**NATIONAL UNIVERSITY OF COMPUTING AND EMERGING SCIENCES**



**Operating system**

**Course code CS-2006**

**Course instructor: Dr Ghufran**

**Project report**

**Project Title: Dining Philosopher Problem**

**Group Members:**

**21K-3345 Seemi Urooj**

**21K-4879 Anfaal Abdul Hai**

# **OBJECTIVE**

The objective is to avoid deadlock condition when the no of philosophers are greater than the no of chopsticks in dining philosophers problem. It is the undesirable condition of concurrent systems. It is marked as in a circular waiting state. In this journal, semaphore used to solve the problem of synchronizing dining philosophers problem. Dining itself is a situation where five philosophers are sitting at the dinner table to eat spaghetti, every philosopher is given a plate of spaghetti and one chopstick to eat spaghetti the two chopsticks are needed to resolve the issue semaphore variable is then applied to each chopstick chopsticks that can be shared all the other philosopher. The problem is to design an efficient distributed deadlock avoidance scheme using wait and signal method that prevents other thread in the chain to make race condition.

Dining Philosophers Problem in OS is a classical synchronization problem in the operating system.

With the presence of more than one process and limited resources in the system the

synchronization problem arises. If one resource is shared between more than one process at the

same time then it can lead to data inconsistency. Consider two processes P1 and P2 executing

simultaneously, while trying to access the same resource R1, this raises the question who will get

the resource and when? This problem is solved using process synchronization. The act of

synchronizing process execution such that no two processes have access to the same associated

data and resources is referred as process synchronization in operating systems. It's particularly

critical in a multi-process system where multiple processes are executing at the same time and

trying to access the very same shared resource or data.

This could lead to discrepancies in data sharing. As a result, modifications implemented by one

process may or may not be reflected when the other processes access the same shared data. The

processes must be synchronized with one another to avoid data inconsistency

Dining Philosophers Problem in OS is a classical synchronization problem in the operating system.

With the presence of more than one process and limited resources in the system the

synchronization problem arises. If one resource is shared between more than one process at the

same time then it can lead to data inconsistency. Consider two processes P1 and P2 executing

simultaneously, while trying to access the same resource R1, this raises the question who will get

the resource and when? This problem is solved using process synchronization. The act of

synchronizing process execution such that no two processes have access to the same associated

data and resources is referred as process synchronization in operating systems. It's particularly

critical in a multi-process system where multiple processes are executing at the same time and

trying to access the very same shared resource or data.

This could lead to discrepancies in data sharing. As a result, modifications implemented by one

process may or may not be reflected when the other processes access the same shared data. The

processes must be synchronized with one another to avoid data inconsistency

Dining Philosophers Problem in OS is a classical synchronization problem in the operating system.

With the presence of more than one process and limited resources in the system the

synchronization problem arises. If one resource is shared between more than one process at the

same time then it can lead to data inconsistency. Consider two processes P1 and P2 executing

simultaneously, while trying to access the same resource R1, this raises the question who will get

the resource and when? This problem is solved using process synchronization. The act of

synchronizing process execution such that no two processes have access to the same associated

data and resources is referred as process synchronization in operating systems. It's particularly

critical in a multi-process system where multiple processes are executing at the same time and

trying to access the very same shared resource or data.

This could lead to discrepancies in data sharing. As a result, modifications implemented by one

process may or may not be reflected when the other processes access the same shared data. The

processes must be synchronized with one another to avoid data inconsistency

Dining Philosophers Problem in OS is a classical synchronization problem in the operating system.

With the presence of more than one process and limited resources in the system the

synchronization problem arises. If one resource is shared between more than one process at the

same time then it can lead to data inconsistency. Consider two processes P1 and P2 executing

simultaneously, while trying to access the same resource R1, this raises the question who will get

the resource and when? This problem is solved using process synchronization. The act of

synchronizing process execution such that no two processes have access to the same associated

data and resources is referred as process synchronization in operating systems. It's particularly

critical in a multi-process system where multiple processes are executing at the same time and

trying to access the very same shared resource or data.

This could lead to discrepancies in data sharing. As a result, modifications implemented by one

process may or may not be reflected when the other processes access the same shared data. The

processes must be synchronized with one another to avoid data inconsistency

Dining Philosophers Problem in OS is a classical synchronization problem in the operating system.

With the presence of more than one process and limited resources in the system the

synchronization problem arises. If one resource is shared between more than one process at the

same time then it can lead to data inconsistency. Consider two processes P1 and P2 executing

simultaneously, while trying to access the same resource R1, this raises the question who will get

the resource and when? This problem is solved using process synchronization. The act of

synchronizing process execution such that no two processes have access to the same associated

data and resources is referred as process synchronization in operating systems. It's particularly

critical in a multi-process system where multiple processes are executing at the same time and

trying to access the very same shared resource or data.

This could lead to discrepancies in data sharing. As a result, modifications implemented by one

process may or may not be reflected when the other processes access the same shared data. The

processes must be synchronized with one another to avoid data inconsistency

Dining Philosophers Problem in OS is a classical synchronization problem in the operating system.

With the presence of more than one process and limited resources in the system the

synchronization problem arises. If one resource is shared between more than one process at the

same time then it can lead to data inconsistency. Consider two processes P1 and P2 executing

simultaneously, while trying to access the same resource R1, this raises the question who will get

the resource and when? This problem is solved using process synchronization. The act of

synchronizing process execution such that no two processes have access to the same associated

data and resources is referred as process synchronization in operating systems. It's particularly

critical in a multi-process system where multiple processes are executing at the same time and

trying to access the very same shared resource or data.

This could lead to discrepancies in data sharing. As a result, modifications implemented by one

process may or may not be reflected when the other processes access the same shared data. The

processes must be synchronized with one another to avoid data inconsistency

Dining Philosophers Problem in OS is a classical synchronization problem in the operating system.

With the presence of more than one process and limited resources in the system the

synchronization problem arises. If one resource is shared between more than one process at the

same time then it can lead to data inconsistency. Consider two processes P1 and P2 executing

simultaneously, while trying to access the same resource R1, this raises the question who will get

the resource and when? This problem is solved using process synchronization. The act of

synchronizing process execution such that no two processes have access to the same associated

data and resources is referred as process synchronization in operating systems. It's particularly

critical in a multi-process system where multiple processes are executing at the same time and

trying to access the very same shared resource or data.

