ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 9, September 2015

A New Methodology to Avoid Deadlock with Dining Philospher Problem in Rust and Go System Programming Languages

Tata A S K Ishwarya[1], Dr.R.China Appala Naidu[2]

Assistant Professor, IT Department, St. Martin's Engineering College, Hyderabad, Telangana[1]

Professor, CSE Department, St. Martin's Engineering College, Hyderabad, Telangana[2]

Abstract: Operating System is playing a major role now-a-days. Operating System acts as a interface between hardware and user. So when number of processes are increased and resources are in limited state then deadlock occurs .To avoid deadlock we have different approaches like dining – philosopher problem one of the approach to omit deadlock concept. In this paper we focus mainly on obtaining deadlock free solution to the dining – philosopher problem. The solution mainly imposes the restriction that a philosopher may only pick the chopsticks if both of them are available. RUST and GO system programming language are used in the coding part. In this paper we also try to justify why Go is widely used than Rust by providing the values for following details program source code details , CPU Seconds, Elapsed seconds, Memory KB Code B and CPU Load.

Key Terms: RUST, GO, dining – philosopher problem, program source code, CPU Seconds, Elapsed seconds, Memory KB Code B, CPU Load.

1. DINING PHILOSOPHER PROBLEM

1.1 Thought Process Related To Dining Philosopher model

In ancient times, a wealthy King Started a College that had five eminent philosophers. All the philosophers were assigned a work that is related to their profession and in the process of thinking related to work if at all they feel hungry they can have the noodles which is placed on the centre of the rounded table by picking the sliver fork which is towards their left side .

They involved themselves into thought process related to professional work if so they felt hungry they can go to their common dining room sit on the chair labeled by their name and pick their own fork on their left and can start having noodles. But the noodles gets twisted it is necessary to make use of second fork carry it to the mouth.

Then philosopher should make use of fork that is towards his right. Once he is done with eating then he should put down the fork and start thinking. A philosopher can make use of one fork at a time . If another philosopher wants the fork he has to wait no matter how hungry he is.

1.2 Dining Philosopher Problem

Consider five philosophers spending their lives thinking and eating. The five philosophers seated on five different chairs. Bowl of noodles was placed in center and to eat the noodles single chopsticks was given.

When a philosopher gets hungry they try to eat the food by picking nearest chopstick. Since only single chopstick is given they can take their own chopstick. If they want another chopstick they can pick their neighbors one only if they are not using it.

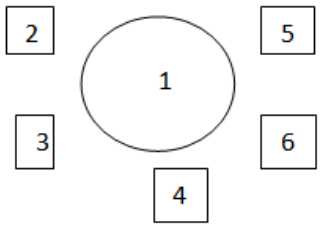

Fig 1.21. Representing Dining Philosopher Model

1- Represents Bowl of Noodles placed on table
2-Represents Philosopher1 with chopsticks in hand
3-Represents Philosopher2 with chopsticks in hand
4-Represents Philosopher3 with chopsticks in hand
5-Represents Philosopher4 with chopsticks in hand
6-Represents Philosopher5 with chopsticks in hand

2. IMPLEMENTATION OF DINING PHILOSOPHER PROBLEM USING RUST

2.1 Rust System Programming Language

Rust was started as a personal project which was developed by Mozilla employee Graydon Hoare . Later in year 2009 Mozilla liked the idea of the project and started sponsoring it and announced about in year 2010 and released pre-alpha version in year 2012 January Rust 1.0. Stable version was released in year 2015 May 15.

Rust is a system programming language that doesn't contain garbage collector and yet it maintains three goals safety, speed and concurrency.
It is useful for use cases embedding and other languages and writing low level code like operating system and device drivers.

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 9, September 2015

It is possible to write programs for space and time requirements. Compile time and runtime overheads are reduced that is we can get the output in very less time.

2.2 Practical Implementation Dining Philosopher Problem using Rust

Coding and the logic technique involved in rust

```
use std::thread;
use std::sync::{Mutex, Arc};
struct T {
   fks: Vec<Mutex<()>>,
}
struct Phil {
   name: String,
   lt: usize,
   rt: usize,
}
impl Phils {
   fn new(name: &str, lt: &usize, rt: &usize) -> Phil {
      Phils {
         name: name.to_string(),
         lt: lt,
         rt: rt,
      }
   }
   fn eat(&self, table: &T) {
      let _lt = table.fks[self.left].lock().unwrap();
      let_rt
= table.fks[self.right].lock().unwrap();
      println!("{} is eating.", self.name);
      thread::sleep_ms(1000);
      println!("{} is Completed eating.", self.name);
   }
}
fn main() {
   let table = Arc::new(T { fks: vec![
      Mutex::new(()),
      Mutex::new(()),
      Mutex::new(()),
      Mutex::new(()),
      Mutex::new(()),
   ]});
   let phils = vec![
         Phil::new("John", 0, 1),
         Phil::new("James", 1, 2),
         Phil::new("Jennifer", 2, 3),
          Phil::new("Franklin", 3, 4),
         Phil::new("Mathew", 0, 4),
      ];

      let handles: Vec<_> =
philosophers.into_iter().map(|p| {
         let t = t.clone();
         thread::spawn(move || {
            p.eat();
         })
      }).collect();
      for h in handles {
         h.join().unwrap();
      }
}
```

