American Journal of Engineering Research (AJER)
e-ISSN: 2320-0847 p-ISSN : 2320-0936
Volume-9, Issue-4, pp-260-271
www.ajer.org
Research Paper Open Access

An Enhanced Dinning Philosophers' Model for Deadlock Situation Awareness using Windows 7 Platform

Onuodu Friday Eleonu[1], Okunka Chinwe Beatrice[2]andNlerum Promise Anebo[3]
[1]Department of Computer Science, University of Port Harcourt, Nigeria
[2]Department of Computer Science, Ignatius Ajuru University of Education, Nigeria
[3]Department of Computer Science and Informatics, Federal University, Otuoke, Nigeria
Corresponding Author: Onuodu, Friday Eleonu

ABSTRACT : *Deadlock is a common problem in multi-processing, parallel computing and operating systems. Most Operating System users are unaware of deadlock when using the computer, and this causes them to be confused and disappointed. In this work, we developed an EnhancedDinningPhilosophers' Model Simulator (EDPMS) for Situation Awareness on Deadlock Occurrence and Avoidance. Secondly, we adopted Soft Computing Methodology and further implemented with JavaScript programming Language and MySQL Relational Database Management System. We also improved the Existing Algorithm on the DinningPhilosophers' Problem, and our results show that the addition of the Windows 7 platform to the Existing Model will tremendously improve Situation Awareness on Deadlock Occurrence and Avoidance, by proffering the "Hold and Wait" concept as solution. This work could be beneficial to Software Developers, to Computer Scientists and to traffic management agencies that requires information on the avoidance of deadlock in traffic control system.*

KEYWORDS:*Deadlock, Dinning Philosopher's Problem, Avoidance, Model*

Date of Submission: 16-04-2020 Date of acceptance: 12-05-2020

I. INTRODUCTION

Most system users are unaware of deadlock when using a system, and this causes them to be confused and disappointed. Deadlock is a common problem in multi-processing, parallel computing and distributed systems. Deadlock can simply be described as a set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set. For instance, if a system has two tape drives, and processes one and two each hold one tape drive, and each process also need the other tape drive that is held by process three.

Furthermore, the problem of deadlock occurs generally in physical events such as traffic congestion and the management of people. In an operating system, a deadlock occurs when a process or thread enters a waiting state because a requested system resource is held by another waiting process, which in turn is waiting for another resource held by another waiting process. If a process is unable to change its state indefinitely because the resources requested by it are being used by another waiting process, then the system is said to be in a deadlock. According to [1] a major illustration of deadlock occurrence is the dinning philosophers' problem which involved five philosophers that sat at a round table with a bowl of spaghetti, and forks placed between each pair of adjacent philosopher. Each philosopher must alternatively think and eat. However, a philosopher can only eat spaghetti when they have both left and right forks. Each fork can be held by only one philosopher that can use the fork only if it is not being used by another philosopher. After an individual philosopher finish eating, they need to put down both forks so that it becomes available to others. A philosopher can take the fork on their right or the one on their left as they become available, but cannot start eating before getting both forks. Most current operating systems cannot prevent deadlocks. When a deadlock occurs, different operating systems respond to them in different non-standard manners.

Under the deadlock detection, deadlocks are allowed to occur. Then the state of the system is examined to detect that a deadlock has occurred and subsequently it is corrected. An algorithm is employed that tracks resource allocation and process states, it rolls back and restarts one or more of the processes in order to remove the detected deadlock. Detecting a deadlock that has already occurred is easily possible since the resources that each process has locked and/or currently requested are known to the resource scheduler of the operating system. One or more processes involved in the deadlock may be aborted. One could choose to abort all competing processes involved in the deadlock. This ensures that deadlock is resolved with certainty and speed. But the expense is high as partial computations will be lost. Or, one could choose to abort one process at a time until the deadlock is resolved. This approach has high overhead because after each abort an algorithm must determine whether the system is still in deadlock. Several factors must be considered while choosing a candidate for termination, such as priority and age of the process. According to the concept of this study, the simplest approach to preventing deadlock is to impose ordering on the condition variables. In the dinning philosopher applet, there is no ordering imposed on the condition variables because the philosophers and the chopsticks are arranged in a circle.

1.1Aim and Objectives of the Study

The aim of this study is to address the problem of deadlock through enhancement of the dinning philosophers' model in order to promote deadlock situation awareness and its avoidance. The specific objectives are to:

i) develop an Enhanced Dinning Philosophers' Model Simulator (EDPMS) for Deadlock Situation Awareness

ii) implement with JavaScript Programming Language, Windows 7 Operating System and MySQL Relational Database Management System as backend.

iii) compare our results with the existing system for deadlock situation awareness

1.2Basics of Operating Systems

An operating system is an integrated set of control programs designed to manage computer resources and maximize the overall operation and effectiveness of the computer system. Examples of operating system include Windows, Linux, UNIX, Mac, etc. To put it in the simplest form; an operating system can also be likened to the managers of resources in the computer system. The computer resources comprises of the hardware, software and all other input/output devices. When the user of the computer system sends a command, the role of the operating system is to ensure that the command is executed; else the user is prompted with a message that explains the error [1].

Even though the operating system does not execute the command or send the error message, it does control the parts of the system that performs such task. Secondly, the operating system performs many functions such as implementing the user interface, sharing hardware among users, allowing users to share data among themselves, preventing users from interfering with one another, scheduling resources among users, and the facilitation of input/output [2] discussed the four major components of an operating system which include the memory manager, processor manager, device manager and file manager.

1.3 Allocation

Akemini [3] stated that it is important to realize that the modality for allocation is dependent on the flexibility of the device. Some devices such as printers and compact disc writers cannot be shared within a process. They must be dedicated to a process. Other devices such as disks and drums may be shared. There will therefore be more flexibility and more complication in their allocation. They may also be needed to convert some devices into virtual devices. The file manager keeps track of information and its location in a file system that includes data files, assemblers, compilers and application programs. In addition, the file manager also decides who makes use of an information, enforces protection requirements and provides necessary accessing routines and access policies such as system only, user only, group only or general access. It also controls the amount of flexibility each user is allowed with a particular file such as read only, read/write only, or the authority to create and/or delete records.