This could lead to discrepancies in data sharing. As a result, modifications implemented by one

process may or may not be reflected when the other processes access the same shared data. The

processes must be synchronized with one another to avoid data inconsistency

Dining Philosophers Problem in OS is a classical synchronization problem in the operating system.

With the presence of more than one process and limited resources in the system the

synchronization problem arises. If one resource is shared between more than one process at the

same time then it can lead to data inconsistency. Consider two processes P1 and P2 executing

simultaneously, while trying to access the same resource R1, this raises the question who will get

the resource and when? This problem is solved using process synchronization. The act of

synchronizing process execution such that no two processes have access to the same associated

data and resources is referred as process synchronization in operating systems. It's particularly

critical in a multi-process system where multiple processes are executing at the same time and

trying to access the very same shared resource or data.

This could lead to discrepancies in data sharing. As a result, modifications implemented by one

process may or may not be reflected when the other processes access the same shared data. The

processes must be synchronized with one another to avoid data inconsistency

Dining Philosophers Problem in OS is a classical synchronization problem in the operating system.

With the presence of more than one process and limited resources in the system the

synchronization problem arises. If one resource is shared between more than one process at the

same time then it can lead to data inconsistency. Consider two processes P1 and P2 executing

simultaneously, while trying to access the same resource R1, this raises the question who will get

the resource and when? This problem is solved using process synchronization. The act of

synchronizing process execution such that no two processes have access to the same associated

data and resources is referred as process synchronization in operating systems. It's particularly

critical in a multi-process system where multiple processes are executing at the same time and

trying to access the very same shared resource or data.

This could lead to discrepancies in data sharing. As a result, modifications implemented by one

process may or may not be reflected when the other processes access the same shared data. The

processes must be synchronized with one another to avoid data inconsistency

1. **INTRODUCTION**

Dining Philosophers Problem in OS is a classical synchronization problem in the operating system. With the presence of more than one process and limited resources in the system the

synchronization problem arises. If one resource is shared between more than one process at the same time then it can lead to inconsistency. Consider two processes P1 and P2 executing simultaneously, while trying to access the same resource R1, this raises the question who will get the resource and when? This problem is solved using process synchronization. The act of synchronizing process execution such that no two processes have access to the same associated resources is referred as process synchronization in operating systems. It's particularly critical in a multi-process system where multiple processes are executing at the same time and trying to access the very same shared resource. This could lead to discrepancies in resource sharing. As a result, modifications implemented by one process may or may not be reflected when the other processes access the same shared resource. The processes must be synchronized with one another to avoid inconsistency.

And the Dining Philosophers Problem is a typical example of limitations in process

synchronization in systems with multiple processes and limited resource.

But before diving deep into the actual problem, we will look at the necessary background for

understanding this problem. We will start with inter-process communication and will make our

way through synchronization before finally looking at the actual problem.

And the Dining Philosophers Problem is a typical example of limitations in process

synchronization in systems with multiple processes and limited resource.

But before diving deep into the actual problem, we will look at the necessary background for

understanding this problem. We will start with inter-process communication and will make our

way through synchronization before finally looking at the actual problem.

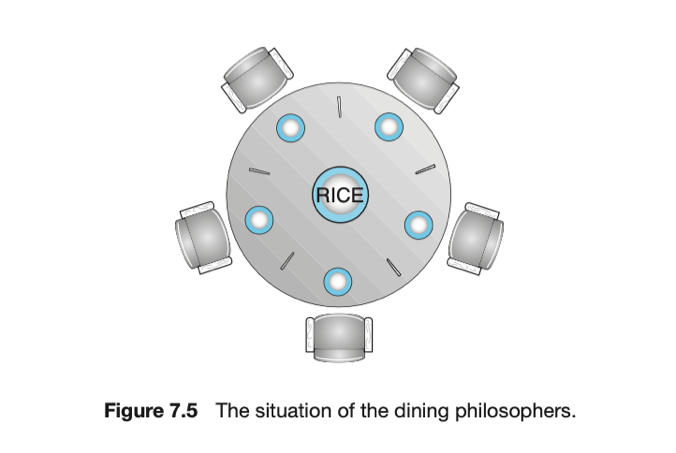
And the Dining Philosophers Problem is a typical example of limitations in process

synchronization in systems with multiple processes and limited resource.

But before diving deep into the actual problem, we will look at the necessary background for

understanding this problem. We will start with inter-process communication and will make our

way through synchronization before finally looking at the actual problem. And the Dining Philosophers Problem is a typical example of limitations in process synchronization in systems with multiple processes(philosophers) and limited resource(chopsticks).



But before diving deep into the actual problem, we will look at the necessary background for understanding this problem. We will start with inter-process communication and will make our way through synchronization before finally looking at the actual problem.

1. **THEORETICAL BACKGROUND**

**3.1 Inter Process Communication**

Inter Process Communication A system can have two types of processes i.e., independent or cooperating. Cooperating processes affect each other and may share data and information among themselves. Inter process Communication or IPC provides a mechanism to exchange data and information across multiple processes, which might be on single or multiple computers connected by a network. There are two main approaches to achieve inter process communication:

* Shared Memory
* Message Passing

**3.1.1 Shared Memory**

Shared memory is a memory shared between two or more processes. Each process has its own address space; if any process wants to communicate with some information from its own address space to other processes, then it is only possible with IPC (inter-process communication) techniques.

To use shared memory, we have to perform two basic steps:

* Request a memory segment that can be shared between processes to the operating system and the processes may read/write on them
* Associate a part of that memory or the whole memory with the address space of the calling process

The drawback of shared memory is that there is no guarantee that the regions will be placed at the same base address

**3.1.2 Message passing**

Message passing provides a mechanism to allow processes to communicate and to synchronize their actions. It is particularly useful in a distributed environment, where the communicating processes may reside on different computers connected by a network. It provides at least two operations:

* send(message)
* receive(message)

Messages sent by a process can be either fixed or variable in size. If only fixed-sized messages can be sent. This restriction, however, makes the task of programming more difficult. Conversely, variable-sized messages require a more complex system-level implementation, but the programming task becomes simpler.

