ADA AND THE CONTROL OF INTRUSION

Gertrude Levine
Fairleigh Dickinson University
levine@fdu.edu

Abstract

One of the strengths of the Ada Programming Language is its support of software engineering principles. In particular, many Ada constructs assist in the control of various anomalies of computer systems. This paper discusses two classes of anomalies that are shown to have similar characteristics. The first of these classes involves the mutual exclusion problem, for which there exists considerable literature within the Ada community. We compare this category of anomalies to intrusion, a security anomaly, and suggest that mechanisms for the control of the first should be considered for the second.

Introduction

Earlier papers have investigated a specific anomaly of mutual exclusion within areas of computer science such as networks and operating systems [9, 10]. The term "interleaved dead states" was coined for the situation in which processes are denied completed service due to the interleaved execution of their requests. (Processes are assumed to be developed correctly so that they could complete service if not for flaws in the scheduling mechanisms.) Interleaved dead states with output requests are now contrasted with intrusion, an anomaly of security control, which we define as output of unauthorized data onto a resource system, preventing or aborting current service of authorized data. Both inadvertent and advertent attacks are covered in this paper. (Victim processes are assumed to be developed correctly so that they could complete service if not for flaws in the security mechanisms.) Dead states caused by the interleaved execution of output requests have many characteristics in common with the anomaly of intrusion. For example, both contain sets of processes that are denied completed service due to the interference of output from other processes.

Input interference is not dealt with in this paper. Such passive anomalies, including, for example, inconsistent retrieval in databases [1] (due to incorrect interleaved execution of input requests with output requests) and identity theft in security systems (due to an attacker's unauthorized input of information), although preventable by similar mechanisms as output interference, are considerably more difficult to detect and recover from.

The categories of interleaved dead states and intrusion are each subdivided into binary dead states and n-ary dead states. Basically, these two subclasses have different characteristics, although there is considerable overlap between them.

1. Binary Dead States

A binary dead state is a dead state in which a single process can be identified as preventing the service of another process. In interleaved binary dead states, each of the processes that are interleaved in execution interferes with the service of the other; in intrusion binary dead states, an attacker interferes with the victim's service.

1.1 Binary interleaved dead states

If processes must hold (at least) two non-sharable resource entities concurrently and if their accesses to these resources are interleaved, inconsistent data can occur. One example of such an anomaly is the lost update problem [1], where two (or more) processes each read a stored value and then output an updated value based upon the original read. The anomaly occurs due to a specific order of interleaved execution of requests. A system may not even be aware of any error until, perhaps, two passengers show up for the same airline seat. As an example of another type of interleaved binary dead state, consider two (or more) processes that alternate in positioning a cylinder arm before completing I/O. Since the I/O cannot be achieved unless the disk arm is over the correct cylinder and each process, in turn, overwrites the output of the other, indefinite or infinite waits may occur [6]. A lock, such as a flag or semaphore, is typically used to protect critical sections, the code to the shared resources, in order to provide mutually exclusive access to shareable resources. Waiting processes either are suspended on a queue or spin on a test of the resource's lock. Sophisticated locking policies have been devised to maximize concurrency. For example, readers may be allowed simultaneous access to resources. The Locking Manager in VMS® contains six degrees of semaphores to control concurrent access to a resource [12]. A task must wait until its request for a lock is compatible with other granted lock modes.

If processes request uniquely identifiable resources and if they hold non-preemptible resources that must be used in mutual exclusion while requesting others, interleaved execution of their requests can cause a circular waiting state called resource deadlock [9]. Various deadlock prevention policies, including resource preallocation and the establishment of a linear order of accesses to resources, can prevent deadlock. These mechanisms, however, are restrictive and are impractical with dynamically requested resources. Alternatively, a system may allow resource deadlock to occur and attempt to recover from it. The VMS® Locking Manager times a task's wait on a lock; if a wait exceeds a preset time interval, detection algorithms check for a circular waiting condition [12]. Circular dead states (sometimes called deadlock [5]), however, may be caused by incorrect programs, those that could not complete service no matter how they are scheduled. Resource deadlock, on the other hand, is caused by interleaved execution of requests; programs may have executed correctly for years before a specific interleaved scheduling order caused an infinite wait. Unlike many other types of deadlocks, it is possible to recover from resource deadlock by periodic backups of processes and data (which is also effective as a control of intrusion and hardware failures [17]); a process is chosen as a victim to be aborted and restarted; if the process is a node in the circular chain of a binary resource deadlock, killing it will break the deadlock.

1.2 Binary intrusion dead states

We compare the above situation with the intrusion problem. An intruder attempts to access a system in order to overwrite a victim's data. We assume that the victim's data is available for disciplined access by multiple authorized processes. A security system must protect the data and the processes from intrusion, however, with services such as authorization and authentication, based on its access policies. Similar to lock management policies, storage management policies protect the operating system and user data from unauthorized accesses [16]. As intruders have become more sophisticated, enhanced mechanisms have been developed. Security policies define services whose mechanisms prevent information from leaking into areas of lesser privilege [4]. User-role based security access for object-oriented systems provide

access based on least privilege [13]. Since it is possible that an intruder circumvents security mechanisms, perhaps by obtaining an authorized user's access codes, or that an insider launches attacks, systems routinely provide for detection and recovery. Detection mechanisms, such as integrity check values and audit trails, warn of system penetration. Recovery typically involves periodic backups (which is also effective as a control of resource deadlock and hardware failure) with restarts of victim processes.