1.4 Process Synchronization

According to Babajide [4] "a process can be defined as a dynamic concept that refers to a program in execution, undergoing frequent state and attribute changes". It is noticeable that process management plays an important role in sequencing a process through its states. For instance, when a user submits his job to the system through an input device, job control statements passes the information to the operating system as to what resources the job will need. The spooling routine reads the job and places it onto a disk in order to find storage space for the machine readable cop of the user's job, the spooling routine must call information management which keeps track of all information and available space in the system.

Furthermore, the job scheduler scans all the spooled files on the disk and picks them to be admitted into the system. In picking a job, the job scheduler will call memory management to determine if sufficient memory is available, and call device management to determine whether devices requested by the job are available. As soon as the job scheduler decides that the job is to be assigned resources, the traffic controller is called to create the associated process information, and the memory management is called to allocate the necessary main storage. The job is then loaded into memory and the process is ready to run. When a processor becomes free, the process scheduler scans the list of ready processes, chooses a process, and assigns a processor to it. If the running process requires access to information, the information management would call and the device management will initiate the reading of the file. The device management will initiate the input/output operation and then call process management to indicate that the process requesting the file is awaiting completion of the wait state. In further analysis of the hierarchical process of the process levels, the function to be placed in the lowest level is that used by all resource managers to keep track of all the allocation of resources. This requires synchronization of the resource allocation. The corresponding primitive operations are frequently called the p-operators. These synchronization primitives operate upon a software mechanism called a Semaphore [5].

1.5 Overview of the Dinning Philosophers' Problem

In computer science, the dinning philosophers' problem is an example often used in concurrent algorithm design to illustrate synchronization issues and techniques for resolving them. It was originally formulated in by Edsger Dijkstra in the year 1965 as a student exam exercise, presented in terms of computers competing for access to tape drive peripherals. Soon after, Tony Hoare gave the problem its present formulation. Five silent philosophers sit at a round table with bowls of spaghetti. Forks are placed between each pair of adjacent philosophers. Each philosopher must alternately think and eat. However, a philosopher can only eat spaghetti when they have both left and right forks. Each fork can be held by only one philosopher and so a philosopher can use the fork only if it is not being used by another philosopher. After an individual philosopher finishes eating, they need to put down both forks so that the forks become available to others. A philosopher can take the fork on their right or the one on their left as they become available, but cannot start eating before getting both forks. Eating is not limited by the remaining amounts of spaghetti or stomach space; an infinite supply and an infinite demand are assumed. The problem is how to design a discipline of behavior (a concurrent algorithm) such that no philosopher will starve; i.e., each can forever continue to alternate between eating and thinking, assuming that no philosopher can know when others may want to eat or think. Figure1.1. Illustrate the picture concept of the dinning philosophers' issue.

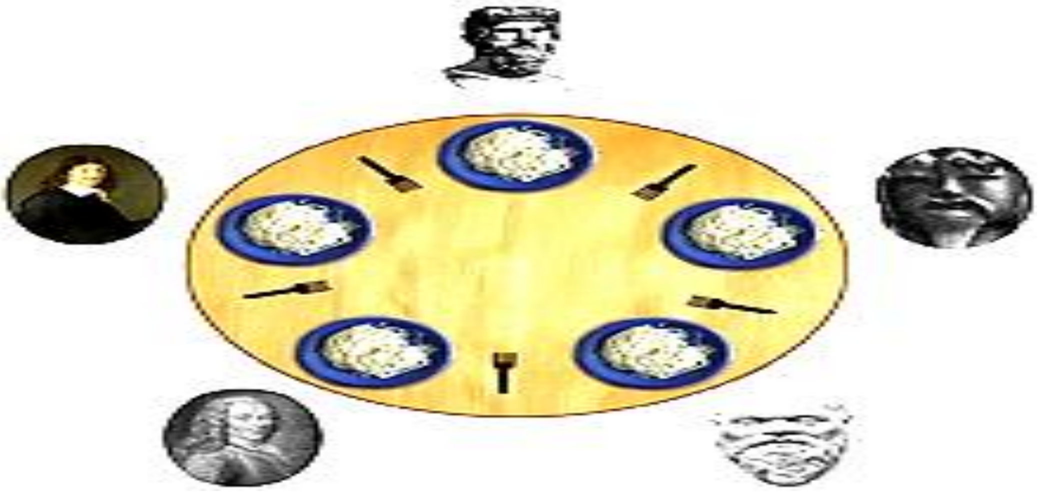


Fig.1.1:Illustration of the Dinning Philosophers' Problem
(**Source:**[5])

From figure 1.1, the problem was designed to illustrate the challenges of avoiding deadlock, a system state in which no progress is possible. To see that a proper solution to this problem is not obvious, consider a proposal in which each philosopher is instructed to behave as follows:

i) think until the left fork is available; when it is, pick it up;
ii) think until the right fork is available; when it is, pick it up;
iii) when both forks are held, eat for a fixed amount of time;
iv) then, put the right fork down;
v) then, put the left fork down;
vi) repeat from the beginning.