**3.2 Process Synchronization**

A cooperative process is the one which can affect the execution of other process or can be affected by the execution of other process. Such processes need to be synchronized so that their order of execution can be guaranteed. The procedure involved in preserving the appropriate order of execution of cooperative processes is known as Process Synchronization. Inconsistency can occur when various processes share a common source in a system which is why there is a need for process synchronization in the operating system. There can arise some problems while synchronizing processes and we need to eradicate those problems in order to ensure the proper functioning of the system. There are some classical problems of synchronization like the producer-consumer problem and the reader-writer problem. But consider the case when multiple processes try to access a common resource; in this case, we have to decide which process will access the resource and in which order. The dining philosopher’s problem describes this very case.

1. **PLATFORM AND LANGUAGES**

**4.1 Platform**

The project was developed and tested on the following platform:

Operating System: Linux (Ubuntu 16.04 LTS)

Hardware: Intel Core i7, 16GB RAM

**4.2 Programming Languages and Tools:**

The implementation of the operating system utilized the following programming languages and tools.

C: C programming language was used for low-level system programming, including kernel development and device driver implementation.

GNU toolchain: GCC (GNU Compiler Collection) was used for compiling C.

1. **PROBLEM STATEMENT**

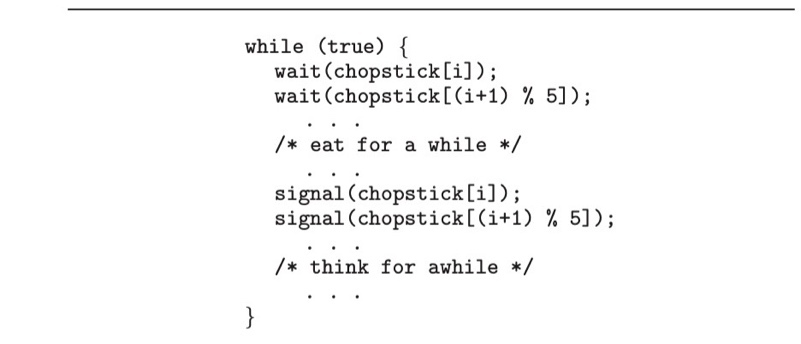
The dining philosopher’s problem states that there are 5 philosophers sharing a circular table and they eat and think alternatively. There is a bowl of noodles for the philosophers and 5 chopsticks. A philosopher needs both their right and left chopstick to eat. A hungry philosopher may only eat if there are both chopsticks available. Otherwise, a philosopher puts down their chopstick and begin thinking again

For example, let’s consider P0, P1, P2, P3, and P4 as the philosophers or processes and C0, C1, C2, C3, and C4 as the 5 chopsticks or resources between each philosopher. Now if P0 wants to eat, both chopstick C0 and C1 must be free, The P1 and P4 void of the resource and the process wouldn't be executed, which indicates there are limited chopsticks (C0, C1...) for multiple philosopher (P0, P1...), and this problem is known as the Dining Philosopher Problem.

**5.1 Solution**

A Simple Solution using Semaphore The solution to the process synchronization problem is Semaphores, A semaphore is an integer used in solving critical sections. The critical section is a segment of the program that allows you to access the shared variables or resources. In a critical section, an atomic action (independently running process) is needed, which means that only single process can run in that section at a time. Semaphore has 2 atomic operations: wait () and signal (). If the value of its input S is positive, the wait () operation decrements, it is used to acquire resource while entry. No operation is done if S is negative or zero. The value of the signal () operation's parameter S is increased, it used to release the resource once critical section is executed at exit.

The wait () operation is implemented when the philosopher is using the resources while the others are thinking. Here, the threads use chopstick[x] and use chopstick [(x + 1) % 5]. After using the chopstick, the signal () operation signifies the philosopher using no resources and thinking. Here, the threads free chopstick[x] and free chopstick [(x + 1) % 5].



The solution we came up with is that no two nearby philosophers can eat at the same time. What if every philosopher sits down about the same time and picks up his left chopstick as shown in the following figure? In this case, all chopsticks are locked and none of the philosophers can successfully lock his right chopstick. As a result, we have a circular waiting (i.e., every philosopher waits for his right chopstick that is currently being locked by his right neighbour), and hence a deadlock occurs. This situation is very rare therefore using semaphore is an effective solution.

1. **Methodology**

The Dining Philosopher Problem states that there are five philosophers which do two think and eat. They share a table having a chair for each one of them. In the centre of the table there is a bowl of rice and the table is laid with 5 single chopsticks.

When a philosopher thinks, he does not interact with others. When he gets hungry, he tries to pick up the two chopsticks that are near to him. For example, philosopher 1 will try to pick chopsticks 1 and 2. But the philosopher can pick up only one chopstick at a time. He cannot take a chopstick that is already in the hands of his neighbour. The philosopher stars to eat when he has both his chopsticks in his hand. After eating the philosopher puts down both the chopsticks and starts to think again.

Represent each chopstick with a semaphore. Each philosopher first picks up the left chopstick and then the right chopstick using the wait () operation each semaphore. After eating he puts down the chopsticks by using the signal () operation on each chopstick. sem\_t chopstick [5] is used to declare 5 semaphore variables, one for each of the five chopsticks. Next, two are the prototypes for functions defined below.

