Deadlock detection using edge chasing algorithm

Faculty – Prof. Parvathi R
Group members – Lawrence Borah
Amita panwar
Sruthy k
Rachna L
Snehal Murshetwad

Abstract -

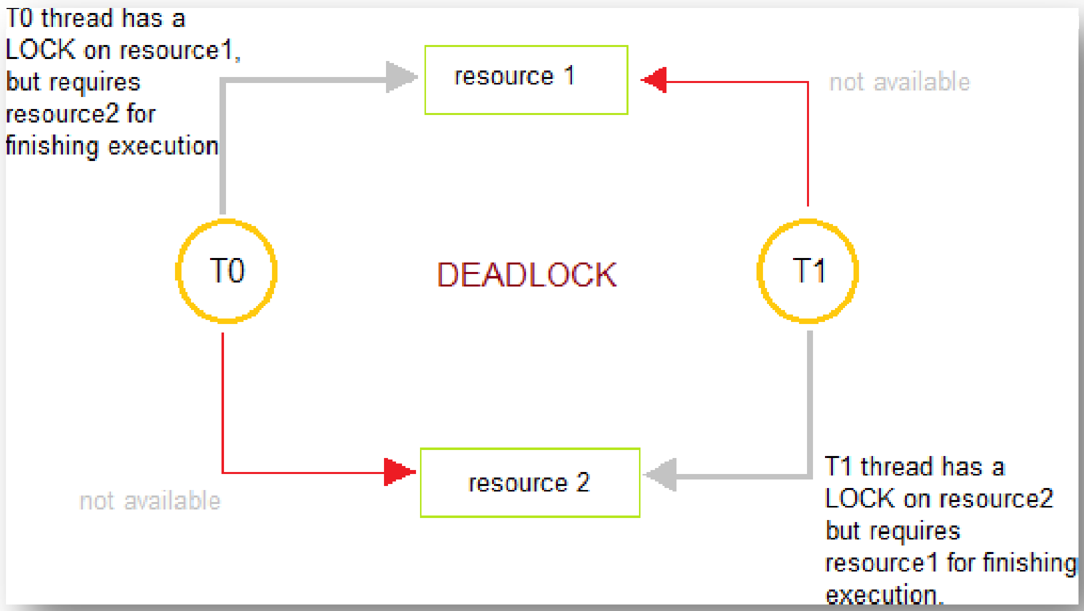
Deadlock is one of the canonical problems in distributed systems. It arises naturally in distributed systems such as distributed databases, distributed operating systems and store and forward networks. In these systems, the data reside at multiple distributed locations, and the processes have to collaborate in order to perform a task. Deadlock is characterized by a set of processes that are waiting for resources held by other processes in the set indefinitely. It reduces the throughput and resource availability in distributed systems. It can be handled through three ways, namely: deadlock prevention, deadlock avoidance and deadlock detection. The suitability of a specific deadlock handling approach highly depends on the application and environment. Deadlock prevention and avoidance are impractical or inefficient and relatively expensive as compared to deadlock detection approach in distributed systems.

Introduction –

A distributed system is a network of sites that exchange information with each other by message passing. A site consists of computing and storage facilities and interface to local users and a communication network. In distributed systems, a process can request and release resources in any order, which may not be known a priori and a process can request some resources while holding others. If the sequence of the allocation of resources to processes is not controlled in such environments, deadlocks can occur. Several deadlock detection techniques based on various control organizations are described. Pros and cons of these techniques are discussed and their performance is compared.

What is a Deadlock?

Deadlocks are a set of blocked processes each holding a resource and waiting to acquire a resource held by another process.


T0 thread has a LOCK on resource1, but requires resource2 for finishing execution
resource 1
not available
T0
DEADLOCK
T1
not available
resource 2
T1 thread has a LOCK on resource2 but requires resource1 for finishing execution.

Causes of Deadlocks

Deadlocks can be avoided by avoiding at least one of the four conditions, because all this four conditions are required simultaneously to cause deadlock.

1. Mutual Exclusion

Resources shared 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.

2. Hold and Wait

In this condition processes must be prevented from holding one or more resources while simultaneously waiting for one or more others.

3. No Preemption

Preemption of process resource allocations can avoid the condition of deadlocks, where ever possible.

4. Circular Wait

Circular wait can be avoided if we number all resources, and require that processes request resources only in strictly increasing (or decreasing) order.

Deadlock handling strategies in distributed systems –

There are three strategies to handle deadlocks that are deadlock prevention, deadlock avoidance and deadlock detection. Deadlock handling is complicated to implement in distributed systems because no one site has accurate knowledge of the current state of the system and because every interstice communication of the relative complexity and involves a finite and unpredictable delay. An examination of the relative complexity and practicality of these three deadlock handling strategies in distributed systems follows.

