Operating system lab Project

Report file

Com312

Project Group 7

Ji

Submitted to : Mr. Saurabh

Sharma Professor

Submitted by: Ayush Pandita -2021a1r174

Rahul Sharma -2021a1r171

Sourav salaria-2021a1r169

Ravinder Singh bogal-

2021a1r176

Sem - 3rd

Lab group - 6

Project

Simulate the prediction of deadlock in operating system when all processes announce their resource requirement in advance

Model institute of engineering and technology ,Jammu. Branch - CSE
Name

email's- 2021a1r174@mietjammu.in
2021a1r169@mietjammu.in , 2021a1r171@mietjammu.in ,
2021a1r176@mietjammu.in

Abstract: - This paper presents a system proposal for preventing, detecting and avoiding deadlocks (*PDADL*), the system applies some common methods to prevent, detect and avoid deadlock on a local operating system. Moreover, the system simulates all the process and resources on a real operating system. The resources and processes can be detected dynamically from different operations running by real application programs through the operating system. We use Matrix-based algorithm for detecting the deadlock, and Banker's algorithm as well. In addition we made some modifications on the use of avoiding deadlocks in term of, eliminating the circular wait condition which is currently used for preventing deadlocks. We implement an application program named as PDADL to simulate real processes and resources from running operations in a local operating system automatically.

Keywords:- Deadlock, methods in operating system, detecting deadlocks, preventing deadlock, avoiding deadlocks, processes and recourses.

1. Introduction

It's known that, computer is the most significant device nowadays, whereas, it contributed in all life-fields [1]. This device is operated by a program called operating system, which considered being the main program on the system platform and any problem may infect this program, it directly causes a great costs and losses. One of these problems that may cause the loss is called a deadlock [3].

Deadlock is a potential problem in any operating system. Deadlock is a specific condition where two or more processes are each waiting for the other process to release a resource. Deadlock is a common problem in multithreaded environments where many processes share a mutually exclusive resource known as a lock. It also occurs when a group of processes each have, granted exclusive access to some resources and each one wants to get another resource that belongs to another process in the group. All of the processes are blocked and non will ever run again [5].

2. Related work

Coffman, Elphick and Shoshani (1971) indicated in [2], lists the following conditions were are necessary and sufficient for causing a deadlock, these conditions are stated as follow:

1) Resources must be held in mutual exclusion.
2) Processes cannot be preempted from held resources.
3) Processes hold resources while waiting to acquire others.
4) A circular chain of processes exists in which each process is requesting and waiting for a resource held by the next process in the chain.

All four conditions must be present for a deadlock to occur. If one of them is absent, no possible deadlock occurs [4].

3. Implementation

There are two major functions implemented in the PDADL system, these functions are pointed as the following:

1- Detecting and Avoiding Deadlock procedure

This procedure is divided into two sub procedures:

The resources allocation procedure:

This procedure is to ensure allocating the resources correctly, whereas is not exceeding the existing resources, containing the following:

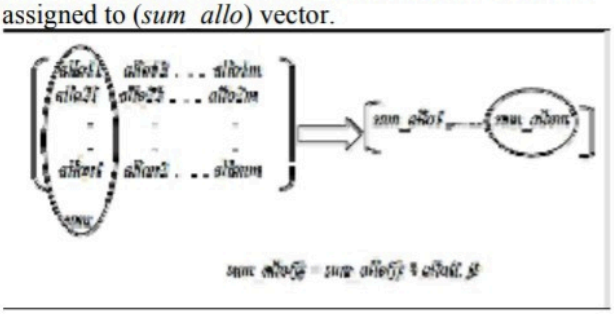
(i) Reading the number of processes (*n*), the number of resource classes (*m*).
(ii) Reading the number of instances of each class is detected as a vector (*E*).
(iii) Representing the existing resources, with E1 resources of class 1, *E2*.
(iv) Resources of class 2, and generally, *Ei* resources of class i ($1 \le i \le m$).

$$(E1, E2 \ldots\ldots Em)$$

The existing resources vector could be known from the number of current resources held by processes which will be presented as a matrix (*alloij*). This matrix represents the current allocated resources, where ($1 \le i \le m$) and ($1 \le j \le n$). Thus (*alloij*) is the number of instances of resource *j* that are allocated for process *i* [4].

$$\begin{array}{l} \\ \text{Process1} \\ \text{Process2} \\ . \\ . \\ \text{Process } n \end{array} \begin{bmatrix} R1 & R2 & \ldots & Rm \\ allo11 & allo12 & \ldots & allo1m \\ allo21 & allo22 & \ldots & allo2m \\ . & . & & . \\ . & . & & . \\ allon1 & allon2 & \ldots & allonm \end{bmatrix}$$

The current allocation matrix is the sum of instances for each resource class in the current allocated resources matrix which are calculated and assigned to (*sum_allo*) vector.