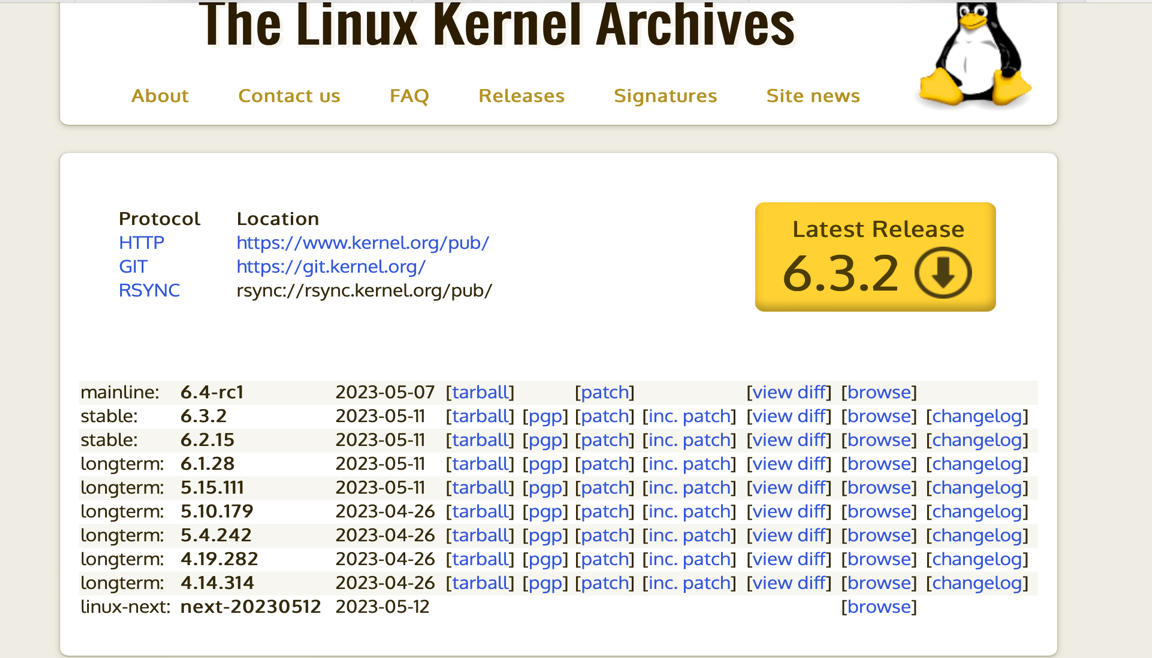
In the main () function there are three for loops. The first loop is used to initialize each semaphore variable with initial value to 1. The second for loop is used to create five threads which will act as the five philosophers. The third for loop uses the pthread\_join function which makes the parent program wait for each of the thread to finish.

Next is the philos function. The philos function when called by each thread receives the same value as the thread number. For example, if thread one runs, the variable ph in the philos function is assigned the value n. This is done because each philosopher n before eating will pick two chopsticks, n and (n+1) %5. Next, the sem\_wait function is used on the left chopstick first[sem\_wait(&chopstick[ph]); If successful, the thread executes the sem\_wait function on the right chopstick sem\_wait(&chopstick[(ph+1) %5]); These two operations are equivalent to picking the left chopstick and then the right chopstick. If both these operations are successful this means that the philosopher is able to pick both the chopsticks and hence will start to eat by calling the eat () function. After eating both the chopsticks are release by using the sem\_post () function.

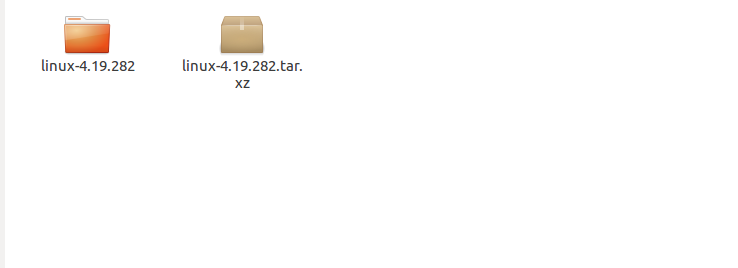
**6.1 Steps followed for the system call**

**1.** Include the following prerequisites

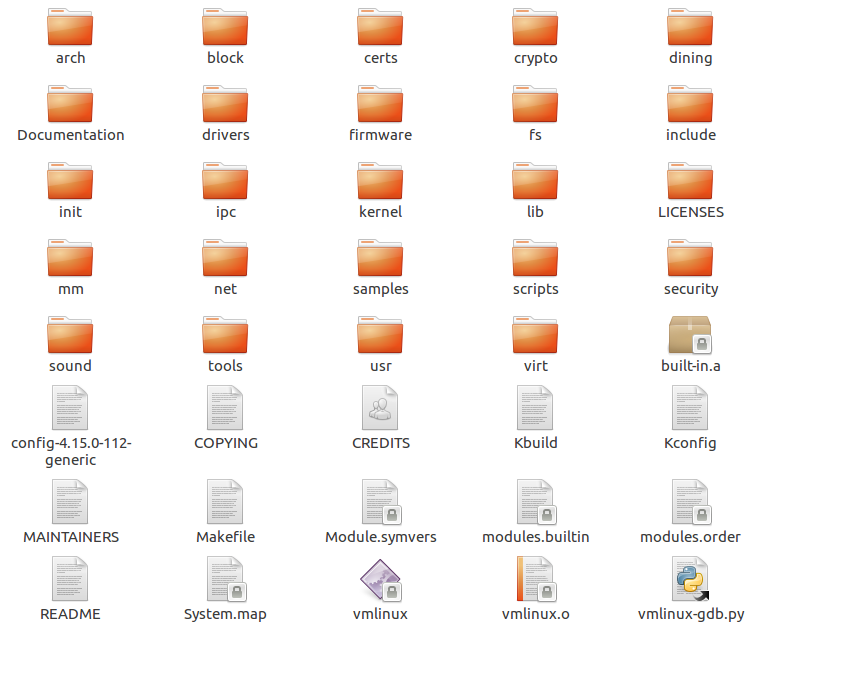
* sudo apt-get install gcc
* sudo apt-get install libncurses5-dev
* sudo apt-get install bison
* sudo apt-get install flex
* sudo apt install make
* sudo apt-get install libssl-dev
* sudo apt-get install libelf-dev
* sudo add-apt-repository "deb http://archive.ubuntu.com/ubuntu $(lsb\_release -sc) main universe"
* sudo apt-get update
* sudo apt-get upgrade
  1. Download kernel 4.19.282 as previous version was 4.15.0-112



* 1. Extract Kernel



* 1. Make a new folder “dining”