When the following code executed in Rust the following output is generated

```
John is Completed eating.
James is Completed eating.
Jennifer is Completed eating.
Franklin is Completed eating.
Mathew is Completed eating.
```

3. IMPLEMENTATION OF DINING PHILOSOPHER PROBLEM USING GO

3.1 Go System Programming Language

Go, also commonly referred to as golang. It is a programming language developed in year 2007 at by three people namely Robert Griesemer, Rob Pike, and Ken Thompson . It is loosely derived from C with additional features like built –in types like key value maps, type safety and large standard library . Several high production were written by using Go and it is very popular at Google.

Example: The server such as Chrome which provides Google binaries for download were rewritten in Go

3.2 Practical Implementation of Dining Philosopher Problem using Go

```
package main
import (
   "fmt"
   "sync"
   "time"
)
var wg sync.WaitGroup
type table struct {
   fks []sync.Mutex
}
type phil struct {
   name  string
   lt  int
   rt int
}
func (p phil) eat(t *table) {
   defer wg.Done()
   t.fks[p.lt].Lock()
   defer t.fks[p.lt].Unlock()
   t.fks[p.rt].Lock()
   defer t.fks[p.rt].Unlock()
   fmt.Println(p.name, "is eating.")
   time.Sleep(1 * time.Second)
   fmt.Println(p.name, "finished eating.")
}
func main() {
   phil := [...]phil{
      phil{"One", 0, 1},
      phil{"Two", 1, 2},
      phil{"Three", 2, 3},
      phil{"Four", 3, 4},
      phil{"Five", 0, 4},
   }
   t := table{fks: make([]sync.Mutex, len(phil))}

   for _, p := range phil{
      wg.Add(1)
```

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 9, September 2015

```
        go p.eat(&t)
    }
    wg.Wait()
}
```

When the following code executed in Go the following output is generated
John is Completed eating.
James is Completed eating.
Jennifer is Completed eating.
Franklin is Completed eating.

4. GO IS BETTER THAN RUST: JUSTIFICATION

4.1 The Total lines of code used for dinning philosopher problem

System Programming Language	Lines of code
Rust	63 Lines of code
Go	50 Lines of Code

Table 4.1.1 The Lines of Code Used in System Programming Language

4.2 We can even Conclude by considering the following data that GO is better than RUST

Program source code	Cpu Seconds	Elapsed Seconds	Memory KB	Code B	CPU LOAD
GO	1.77	1.77	1,668	1237	1%0%1%99%
RUST	3.70	3.70	6,056	1747	1%0%0%100%

Table 4.2.1 comparative data analysis between GO and RUST

With reference Table 4.1.1 and Table 4.2.1 we can conclude that Go is better than RUST

5. CONCLUSION

We have proposed a methodology this methodology identified deadlock well before with the use of Go and RUST. Go and RUST will perform the task in less amount time that is compilation and execution time is reduced very much. We have tried to explore GO and RUST how it is evolved in the current Market. By taking a simple concept like Dining philosopher problem we have tried to explain the way this Go and Rust Works. We have shown how the simple concepts of operating system by using code snippets of RUST and Go and finally we can conclude they are many programming languages which programming users are not aware this programming language makes the life of programmer simpler

REFERENCES

[1] http://userpage.fuberlin.de/~lex/drop/drinking_philosophers.pdf
[2]http://www.researchgate.net/publication/220630692_Application_of_TLRO_to_dining_philosophers_problem
[3] Silberschatz, A., Peterson, J.L.: Operating Systems Concepts. Addison-Wesley, Reading (1988)
[4] Peterson, G.L.: Myths About the Mutual Exclusion Problem. IPL 12(3), 115–116 (1981)
[5] Shavit, N.: Lecture Notes for Lecture 2, Chapter 2.4.1. Tel-Aviv University (2003), http://www.cs.tau.ac.il/~shanir/multiprocessor-synch-2003/
[6] https://en.wikipedia.org/wiki/Rust_(programming_language)
[7]https://en.wikipedia.org/wiki/Comparison_of_programming_languages
[8] https://en.wikipedia.org/wiki/Go_(programming_language)
[9] https://en.wikipedia.org/wiki/Dining_philosophers_problem