This attempted solution fails because it allows the system to reach a deadlock state, in which no progress is possible. This is a state in which each philosopher has picked up the fork to the left, and is waiting for the fork to the right to become available. With the given instructions, this state can be reached, and when it is reached, the philosophers will eternally wait for each other to release a fork. Resource starvation might also occur independently of deadlock if a particular philosopher is unable to acquire both forks because of a timing problem. For example, there might be a rule that the philosophers put down a fork after waiting ten minutes for the other fork to become available and wait a further ten minutes before making their next attempt. This scheme eliminates the possibility of deadlock (the system can always advance to a different state) but still suffers from the problem of live-lock. If all five philosophers appear in the dinning room at exactly the same time and each picks up the left fork at the same time the philosophers will wait ten minutes until they all put their forks down and then wait a further ten minutes before they all pick them up again. Mutual exclusion is the basic idea of the problem; the dinning philosophers create a generic and abstract scenario useful for explaining issues of this type. The failures these philosophers may experience are analogous to the difficulties that arise in real computer programming when multiple programs need exclusive access to shared resources. These issues are studied in concurrent programming. The original problems of Dijkstra were related to external devices like tape drives. However, the difficulties exemplified by the dinning philosophers' problem arise far more often when multiple processes access sets of data that are being updated. Complex systems such as operating system kernels use thousands of locks and synchronizations that require strict adherence to methods and protocols if such problems as deadlock, starvation, and data corruption are to be avoided

II. RELATED WORKS

According to Jinsong et al [6] in their work titled a deadlock prevention using adjacency matrix on dinning philosophers' problem; "the dinning philosopher's problem is summarized as five silent philosophers sitting at a circular table doing one of two things: eating or thinking". While eating, they are not thinking, and while thinking, they are not eating. A large bowl of Spaghetti is placed in the center, which requires two forks to serve and to eat (the problem is therefore sometimes explained using rice and chopsticks rather than spaghetti and forks). A fork is placed in between each pair of adjacent philosophers, and each philosopher may only use the fork to his left and the fork to his right. However, the philosophers do not speak to each other. dinning philosophers' problem is a classic synchronization problem. By the algorithm to limit damage resulting deadlock four necessary conditions, can prevent the occurrence of deadlock. Java language-level support multithreading, the programmer can use Java multithreading deadlock on the dinning philosophers' problem and its prevention study provides a good simulation and verification. The pre-allocation method will cause all resource lower usability. We proposed a solution where it has high efficiency for the system performance and usability in resource. In the same time, [6] recalculated the bound range between upper and lower to increase resource usability. Even though it's a simple and tiny model, but it has still an interesting and valuable issue to multi thread/process topic for computer programming.

Endsley[7] emphasized the importance of the information on deadlock avoidance as the building blocks of the technological tools for Situation Awareness. According to him; numerous approaches have been adopted for the implementation of the mentioned situation awareness models. The blackboard systems have been used to model all levels of situation awareness as defined by his work. The blackboard systems are as a result of language process modeling. Secondly, the blackboard system involves the decomposition of a situation into one or more hierarchical panels of symbolic information often organized as layers of abstraction. Sensory data are encoded by perceptual knowledge sources and posted on appropriate locations of the blackboard, while other knowledge sources reason about the information posted and make inferences about future situation or states, posting all their conclusions back onto the blackboard structure.

Gavin [8] defined deadlock as a state in which each member of a group is waiting for some other member to take action, such as sending a message or more commonly releasing a lock. Deadlock is a common problem in multiprocessing systems, parallel computing, and distributed systems, where software and hardware locks are used to arbitrate shared resources and implement process synchronization.

According to Fraubert [9], "in an operating system, a deadlock occurs when a process or thread enters a waiting state because a requested system resource is held by another waiting process, which in turn is waiting for another resource held by another waiting process". If a process is unable to change its state indefinitely because the resources requested by it are being used by another waiting process, then the system is said to be in a deadlock.

According to Zuhri et al [10], numerous approaches have been adopted for the implementation of the mentioned situation awareness models. The blackboard systems have been used to model all levels of situation awareness as defined by his work. The blackboard systems are as a result of language process modeling. Secondly, the blackboard system involves the decomposition of a situation into one or more hierarchical panels of symbolic information often organized as layers of abstraction.

III. ANALYSIS OF THE EXISTING SYSTEM

An important deadlock avoidance model is the dinning Philosopher Model Simulator (DPMS) as proposed by Zuhri et al [10] According to them; the DPMS is one of the best synchronization models in the avoidance of deadlock in an operating system. The dinning Philosophers problem can be illustrated as follows; there are five philosophers sitting around a table. There are five bowls of noodles in front of each philosopher and one chopstick in between each philosopher. The philosophers spend time thinking (when full) and eating (when hungry). When hungry, the philosopher will take two chopsticks (in the left hand and right hand) and eat. If there are philosophers who took two chopsticks, then the next two philosophers who was eating must wait until chopsticks are put back. It can be implemented with the wait and signal (see figure 3.1).Zuhri et al [10] also promised to modify the DPMS so that after taking chopsticks left, the program checks whether the right chopstick allows being taken. If the right chopstick is not likely to be taken, the philosopher left chopsticks laid back, waiting for some time, later repeating the same process. However, the proposal went wrong due to the fact that, with a bit of bad luck, all the philosophers can start algorithms simultaneously. Furthermore, a unique part of the Existing System where all the programs continue to run in unlimited but no changes/advancements is called starvation. Although the solution ensures that no two neighbors are eating together, but still possible deadlock, i.e., if each philosopher is hungry and took the chopsticks at the left; all grades chopsticks = 0, and then every philosopher that take chopsticks at the right; will result to deadlock. The DPMS further showed several ways to avoid deadlock which include:

i) allowing at most four philosophers to sit together at one table.
ii) allowing a philosopher take chopsticks only if both chopsticks were there.
iii) the usage of asymmetric solutions which implies that philosophers in odd number took a chopstick at the left side, before taking another chopstick at the right side.