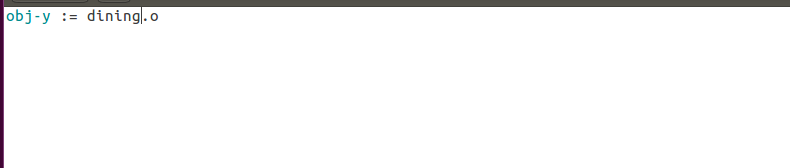


**5.** Move inside the folder dining then right click, and select open in terminal.

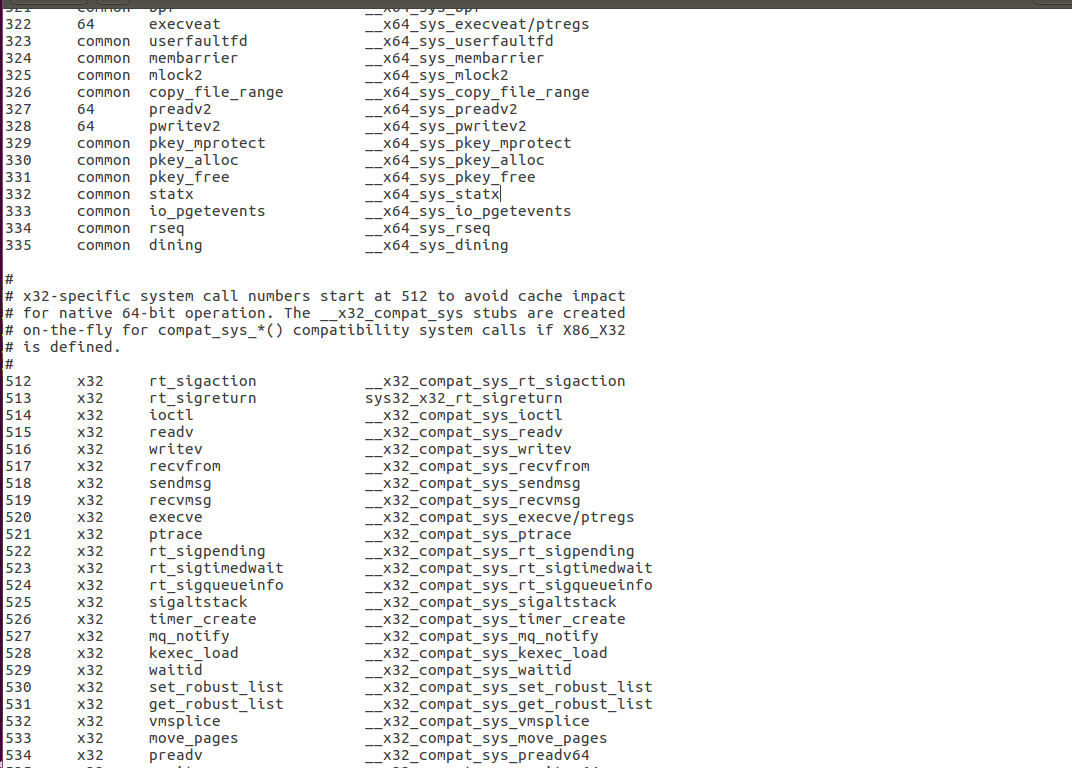
**6.** write “gedit dining.c” on terminal opened. Write the following code in dining.c. Save the file and then close dining.c



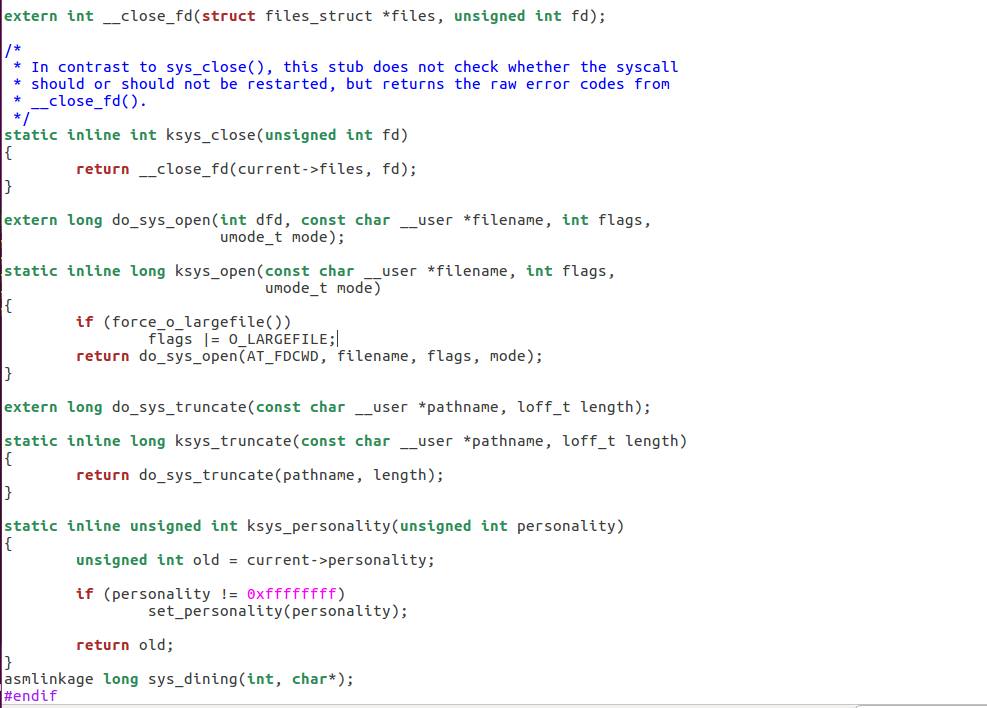
**7**. write “gedit Makefile” on the same terminal and put the following statement in Makefile. Save and close the file.



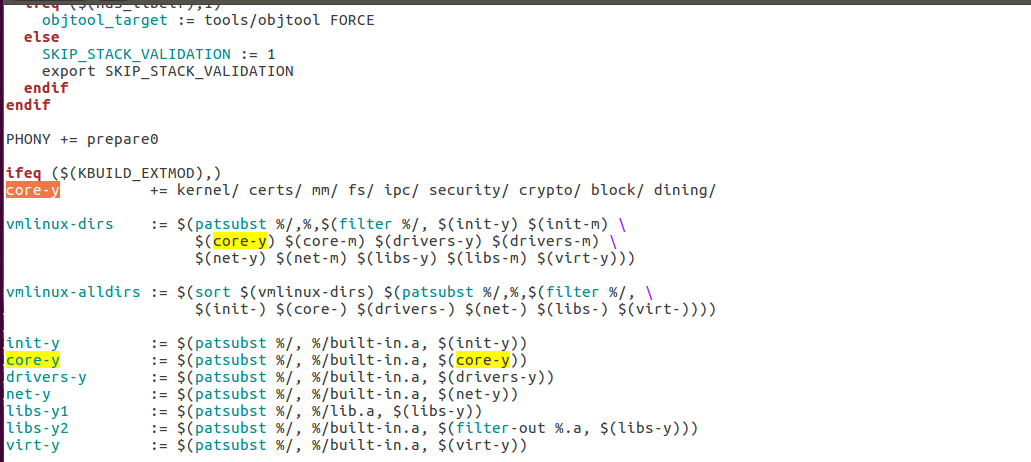
**8**. write on terminal “cd ..” and then write “gedit arch/x86/entry/syscalls/syscall\_64.tbl”. After 334 write the following system call entry. Save and close the file.