The comparison between the sum of *allo* column *(sum_allo*) vector and (*E*) vector will be assigned, if one element of (*sum_allo*) vector is greater than the corresponding element of (*E*) vector, whereas a clause shows an error message arise during allocating the resources, i.e. The number of resource instances were allocated has to be exceeded in the existing resources [4].

$$\begin{bmatrix} sum\_allo1, \ldots\ldots, sum\_allom \end{bmatrix} \quad \text{Comparison} \quad \begin{bmatrix} exist1, \ldots\ldots, existm \end{bmatrix}$$

If $sum\_allo[i] > exist[i]$ then "error message arise during allocating the resources "

This comparison tolerates the using of all resources exists in the system. On the opposite side, the original algorithm uses a matrix that represents the greatest number of resources used by the process.

The needed resources matrix is the number of resources requested by processes and entered as a matrix (*needij*). It represents the resources need by the system, while ($1 \le i \le m$) and ($1 \le j \le n$). Thus need*ij* is the number of instances of resource *j* that are needed by process *i*.

$$\begin{array}{l} \\ \text{Process1} \\ \text{Process2} \\ \\ \text{Process } n \end{array} \begin{bmatrix} R1 & R2 & \ldots & Rm \\ need11 & need12 & \ldots & need1m \\ need21 & need22 & \ldots & need2m \\ & & & \\ needn1 & needn2 & \ldots & neednm \end{bmatrix}$$

The safety procedure:

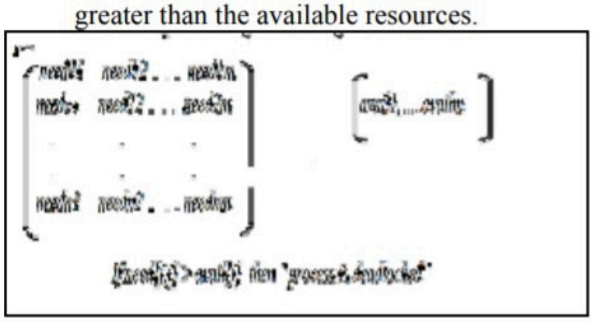
This procedure intended to test the system if it's protected from deadlocks or not, by detecting some processes and checking the ability of avoiding the deadlock via applying the following progresses:

- The available resources that may be allocated for the processes are calculated by subtracting the currently allocated resources from the existing resources.

$$Avail[i] = exist[i] - sum\_allo[i]$$

$$\begin{bmatrix} avail1, \ldots\ldots availm \end{bmatrix} = \begin{bmatrix} exist1, \ldots\ldots, existm \end{bmatrix} - \begin{bmatrix} sum\_allo1, \ldots\ldots, sum\_allom \end{bmatrix}$$

- The comparison between the needed resources for each process, with the availability of the resources. In case of being greater then the need, the clause will view a message conforming that the process is in a deadlock state. This step considered to be the starting state of detecting deadlock, whereas a clause shows that the need is greater than the available resources.



- Next step is to check the next process by testing the needed resources with the available resources. If it is greater than the available, it means that, this process is facing a deadlock state, therefore the algorithm will be running continuously until it reaches a process needs with a less than or an equal to the available resources.
- In the case of, a waiting state of all the process to be released from there resources. It should be checked if the system is fully deadlocked or not?, but if it is the case then there is no way to resolve the problem, otherwise it could follow the next step.
- When the process of avoiding is done from previous step; by releasing the allocated resources and adding them into the available resources vector this case occurs. The process will be satisfied, whereas the clause will declare the start of avoiding, and the clause for each avoiding state as well.

  if need[i,j] < avail[j] then

  "avoidance at this process "

  $avail[i] = avail[i] + allo[i,j]$

- After testing all resources and all processes, a clause will send back a message which conforms the ending of each cycle.