Deadlock prevention

Deadlock prevention is commonly achieved by either having a process acquire all the needed resources simultaneously before it begins execution or by preempting a process that holds the needed resource. In the former method, a process requests a remote resource by sending a request message to the site where the resource is located. This method has a number of drawbacks. First, it is inefficient as it decreases the system concurrency.

Second, a set of processes can become deadlocked in the resource acquiring phase.

Deadlock Avoidance

In the deadlock avoidance approach to distributed systems, a resource is granted to a process if the resulting global system state is safe. Because of the following problems, deadlock avoidance can be seen as impractical in distributed systems:1) every site has to maintain information on the global state of the system, which translates into huge storage requirements and extensive communication costs 2) The process of checking for a safe global state must be mutually exclusive, because if several sites concurrently perform checks for a safe global state, they may all find the state may not be safe. This restriction will severly limit the concurrency and throughput of the system 3) Due to the large number of processes and resources, it will be computationally expensive to check for a safe state.

Deadlock Detection

Deadlock detection requires an examination of the status of process- resource interactions for the presence of cyclical wait. Deadlock detection in distributed systems has two favourable conditions 1) once a cycle is formed in the WFG, it persists untilit is detected and broken and 2) cycle detection can proceed concurrently with the normal activities of system.

An edge chasing algorithm-

Chandys algorithm uses a special message called a probe. A probe is a triplet(i, j, k) denoting that it belongs to a deadlock detection initiated for process Pi and it is being sent by the home site of process Pj to the home site of process Pk.

A probe message travels along the edges of the global TWF graph, and a deadlock is detected when a probe message returns to its initiating process.

We now define terms and data structures used in the algorithm. A process Pj is said to be dependent on another process Pk if there exists a sequence of processes Pj, Pi1, Pi2,…….Pim, Pk such that each process except Pk in the sequence is blocked and each process, except the first one (Pj) holds a resource for which the previous process in the sequence is waiting. Process Pj is locally dependent upon process Pk if Pj is dependent upon Pk and both the processes are at the same site. The system maintains a Boolean array, dependent, for each process Pi,where dependent (j) is true only if Pi knows that Pj is dependent on it. Initially, dependent (j) is false for all I and j.

The algorithm – to determine if a blocked process is deadlocked, the system executes the following algorithm:

If Pi is locally dependent on itself then declare a deadlock

Else for all Pj and Pk such that

a) Pi is locally dependent upon Pj and

b) Pj is waiting on Pk, and

c) Pj and Pk are on different sites, send probe(i, j, k) to the home site of Pk on receipt of probe (i, j, k) the site takes the following actions: if

d) Pk is blocked, and

e) dependent k (i) is false, and

f) Pk has not replied to all requests of Pj, then begin dependent k (i)= true; if k=I then declare that Pi is deadlocked else for all Pm and Pn such that a') Pk is locally dependent upon Pm, and b') Pm is waiting on Pn and c') Pm and Pn are on different sites, send probe (I,m,n) to the home site of Pn end thus a probe message is successively propagated along the edges of the global TWF graph and a deadlock is detected when a probe message returns to its initiating process.

Problems in deadlock detection and resolution

Deadlock detection and resolution entails addressing two basic issues – first, detection of exisiting deadlocks and second resolution of detected deadlocks.

Detection

The detection of deadlocks involves two issues: maintainance of the WFG and search of the WFG for the presence of cycles greatly depends upon the manner in which WFG information is maintained and the search for cycles is carried out, there are centralized, distributed and hierarchical algorithms for deadlock detection in distributed systems.

A correct deadlock detection algorithm must satisfy the following two conditions-

Progress-No undetected deadlocks. The algorithm must detect all existing deadlocks in finite time. Once a deadlock has occurred has occurred, the deadlock detection activity should continuously progress until the deadlock is detected. In other words, after all wait for dependencies for a deadlock have formed, the algorithm should not wait for any more wait-for dependencies to form to detect the deadlock.

Safety-No false deadlocks. The algorithm should not report deadlocks which are non-existent. In distributed systems, where there is no global memory and communication occurs solely by messages, it is difficult to design a correct deadlock detection algorithm because sites may obtain out of date and inconsistent WFGs of the system. As a result, sites may detect a cycle that doesn't exist, but whose different segments were existing in the system at different times.

Resolution

Deadlock resolution involves breaking existing wait-for dependencies in the system WFG to resolve the deadlock. It involoves rolling back one or more processes that are deadlocked and assigning their resources to blocked processes in the deadlock so that they can resume execution. Note that several deadlock processes in the deadlock so that thay can resume execution. Note that several deadlock detection algorithms propagate information regarding wait-for dependencies along the edges of the wait-for graph. Therefore, when a wait-for dependency is broken, the corresponding information should be immediately cleaned from the system. If this information is not cleaned appropriately in a timely manner, it may result in detection of phantom deadlocks.