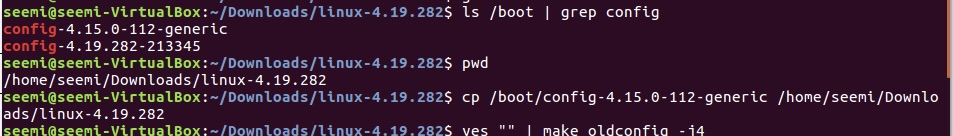
**9.** write “gedit include/linux/syscalls.h” on the terminal and then write the prototype of system call function in this file before #endif at the end of file. Save and close the file.



**10.** write “gedit Makefile” on terminal and in the start of file in extraversion write your roll number without space EXTRAVERSION=-213345 then press control+ F and search core-y in the second occurrence write dining/ as follows. Save and close the file.

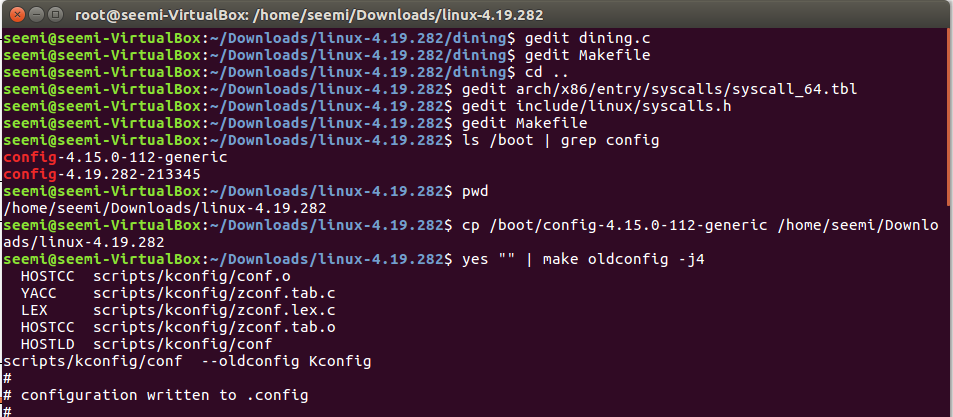


**11.** write the “ls/boot | grep config” on terminal. After this write pwd press enter and then write “cp /boot/config-4.15.0-112-generic” after this paste the pwd path as follows.



**12.** write following commands on terminal one by one and press enter.

* yes "" | make oldconfig -j4
* sudo su
* make clean -j4
* make -j4



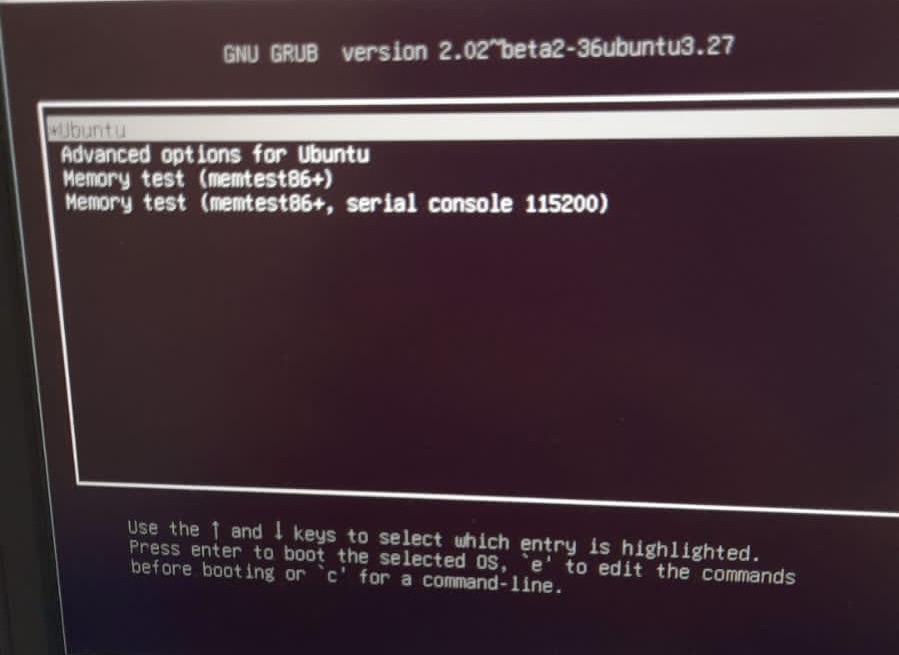


**13.** Now wait the kernel to compile. It will take 1.5 hours.

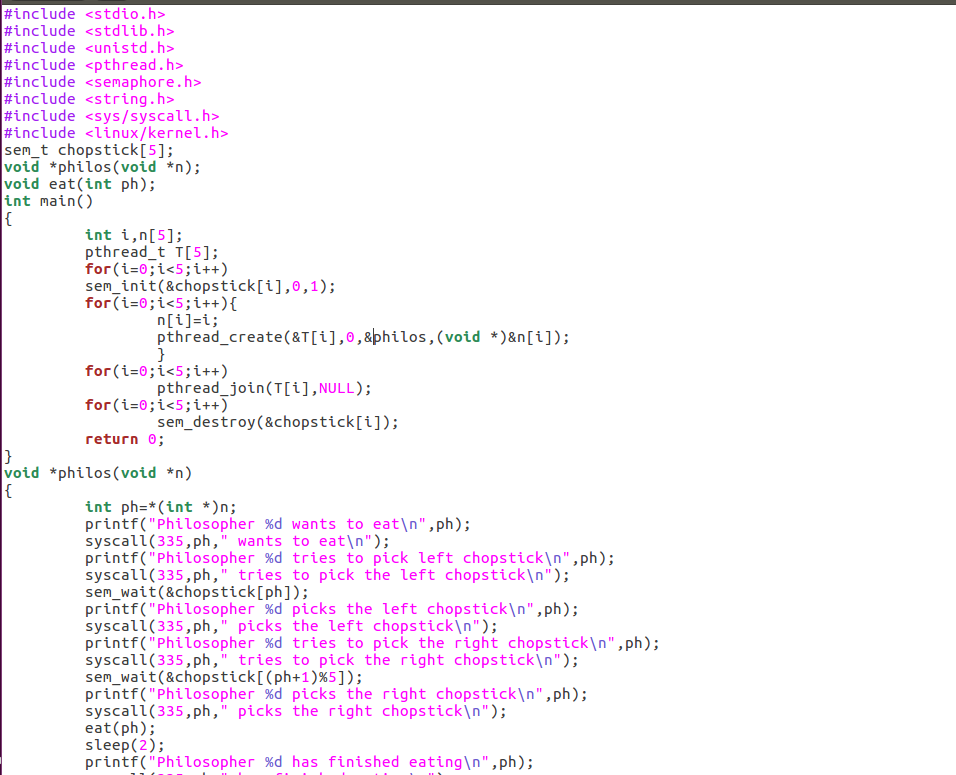
**14**. After kernel compilation write “make modules\_install install” press enter.