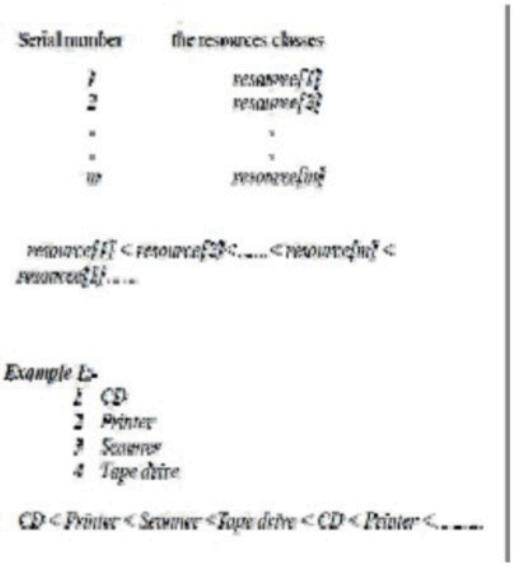
- The next cycle will be started and the system will be running until the deadlock is avoided in all running processes in the system. It ensures that all allocated processes are free, and the clause is going to view a message which conforms, the system finally is in a safe state.

2- Preventing Deadlock

This part intends to prevent the deadlock by eliminating the condition of cycle waiting by placing the resources of the system into a numerical system priority. The major steps in this part are as follows:

(i) Reading the number of resources classes (*m*) in the system.

(ii) Entering the resource classes according to circular numerical order by using a matrix storing the names of resources in a circular serial form.



Example 1:-

1. *CD*
2. *Printer*
3. *Scanner*
4. *Tape drive*

CD < Printer < Scanner < Tape drive < CD < Printer <. . . .

(iii) Allocating the resources of the running processes in the first time from the matrix. The clause will view a message conforming that all the resources are allocated for each process.

Example 2:-

(a) The resource "*Printer*" is allocated for process 1 at the first time.
(b) The resource "*Scanner*" is allocated for process 2 at the first time.
(c) The resource "*CD*" is allocated for process 3 at the first time.
(d) The resource "*Tape drive*" is allocated for process 4 at the first time.

(iv) The storing matrix (*p*) consists of the last resources allocated to every process. By this matrix, the tracks of the last allocated resources are kept. In example 2, the matrix

(p) holds the last allocated resources and it is as follow:

	Process1	process2	process3	process4	
P	*[*	*2*	*3*	*1*	*4*	*]*

(v) In the end of holding the resource during the process, the resource is allocated once again, but the allocating must be in an increasing order, whereas the process can't hold a resource less than the last allocation. The serial number of the new resource must be greater than the serial number of the old resource that was stored in the matrix (*p*).

According to the example 2:

- Process1 can hold the resources number 3 or 4 *(Scanner or Tape drive*).
- Process2 can hold the resource number 4 (*tape*).
- Process3 can hold the resources number 2,3 or 4 (*Printer*, *Scanner* or *Tape*).
- Process4 can hold the resources number 1,2 or 3 (*CD*, *Printer* or *scanner*).

A clause will view the acceptance of allocating the resource, and if not the clause will refuses the current allocation and ask for another allocating resource.

4. Used algorithms in PDADL implementation

The implementation involves several methods and approaches to be finalized. Theses approaches and methods are establishing some important algorithms to improve there capabilities of the system theses algorithms are conducted as follow:

4.1 The allocating resources algorithm

This algorithm defines the allocating resource procedure and which contracted with sequences of events shown in figure 1, as follow:

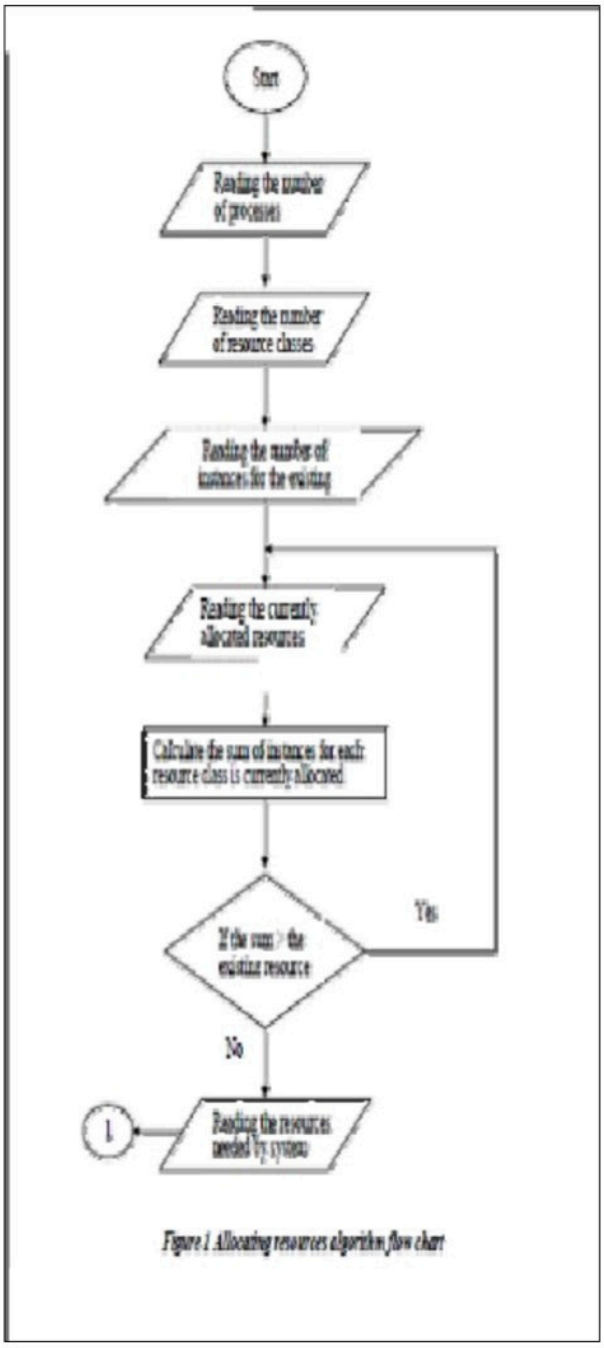


Figure 1 Allocating resources algorithm flow chart

4.2 The safety algorithm

The safety algorithm evaluates the available, existing and the allocating resource procedure to determine whether the state is safety or not. The algorithm is stated in figure 2, as follow:

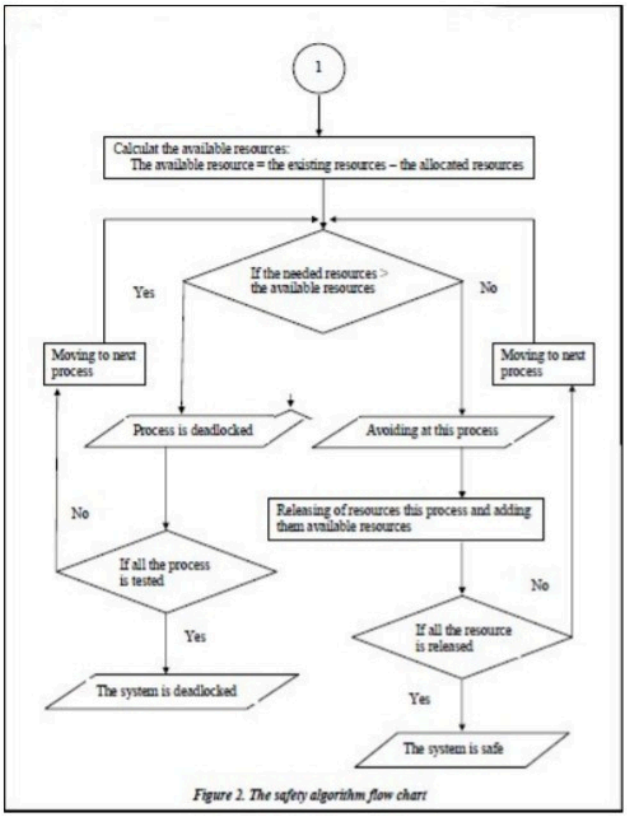


Figure 2. The safety algorithm flow chart

4.3 Preventing deadlock algorithm

The preventing algorithm investigates the resource classes, the numerical serial order, in order to prevent deadlocks from occurring, and the algorithm was indicated in figure 3, as follow:

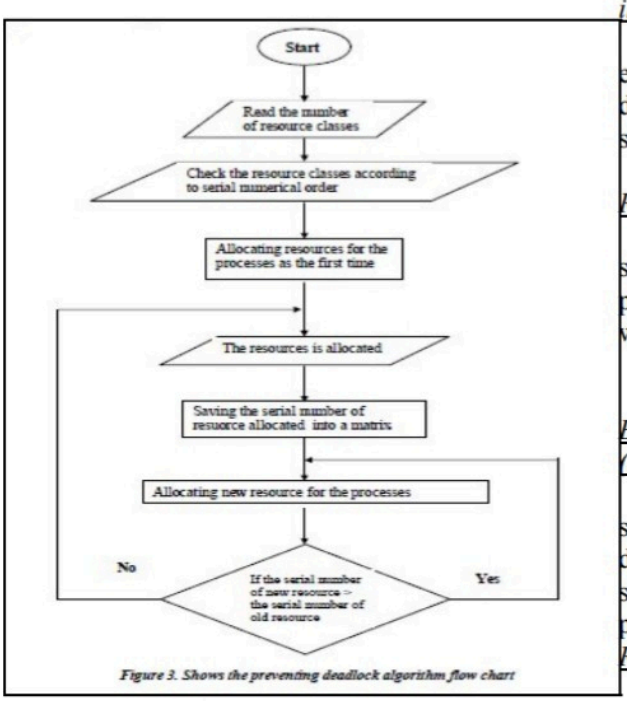


Figure 3. Shows the preventing deadlock algorithm flow chart

5. Experiments and Results

In this paper, the implemented proposal system for preventing, detecting and avoiding deadlock were designed especially for local operating systems. The performance of the proposed system is to evaluate the processes and resources through out an extensive simulation experiments. This study has also been carried out to compare the performance of the several local operating systems with our proposed system in the firm of using our modified algorithm. Results indicated that our proposed system performs better than the current preventing deadlock algorithm used. Results also indicated that the performance of the new modified algorithm is substantially better than the used in current operating systems. Theses experiments of our proposed system will be expressed and some meaningful results will be shown in table 1, the experiments are listed as follow:

Experiment #1: Testing the resources allocation procedure:

The purpose of this experiment is to ensure allocating the resources.

Results:-

The capability of control in allocating the resources by the Resources allocation procedure was shown clearly in this experiment.

Experiment #2: Detecting and avoiding Deadlock in some processes:

This experiment examines the safety procedure established for, whether it protect the system from deadlock or not, while detecting procedure in some situations in system processes.

Results:-

In this experiment a system that contains some stopped (hold) processes should be tested for better performance via a safety process. Safety procedure was succeeded in avoiding the problem of deadlock.

Experiment #3: Detecting deadlock in all processes (the full system is deadlocked):

The aim of this experiment is to examine the safety procedure by testing the system when it's full deadlocked, which means that, all processes are stopped, whereas it can not avoid occurring the problem.

Results:-

System with a complete stopping (holding) state

should be examined as well, meanwhile, the safety procedure failed to avoid the deadlock, because this procedure needs at lest one active process to avoid the current problem.

Experiment #4:Detecting deadlock in some processes with no avoiding ability:

This experiment focus on examining the safety procedure in case of protecting the system from deadlocks, after detecting it in some running processes in the system.

Results:-

This experiment is executed for the system when it's partially stopped; in this case the safety procedure fails to avoid the deadlock although there is a non stopped process. This situation happens because of the allocated resources for this process was not enough to support the available resources.

Experiment #5: Examine the preventing deadlock with an accepted allocation:

The goal of this experiment is to investigate the deadlock preventing method used, whereas the resources allocations are accepted according to the applied conditions in the method.

Results:-

In this experiment, the system was using the allocation of resources according to the utilized method to prevent deadlock. The result shows that the acceptances of the allocation process never cause or permit the deadlock at all.

Experiment #6: Testing the system from prevented allocation to prevent deadlock:

The final experiment examines the resources allocation that are prevented in case of contrary to the conditions clause in the method used for preventing deadlock.