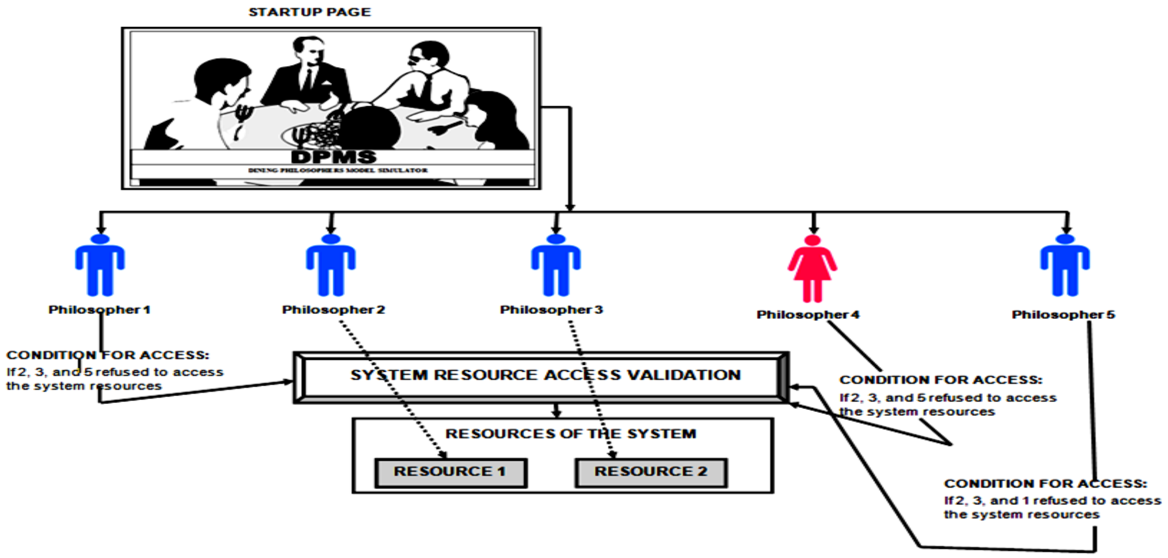

STARTUP PAGE
DPMS
DINING PHILOSOPHERS MODEL SIMULATOR
Philosopher 1
Philosopher 2
Philosopher 3
Philosopher 4
Philosopher 5
CONDITION FOR ACCESS:
If 2, 3, and 5 refused to access the system resources
SYSTEM RESOURCE ACCESS VALIDATION
CONDITION FOR ACCESS:
If 2, 3, and 5 refused to access the system resources
RESOURCES OF THE SYSTEM
RESOURCE 1
RESOURCE 2
CONDITION FOR ACCESS:
If 2, 3, and 1 refused to access the system resources

Fig. 3.1: Structure of the Dinning Philosopher Model Simulator (DPMS)
(Existing System)
(Source: [10])

By the time the philosophers will take the chopsticks in the right hand, there will be deadlock condition since all philosophers will both waiting for chopsticks on the right. The process is said to be experiencing starvation when the processes are waiting for the allocation of resources to infinity, while the other processes can obtain resource allocation. Starvation caused bias in policy or strategy of resource allocation. The basic idea behind the scenario is that; if a concurrent activity always does what seems best for itself or what seems to be the right thing for itself in a shared resources scenario, the result can be chaos. Is there a solution to the dinning Philosopher Problem? The scenario was posed not for a solution but to illustrate a basic problem if the traditional programming approach is applied to concurrent systems. The problem itself crops up in the concurrent systems, and the design decisions should be aware of this, and that is what we have to solve. Any set of concurrent programming techniques that we use is expected at the basic level to offer us features that can be used to deal with the dinning Philosophers problem in some way.

3.1 Description of the Existing System Components

The following Existing System Components are:

i)Startup page:

This component is the welcome screen of the system that is displayed to a user, and also illustrates the importance of the dinning Philosophers' Model Simulator (DPMS)

ii)Group of philosophers:

This component comprises of five philosophers that wants to access the resources of the system. Unfortunately; they all have to pass through the system resource access validation platform with conditions, as there are only two resources available in the system. This also implies that; only two out of the five philosophers can access the system resources at a time.

iii)System resource access validation:

This component involves the validation process that grants access to only two philosophers.

iv)Resources of the system:

This component shows the number of available resources for any philosopher that is granted access by the validation component of the system

3.2 Disadvantages of the Proposed System

The following disadvantages of the Existing System are:

i) difficulty in its implementation on Operating Systems:

The existing system is just a simulator to create awareness on deadlock avoidance, and is yet to be implemented on any operating system platform.

ii) absence of real-time operating system components in its simulations:

The existing system does not use operating system components to simulate deadlock.

IV. ANALYSIS OF THE PROPOSED SYSTEM

This study further aimed at enhancing the Proposed System, and also proposing an Enhanced Dinning Philosophers' Model Simulator (EDPMS). The EDPMS contains a newly added windows 7 operating system platform that will prompt and manage the user on activities that might result to deadlock occurrence (see figure 4.1). Secondly, most system users are unaware of what causes deadlock when using a system, and as a result of that; they are often trapped, confused and disappointed. Thus, the EDPMS enables a user friendly interface that enlightens the user on different application measures for the avoidance of deadlock occurrence. Furthermore, the EDPMS aimed at improving awareness on the avoidance of deadlock occurrence through in-depth analysis of conditions which include: Mutual Exclusion which involved shared resources such as such as read-only files do not lead to deadlocks but resources, such as printers and tape drives, requires exclusive access by a single process, Hold and wait which involved processes to be prevented from holding one or more resources while simultaneously waiting for one or more others.

In addition, the proposed system development also involved feasibility study of different deadlock paradigms. For instance, detecting the possibility of a deadlock before it occurs is much more difficult and is, in fact, generally un-decidable. However, in specific environments, using specific means of locking resources, deadlock detection may be decidable. This is because the system does not require additional prior information regarding the overall potential use of each resource for each process in all cases. In order for the system to detect the deadlock condition it does not need to know all the details of all resources in existence, available and requested. A deadlock detection technique also includes Model checking. This approach constructs a Finite State-model on which it performs a progress analysis and finds all possible terminal sets in the model. Resource allocation strategy for deadlock detection is very liberal due to the fact that resources are granted as requested.