Language – the C language used in this coading

C is a general-purpose programming language that is extremely popular, simple and flexible. It is machine-independent, structured programming language which is used extensively in various applications.

C was the basics language to write everything from operating systems (Windows and many others) to complex programs like the Oracle database, Git, Python interpreter and more.

It is said that 'C' is a god's programming language. One can say, C is a base for the programming. If you know 'C,' you can easily grasp the knowledge of the other programming languages that uses the concept of 'C'

It is essential to have a background in computer memory mechanisms because it is an important aspect when dealing with the C programming language.

Key Applications

1. 'C' language is widely used in embedded systems.
2. It is used for developing system applications.
3. It is widely used for developing desktop applications.
4. Most of the applications by Adobe are developed using 'C' programming language.
5. It is used for developing browsers and their extensions. Google's Chromium is built using 'C' programming language.
6. It is used to develop databases. MySQL is the most popular database software which is built using 'C'.

7. It is used in developing an operating system. Operating systems such as Apple's OS X, Microsoft's Windows, and Symbian are developed using 'C' language. It is used for developing desktop as well as mobile phone's operating system.
8. It is used for compiler production.
9. It is widely used in IOT applications.

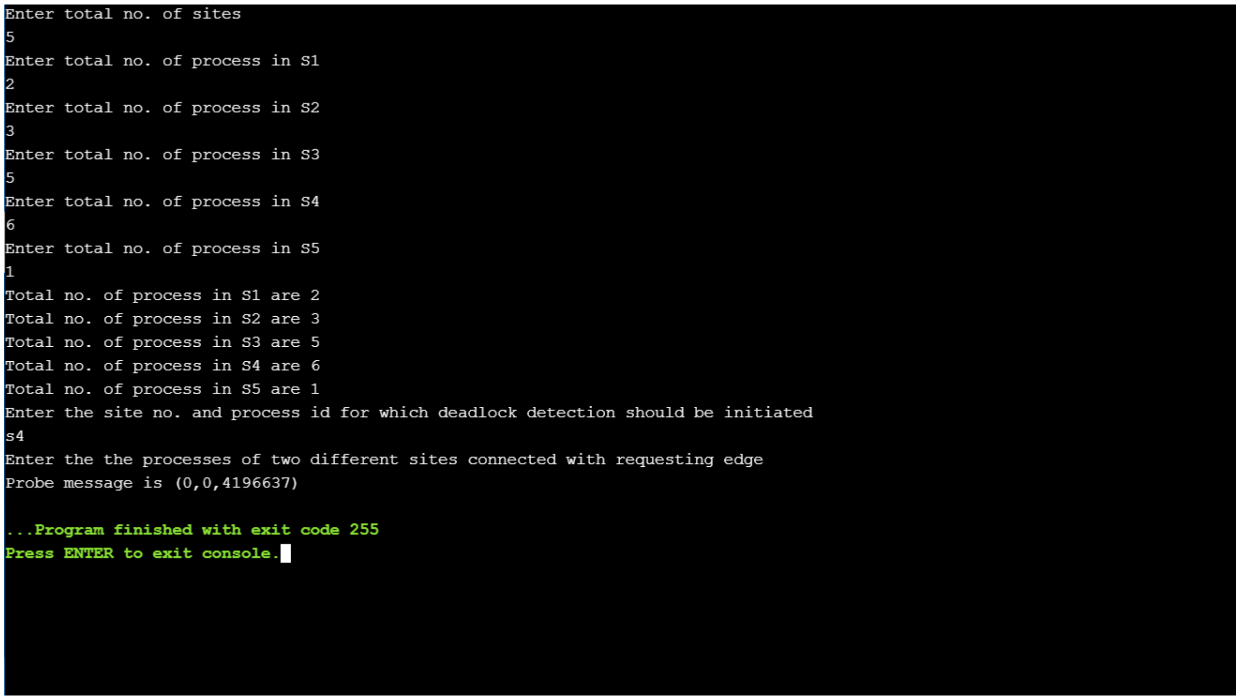
Code

```
include<conio.h>
#include<stdio.h>
void main()
{
int p[10],n,i,p1,s1,sp1,sp2;
printf("Enter total no. of sites\n");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter total no. of process in S%d\n",i+1);
scanf("%d",&p[i]);
}
for(i=0;i<n;i++)
{
printf("Total no. of process in S%d are %d\n",i+1,p[i]);
}
printf("Enter the site no. and process id for which deadlock detection shold be initiated\n");
scanf("%d %d",&s1,&p1);
printf("Enter the the processes of two different sites connected with requesting edge\n");
scanf("%d %d",&sp1,&sp2);
printf("Probe message is (%d,%d,%d)",p1,sp1,sp2);

if(p1==sp2)
{
printf("Deadlock detected");
}
else
{
getch();
}
}
```