Results:-

Finally the system should be tested, upon the allocated resource which leads to deadlock; therefore, the result in this experiment shows the rejected allocation process. It means that the system must reallocate the resources again according to the utilized method to prevent the deadlock.

5.1 Summary of the experiments and results

Experiment	The purpose	The result
Exp #1	Testing the resources allocation procedure	The capability of control in allocating the resources by *the resources allocation procedure* were shown clearly.
Exp #2	Detecting deadlock in some processes and he ability for avoiding it.	*The safety procedure* succeeds in avoiding the problem of deadlock
Exp #3	Detecting the deadlock in all processes (the system is fully deadlocked).	The *safety procedure* will be failed in avoiding the problem of deadlock.
Exp #4	Detecting deadlock in some processes and no ability for avoiding it.	The *safety procedure* failed to avoid the deadlock although there is a non stopped process.
Exp #5	Testing system which is using the allocation of resources according to the utilized method to prevent the deadlock.	Results show the acceptance of the allocation process which never cause or permit the deadlock at all.
Exp #6	Testing system which is using the allocation of resources by a random method.	Result shows the rejection of the allocation process, and the system must reallocate the resources. According to the utilized method to prevent the deadlock.

Table 1. Shows the summary of the experiments

6. Conclusion

In this paper we have dealt with some cases of deadlock in some modern operating system by carrying out the strategy of allocating the resources of running processes. The resources and processes represent the most important factors in the problem domain. The selection of the ideal style to allocate the resource would serve the best protection for the system without excessive time or trial to give the system the utmost efficiency.

Our contribution were the presenting of a new proposed system called (PDADL) which may deal with deadlock through dividing the system into two sub models. The first model was to detect and avoid the deadlock. The second model is to prevent the deadlock from occurring.

Within the first model the system begins automatically to detect the problem by searching for the capability of avoiding the deadlock before the

holding state occurs in the system. The second model provided us with the best method of this problem by preventing it from occurring. We observe that avoiding deadlocks may stand as the following:

1) The avoiding may occur by only one process which shouldn't be in the waiting state.

2) Sometime one process will never satisfy to avoid deadlock between other Processes, i.e, adding the process resources to the available resources are useless, because the needed resources is still grater than the available resources. Therefore, the recovering of the problem is necessary.

In this part the banker's algorithm plays the main role in avoiding the deadlock but with some modification made by us. The modification is to request indefinite resources to the process rather than using definite resources represent the greatest number of resources which needed by each process, this number is defined before starting execution. Moreover, this may be considered as a demerit or a weak point of this algorithm because we can never expect it in advance.

To prevent deadlock we eliminate the circular wait condition by making the resources of the system in a numerical circular order. With this condition it will reduce the possibilities of occurred the deadlock in the future.

In some cases the possibility of waiting some processes may occur, since it won't involve the system in a deadlock, for the reason that the deadlock condition will stop in each processes in the system, but it may retardate some important processes as well.

In comparison with the an alternative methods for preventing deadlock we find out that the used methodology is the best method to prevent deadlock state in some circumstances listed as follow:

- In elimination of mutual exclusion method it gives the possibility of allocating one resource for more than one process, whereas every process tries to perform definite application which may causes great problems to the system.

- In elimination of hold and wait method it achieved by preventing the processes that holds some resources from a waiting resources which requires all the processes to request all their resources before starting execution. It may cause problems since we can never expect it in advance.

- The elimination of no-preemption method it snatches the resource and reallocates it to another process even if the process needs resource or not, as a consequence it would affect the system.

Therefore, we finally state that utilizing the method of circular condition is a better method solution since it gives high efficiency to the system without complications.

References

[1] Andrew S. Tanenbaum and Albert S. Woodhull, "Operating System:Design and Implementation". New jersey: Prentice Hall, 2006.

[2] Coffman E. G., Elphick M. J and Shoshani A. " System Deadlock", Computing Surveys, June 1971.

[3] Holt R., "Some Deadlock Properties of computer Systems". Computing, Surveys, September, 1972.

[4] Isloor s., Marsland T., "The Deadlock Problem: An Overview". Computer, September 1980.

[5] Tanenbaum Andrews, "Modern Operating System", 2nd Edition. Englewood Cliffs, New jersey: Prentice Hall, Dec 2001.