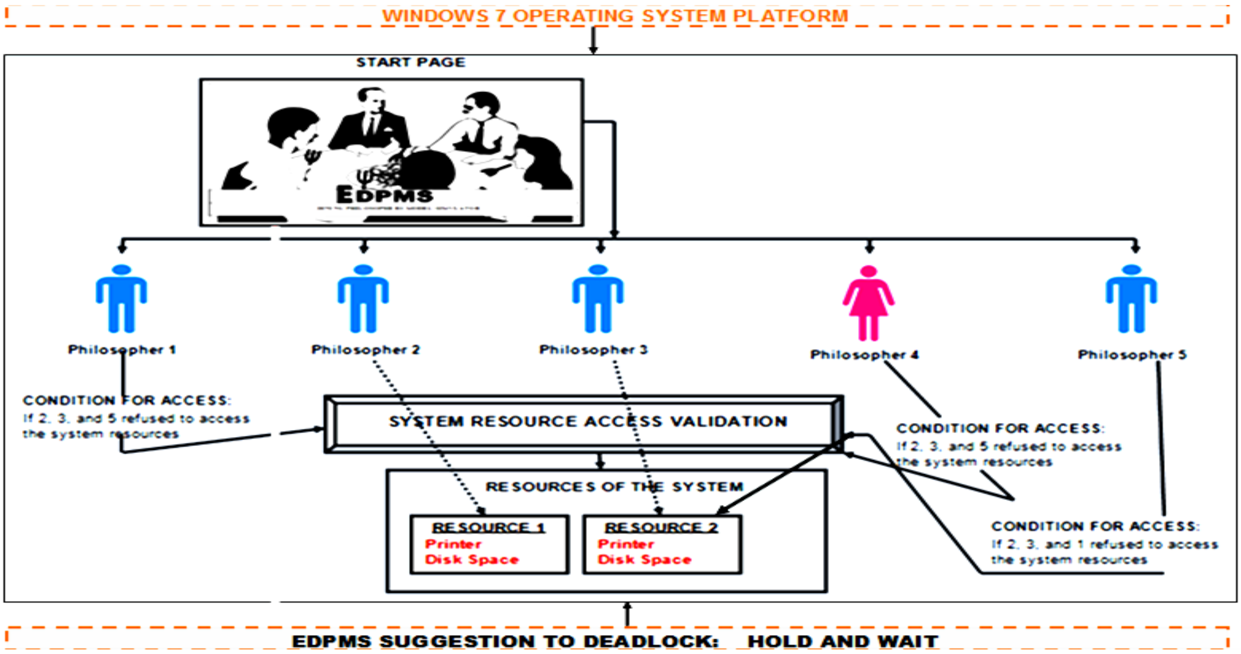

WINDOWS 7 OPERATING SYSTEM PLATFORM
START PAGE
EDPMS
Philosopher 1
Philosopher 2
Philosopher 3
Philosopher 4
Philosopher 5
CONDITION FOR ACCESS:
If 2, 3, and 5 refused to access the system resources
SYSTEM RESOURCE ACCESS VALIDATION
CONDITION FOR ACCESS:
If 2, 3, and 5 refused to access the system resources
RESOURCES OF THE SYSTEM
RESOURCE 1
Printer
Disk Space
RESOURCE 2
Printer
Disk Space
CONDITION FOR ACCESS:
If 2, 3, and 1 refused to access the system resources
EDPMS SUGGESTION TO DEADLOCK: HOLD AND WAIT

Fig.4.1. Proposed System Architecture of the Dinning Philosopher Model Simulator (DPMS) using Windows 7 platform

4.1 Description of the Proposed System Components

The following components of the Proposed System Architecture are:

i)Windows 7 operating system platform:
This component is an embedded system program that manages the resources of the computer system such as memory allocation, drivers, printers, etc. Furthermore, it is the most used system program by computer users.

ii)EDPMS proposed method for handling deadlock occurrence issues:
This is the most important component of the proposed system as it enables the user to implement useful steps in managing the issue of deadlock occurrence. In other words; a confused user that faces the issue of deadlock will be instructed to abort the current process, and as well reboot the operating system.

4.2 Advantages of the Proposed System

The following advantages of the Proposed System include:

i) Awareness on eliminating hold and wait:
The proposed system creates the awareness on allocation of all required resources to the process before start of its execution so as to enable the system user to eliminate the hold and wait condition.

ii) Awareness on shareable and non-shareable system resources:
The proposed system also enables awareness to its users on which resources to be shared and vice-versa.

iii) Ability to provide system users with relevant information on managing the issues of deadlock occurrence:
Recall that most system users are unaware of what causes deadlock when using a system, and as a result of that; they are often trapped, confused and disappointed. Hence, the proposed system always prompts a user on ways for escaping the occurrence of deadlock.

4.3 Existing System Algorithm

STEP 1: Start
STEP 2: Variables Declaration: L AV, UN, PW, FP, SP, TP FOUP, FTP, DPM RES, R: Where L represents the System Login, AV represents Access Validation, UN represents Username, PW represents Password, FP represents the First Philosopher, SP represents the Second Philosopher, TP represents the Third Philosopher, FOUP represents the Fourth philosopher, FTP represents the Fifth Philosopher, RES represents Resources of the system, DPM represents the dinning Philosophers' Model, and R represents results from DPM.
STEP 3: Initiate L
STEP 4: L = AV

STEP 5: AV = UN + PW

STEP 6: Input Details for FP, SP, TP, FOUP, FTP, RES

STEP 7: Implement DPM

STEP 8: Display R of DPM

STEP 9: Stop

4.4Proposed (Modified) System Algorithm

STEP 1: Start

STEP 2: Launch Windows 7 Platform

STEP 3: Click on EDPMS Button

STEP 4: Variables Declaration: L AV, UN, PW, FP, SP, TP FOUP, FTP, DPM RES, R: Where L represents the System Login, AV represents Access Validation, UN represents Username, PW represents Password, FP represents the First Philosopher, SP represents the Second Philosopher, TP represents the Third Philosopher, FOUP represents the Fourth philosopher, FTP represents the Fifth Philosopher, RES represents Resources of the system, DPM represents the dinning Philosophers' Model, and R represents results from DPM.