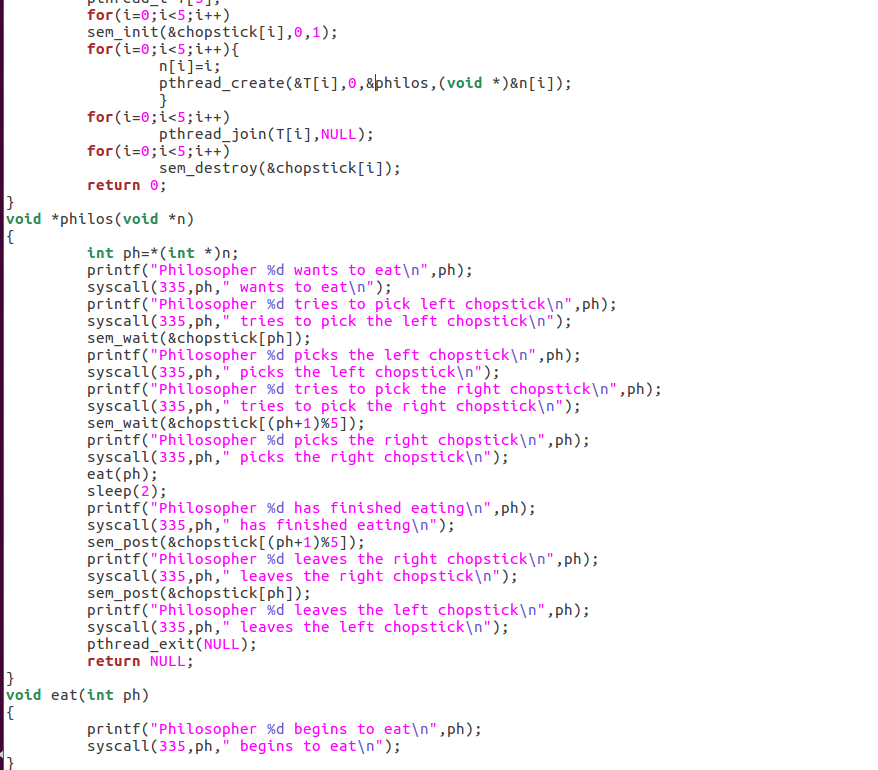
**15.** when it is done write “shutdown -r now” on terminal and immediately press shift

