connected from the child.

**Fork:**

In this file we will manage fork and host id and we need to run the fork below mentioned command

“go run fork.go -id 0 -host sigma25@bmt.lamar.edu:2090 -serve sigma25@bmt.lamar.edu : 2091”

Here I assigned the 2090 port to the server and 2091 port is the fork for the execution.

Text

Description automatically generated

Whenever new fork will be taken it will need id and host again here to check is it free or not.

Here I’ve implemented other function for read and check for new connection as well. In read function it will read and response to the server when new connection has been initiated. Another function does to check the status of the fork with fork id to check it is busy or free. Implemented a condition here based on the response from the fork. If we mention the wrong port, it will mismatch and throw the error.

**Philosopher:**

Here is the philosopher is an important part of this problem. We will be declaring some of the type of the struct for an execution

“go run philosopher.go -id 0 -n 5 -host sigma26@bmt.lamar.edu -manager sigma24.bmt.lamar.edu:2080 -printer sigma24.bmt.lamar.edu:2091”

A screenshot of a computer

Description automatically generated with low confidence Text

Description automatically generated

A function will store the value of the philosopher. It will store the value of both fork and Id and connection details. Two other functions have been created for reading the message through buffer.

When the Register functions have been called, it will take the right and left fork and store the values of it. Tcp will be communicated with the server to go to the left and right fork address and store it in the respected pointer.

Here is the different process is to pick the right fork and check for a left fork. If the left fork is busy, it should drop the right one as well

Text

Description automatically generated

The process is to pick the right fork first and check for a left fork. If the left fork is busy, it should drop the right one as well.

Text

Description automatically generated

**Printer:**

here is the command to execute the printer golang file.

“go run printer.go -n 5 -host sigma24@ bmt.lamar.edu:2091”

Here the printer file will store and execute the number of philosophers and child port number and update of it and return it. There is a function to handle the connection between children and if there would be an error. It will return the error.

Text

Description automatically generated

Main function is to maintain the connection and check for the port. If it is invalid, it will return error or check for listening and handle the connection with the users/forks

First we need to run the server code and then establish the printer to display the connection between forks. Once it’s done, then run the child process in different machines and connect with server.

Once all setup is done and running, it will execute the diners by checking right and left fork. Diners can think or eating or waitng status. With this it will take on the status of the diners and show it on the printer.

OUTPUT:

Graphical user interface

Description automatically generated

Graphical user interface

Description automatically generated

Graphical user interface

Description automatically generated

Graphical user interface

Description automatically generated