STEP 5: Initiate L

STEP 6: L = AV

STEP 7: AV = UN + PW

STEP 8: Input Details for FP, SP, TP, FOUP, FTP, RES

STEP 9: Implement DPM

STEP 10: Display R of DPM

Step 11: Stop

V. RESULTS AND DISCUSSION

The EDMS is the most important component of the proposed system as it enables the user to implement useful steps in managing the issue of deadlock occurrence (see figure 5.1). The EDPMS contains a newly added windows 7 operating system platform that will prompt and manage the user on activities that might result to deadlock occurrence. Secondly, most system users are unaware of what causes deadlock when using a system, and as a result of that; they are often trapped, confused and disappointed. Thus, the EDPMS enables a user friendly interface that enlightens the user on different application measures for the avoidance of deadlock occurrence. Furthermore, the EDPMS aimed at improving awareness on the avoidance of deadlock occurrence through in-depth analysis of conditions which include: Mutual Exclusion which involved shared resources such as such as read-only files do not lead to deadlocks but resources, such as printers and tape drives, requires exclusive access by a single process, Hold and wait which involved processes to be prevented from holding one or more resources while simultaneously waiting for one or more others. From figure 5.1, the user has to launch the windows 7 operating system and further click on the EDPMS button to enable the simulator. Recall that the purpose of the simulator is to enlighten the operating system user on the need to be aware of what causes deadlock in an operating system, and the necessary measures to take for its avoidance. Secondly, once the user clicks on the EDPMS button, it navigates to the EDPMS welcome page as further illustrated in

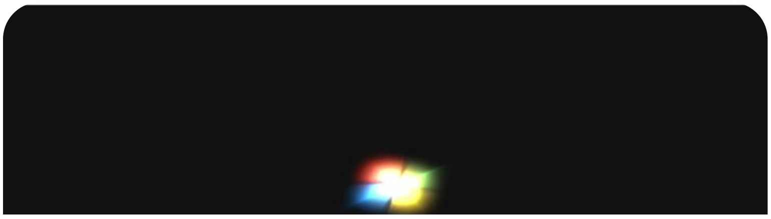
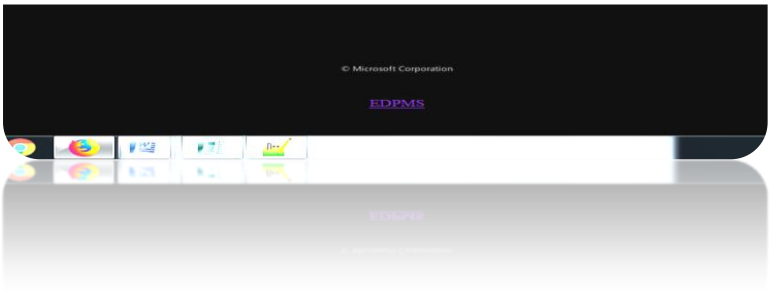


Fig.5.1: Windows 7 Platform



© Microsoft Corporation

EDPMS

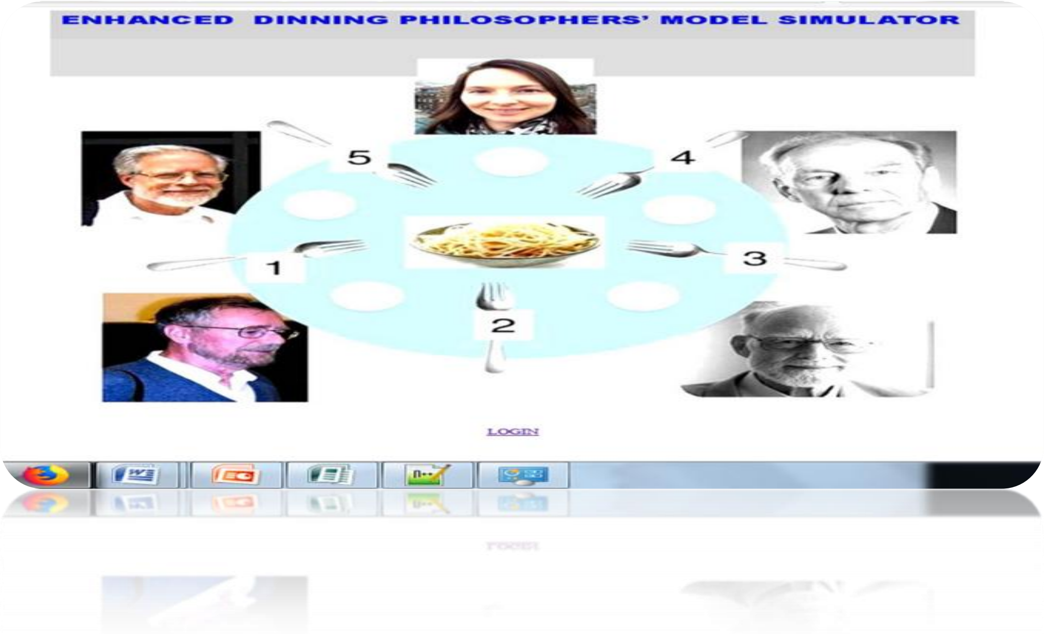
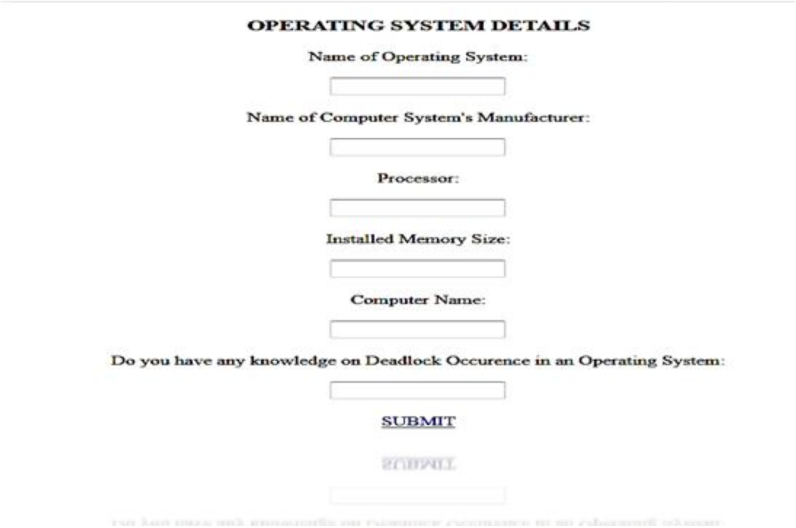