**16.** select advanced options for ubuntu and then select the first option.

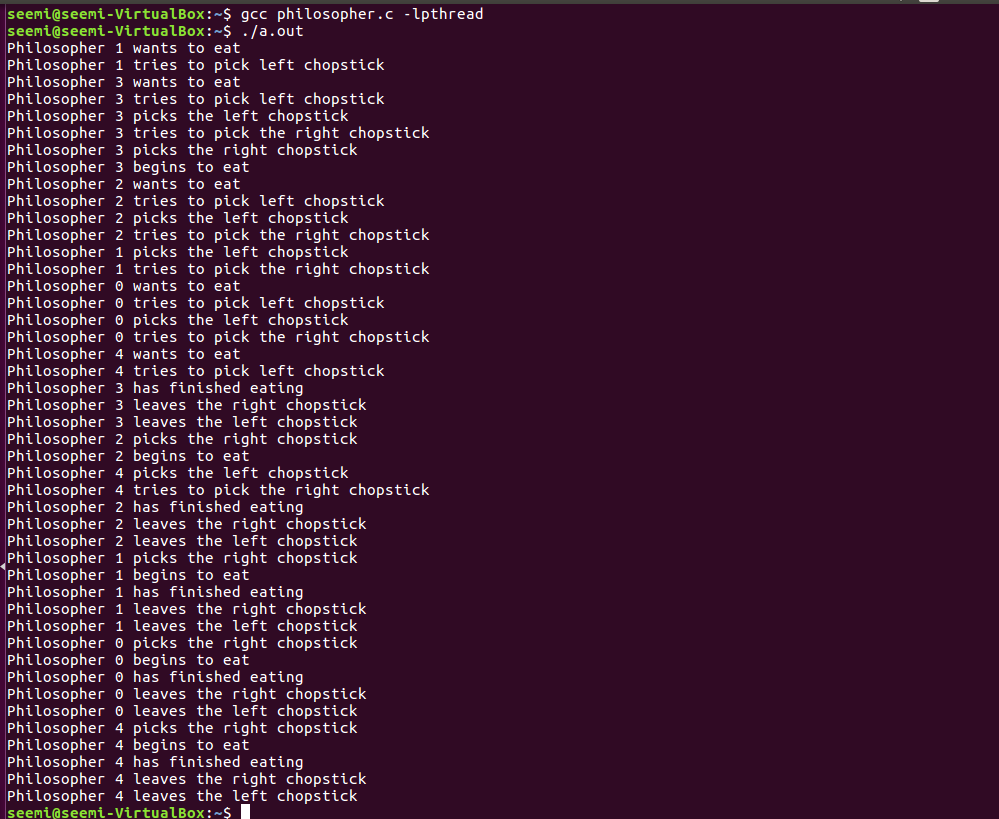


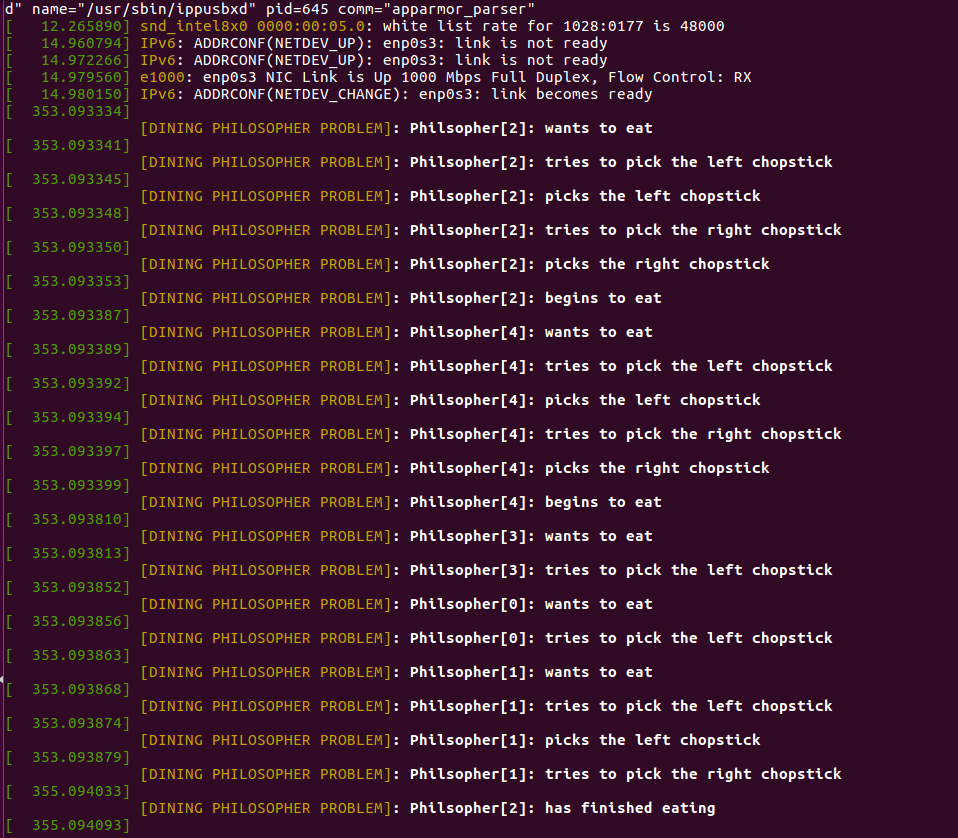
**17.** After log in open terminal write “gedit philosopher.c”. write the following code in file. Save and close the file.

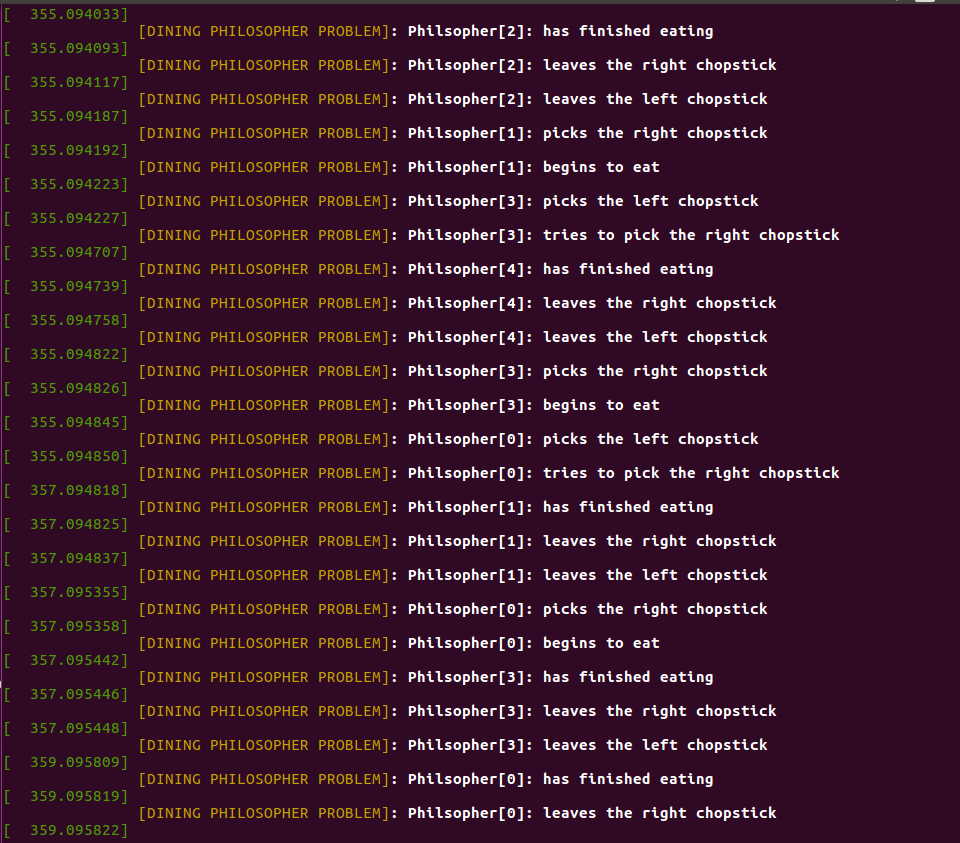




**18.**write gcc “philosopher.c” -lpthread , press enter, write “./a.out “ ,after this write dmesg. Output after running the following commands is.







1. **CONCLUSION**

In conclusion, our solution to the Dining Philosophers problem successfully addressed the challenges of concurrency, resource sharing, and synchronization. By implementing a semaphore, we ensured fair resource allocation, avoided deadlock, and provided a practical solution for multi-threaded systems. This project has deepened our understanding of concurrent programming and the importance of careful design and synchronization strategies in overcoming resource contention issues.

1. **REFERENCES**

* Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). Operating System Concepts. Wiley.
* Stallings, W. (2018). Operating Systems: Internals and Design Principles. Pearson.
* Tanenbaum, A. S., & Bos, H. (2014). Modern Operating Systems. Pearson.
* P. U. Siahaan, "Penyelarasan Pada Masalah Dining Philosophers Menggunakan Algoritma Lock & Release," Techsi, vol. 6, no. 1, pp. 14-18, 2015.