Screenshot-

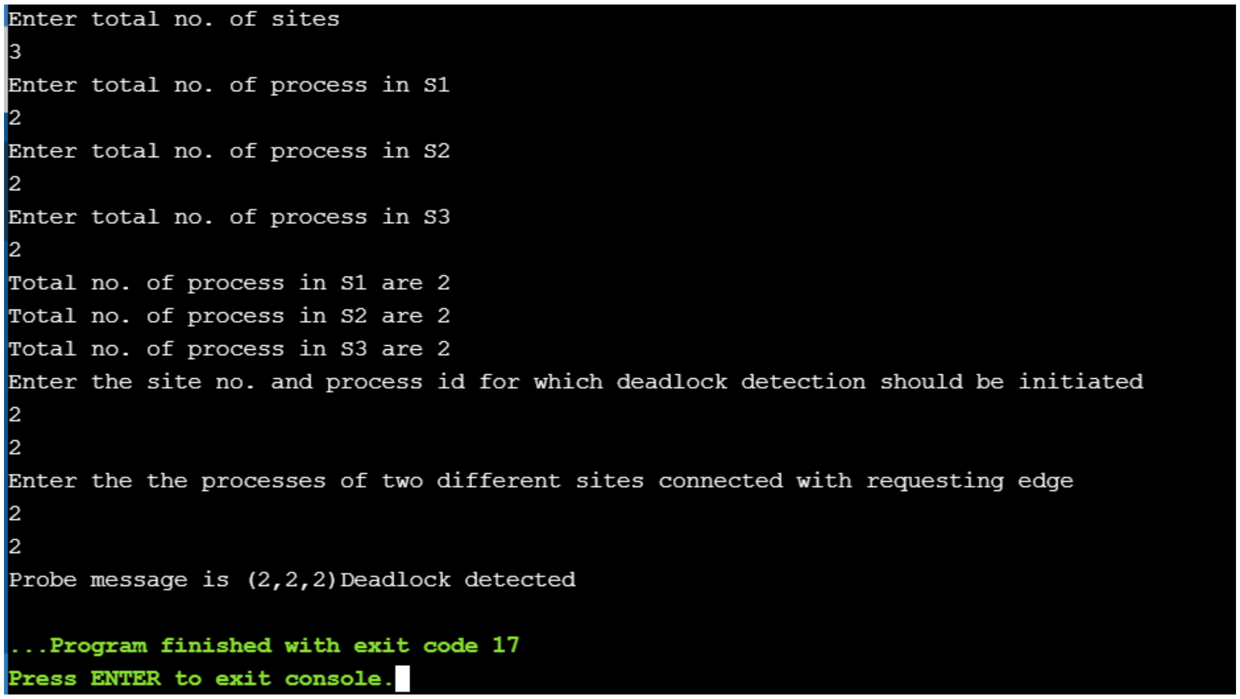
(Output 1: No Deadlock Detected)



```
Enter total no. of sites
5
Enter total no. of process in S1
2
Enter total no. of process in S2
3
Enter total no. of process in S3
5
Enter total no. of process in S4
6
Enter total no. of process in S5
1
Total no. of process in S1 are 2
Total no. of process in S2 are 3
Total no. of process in S3 are 5
Total no. of process in S4 are 6
Total no. of process in S5 are 1
Enter the site no. and process id for which deadlock detection should be initiated
s4
Enter the the processes of two different sites connected with requesting edge
Probe message is (0,0,4196637)

...Program finished with exit code 255
Press ENTER to exit console.
```

(Output 2: Deadlock Detected)



```
Enter total no. of sites
3
Enter total no. of process in S1
2
Enter total no. of process in S2
2
Enter total no. of process in S3
2
Total no. of process in S1 are 2
Total no. of process in S2 are 2
Total no. of process in S3 are 2
Enter the site no. and process id for which deadlock detection should be initiated
2
2
Enter the the processes of two different sites connected with requesting edge
2
2
Probe message is (2,2,2)Deadlock detected

...Program finished with exit code 17
Press ENTER to exit console.
```

Reference-

1. chandy, K.M.,J, misra and L.M Haas."Distributed Deadlock detection", ACM trans. On computer systems, May 1983.
2. Geeks for geeks
3. Composition.AI blog
4. https://github.com
5. www.cs.uic.edu.