Fig.5.2: Welcome Page

The EDPMS welcome page (figure 5.2) contains a login link designed with JavaScript and styled with cascading style sheet. The login link navigates the user to the EDPMS user details form. Furthermore, we saw the need to implement the login link to EDPMS form due to the fact that in the quest for search engine optimization, a lot of link building is often done with the aim of increasing the amount of valuable inbound links to the webpage in question. This exercise goes to increase the likelihood of that website being ranked high in the search results from search engines. Search engines use link building as a determinant for the ranking of a website, and therefore the higher you want to be, the more work you need to do on link building. Higher ranking is what everyone wishes for because it will definitely increase the number of people visiting their website. The value of links has often been taken for granted because there is no clear measure of their worth. The reason for this is simply the fact that links are everywhere. Links are essentially the threads that hold the whole World Wide Web together. Figure 5.3 is EDPMS user details form contains field such as the name of the operating system used by the user, the name of the computer system's manufacturer, the name of the system processor, the size of the computer's installed memory, the computer system's name, and any knowledge of deadlock occurrence in an operating system. The one thing standing between the user and their goal is a form. Forms remain one of the most important types of interactions for users on the web and in apps. In fact, forms are often considered the final step in the journey of completing their goals. Forms are just a means to an end. Users should be able to complete them



OPERATING SYSTEM DETAILS

Name of Operating System:

Name of Computer System's Manufacturer:

Processor:

Installed Memory Size:

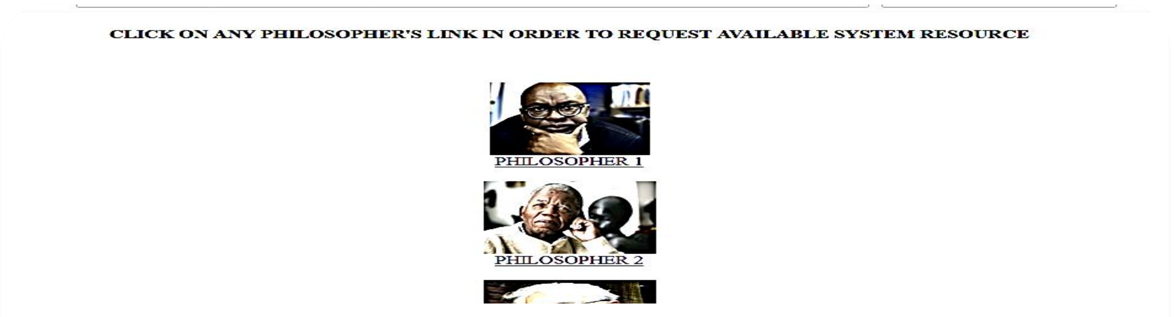
Computer Name:

Do you have any knowledge on Deadlock Occurence in an Operating System:

SUBMIT

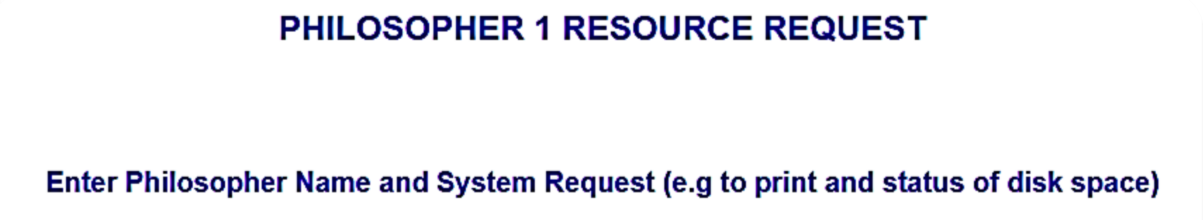
Fig.5.3: Operating Details Page

quickly and without confusion. The user is made to understand the result of their input through feedback. Most apps and websites use plain text as a form of feedback. A message will notify the user about the result and can be positive (indicating that the form was submitted successfully) or negative. Other figures of the EDPMS show the activation of the simulator which involved the behaviors of the philosophers in accessing the available system resources. Since there are only two system resources available for three philosophers at a time, the system only grants access to philosophers 2, 3 and 5. Thus, the EDPMS shows the importance of using the Dinning Philosophers' issue to create awareness on deadlock occurrence and avoidance. A philosopher can take the fork on their right or the one on their left as they become available, but cannot start eating before getting both forks. Eating is not limited by the remaining amounts of spaghetti or stomach space; an infinite supply and an infinite demand are assumed. The problem is how to design a discipline of behavior (a concurrent algorithm) such that no philosopher will starve; i.e., each can forever continue to alternate between eating and thinking, assuming that no philosopher can know when others may want to eat or think. In addition, the proposed system development also involved feasibility study of different deadlock paradigms. For instance, detecting the possibility of a deadlock before it occurs is much more difficult and is, in fact, generally un-decidable. This is because the system does not require additional prior information regarding the overall potential use of each component.



CLICK ON ANY PHILOSOPHER'S LINK IN ORDER TO REQUEST AVAILABLE SYSTEM RESOURCE

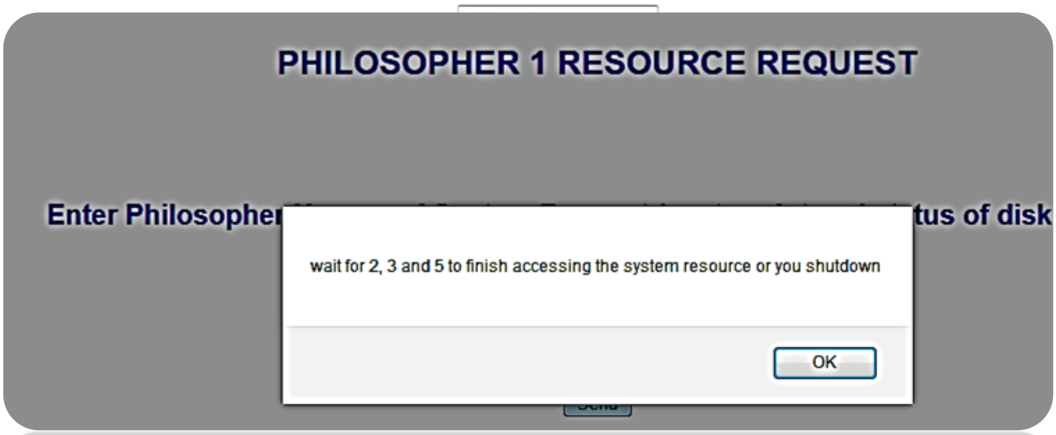
Fig.5.4: Operating System Details Page



PHILOSOPHER 1 RESOURCE REQUEST

Enter Philosopher Name and System Request (e.g to print and status of disk space)

Fig.5.5: Philosopher 1 Request Page



PHILOSOPHER 1 RESOURCE REQUEST

Enter Philosopher ... tus of disk

wait for 2, 3 and 5 to finish accessing the system resource or you shutdown

Fig.5.6: System output to Philosopher 1

Table 5.1: Online Survey Result on Deadlock occurrence and avoidance using the Dinning Philosophers' Problem

ALGORITHM	RESEARCH AREA OF DEPLOYMENT	USERS' FEEDBACK ON ALGORITHM EFFICIENCY (%)
Zuhri et al (2018)	A Deadlock Prevention Using Adjacency Matrix on Dinning Philosophers Problem	82
Proposed System Algorithm (2019)	An Enhanced Dinning Philosophers' Model for Situation Awareness using Deadlock Paradigm	95

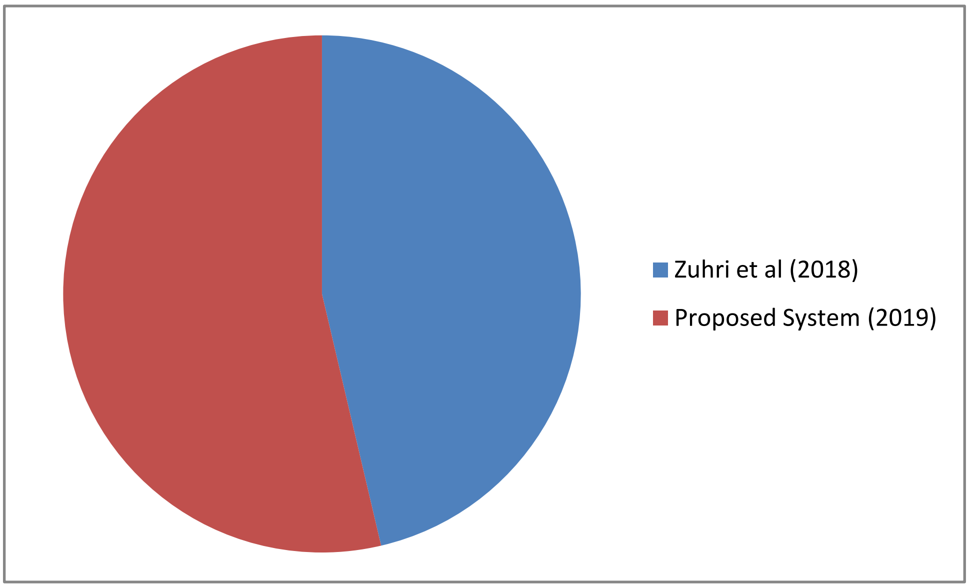


Fig.5.7: Performance Evaluation Chart of Algorithm against Users' Feedback on Algorithm Efficiency

VI. CONCLUSION AND FUTURE WORK

In this study, we have developed an Enhanced Dinning Philosophers' Model for deadlock situation awareness in operating systems. The dinning philosophers' problem is an example often used in concurrent algorithm design to illustrate synchronization issues and techniques for resolving them. It was originally formulated in by Edsger Dijkstra as a student exam exercise, presented in terms of computers competing for access to tape drive peripherals. In addition, we intend to propose the development of a mobile EDPMS application for situation Awareness on deadlock occurrence and Avoidance, which would be of immense benefits to Smartphones users.

REFERENCES

[1]. Nwachukwu, E. O.,Onuodu, F. E.: Principles of Operating System, an International Textbook published to the Department of Computer Science, (2012)

[2]. Greggs, H.: Review of the issues surrounding the adoption of the dinning philosophers' Issues in the avoidance of deadlock, International Journal on Trends and Technology (IJTT), 3(4), 201 – 217, (2016)

[3]. Akemini, N.: A preliminary study on the components of an Operating System, International Journal of Computer Applications (IJCA), 2(8), 411 – 432, (2017)

[4]. Babajide, K.:The review of the context, concept and terminologies of Windows Operating Systems, International Journal of Engineering Research and Applications (IJERA), 2(3),402 – 410, (2016)

[5]. Fred, G.: An Enhanced Survey on the Dinning Philosophers' Problem, International Journal of Engineering Technology (IJET), 7(8), 23 – 29, (2017)

[6]. Jinsong, O.,Halorde, G., Melody, M.:A deadlock prevention using adjacency matrix on Dinning Philosophers' Problem, Applied Mechanics and Materials, 10(18), 121 – 126, (2018)

[7]. Endsley, H.: Review of the issues surrounding the adoption of the dinning philosophers' Issues in the avoidance of deadlock, International Journal on Trends and Technology (IJTT), 3(4), 201 – 217, (2018)

[8]. Gavin, L.: the concepts of Operating Systems for the transactions on programming in Systems, International Journal of Software Research (IJSR), 6(1984), 632 – 646, (2017)

[9]. Fraubert, J.: achieving system programming anonymity privacy protection using generalization and Suppression, International Journal of advances in Engineering and Technology, (IJAET), 3(1), 315 – 321, (2017)

[10]. Zuhri, F., Palmas, O., Vredi, G.:Dinning Philosophers' Model for Deadlock Awareness, International Journal of Engineering Technology (IJET), 9(6), 13 – 19 (2018).

Onuodu, Friday Eleonu,et. al. "An Enhanced Dinning Philosophers' Model for Deadlock Situation Awareness using Windows 7 Platform." *American Journal of Engineering Research (AJER)*, vol. 9(04), 2020, pp. 260-271.