1.3 Ada and binary dead states

There exists a considerable amount of literature in the Ada community on the mutual exclusion problem [2, 3, 14, 18, 19] as well as on deadlocks [5, 7, 8, 11, 15], (although some of the literature on deadlocks does not deal with interleaved execution of requests, i.e., resource deadlock). Policies for the control of the mutual exclusion problem typically utilize Ada mechanisms for the protection of critical code within a protected module or passive task in order to prevent interleaved execution. Tasks may hold onto some resources while requesting others within different modules, however, and the locks may exist in a distributed environment where a lock manager is not appropriate. Thus the possibility of resource deadlock must be dealt with. Detection schemes may check for circular waits that satisfy the criteria of wait-for graphs [7], then choose a task to abort and restart.

The Ada Programming Language and an APSE also support various security policies for intrusion control. Mechanisms that assist storage management policies include strong typing, range checks, restricted pointer accesses, and parameter checks. Many viruses and worms have taken advantage of stack and buffer overflows that could be prevented with Ada mechanisms. Ada also supports encapsulation and restricted access to methods, with packages and private interfaces. An APSE interacts with an operating system [4, 12, 13, 16], so that locking and access mechanisms provided by an operating system can be utilized by an Ada program.

Binary dead states	Caused by	Prevented by	Detected By	Recovery by
Interleaved dead states with output requests	Interference between two independent processes	Lock management policies Resource server (protected module, passive task)	Client conflicts; Wait-for graphs	Replication (system backup) Restart of victim(s)
Intrusion	Interference of intruder with independent process	Access policies Security server (firewall, router)	Client conflicts; Integrity check values; audit logs	Replication (system backup) Restart of victim(s)

Table 1: Binary Dead States
Comparison of Interleaved Dead States and Intrusion

2. N-ary Dead States

All systems are designed to provide service for some volume of traffic. If requests exceed the maximum service rate of the system for an extended period of time, anomalies can occur. A single process typically does not cause such anomalies and thus denying or aborting a single process will not allow the system to recover. In interleaved n-ary dead states, each of the processes has begun service, but, because the system keeps switching to servicing other processes, virtually none of these processes complete. In intrusion dead states, the large amount of traffic generated by the attacker(s) denies the service of multiple valid users.

Obviously, a binary dead state is a specific case of n-ary dead states, and the characteristics considered in the previous sections are applicable to many n-ary dead states where only a few processes are involved. As the number of requests and/or processes in interleaved dead states or in intrusion increases, however, prevention and recovery become more difficult.

2.1 N-ary interleaved dead states

When resources consist of multiple units that are not uniquely identifiable, resource deadlock can occur if processes obtain some units, but must wait for others to complete. The awaited resources are all held by other processes that also wait for resources. These processes could have completed service if their requests for resources had not been interleaved. Resource preallocation is an effective, though restrictive, prevention method for this type of deadlock (as it is for all interleaved dead states), but linearly ordering requests is not possible since it is impractical to uniquely identify each unit. Although n-ary resource deadlock has the same necessary preconditions for deadlock as binary resource deadlock, a circular wait is insufficient to characterize the deadlock (see figure 1). All units of requested resources must be held by processes in the circular wait to cause a resource deadlock (assuming that processes request one resource unit at a time) [10].

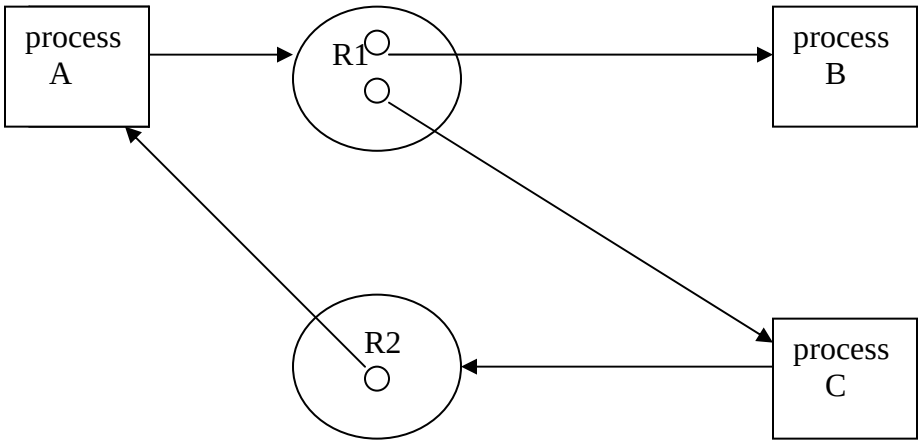

Figure 1: a circular wait without resource deadlock. When B completes, A and then C can continue executing.

.

N-ary resource deadlocks can occur in storage systems, with spooling discs, dynamic real memory, or network buffers, for example. They can occur with two processes, but this is rare; typically these systems have been designed with enough resources to satisfy many processes, but are overwhelmed with a sudden surge in requests. Prevention methods for these types of deadlocks, such as Banker's and Safe State algorithms, are restrictive. A resource system may rely on a heuristic, therefore, such as throttling new processes when 70% of resources are allocated.

In a virtual memory environment, a process routinely causes page (or segment) faults, resulting in faulted pages being scheduling for swapping into main memory. The process requesting the I/O for the missing pages is suspended, and a new process begins executing on the CPU. If too many processes are in a state of execution, their requests will continually be interleaved in execution and cause page thrashing. Operating systems monitor metrics such as the activity to the swapping disk and the rate of page faults, and throttle processes if the system is in danger of being overloaded.

In networks, during natural disasters and other stress conditions, a system may receive more packets than it can handle. If the packets of one connection are interleaved with many others, packets may be delayed, exceed their allocated time-to-live, and be discarded. Connections then time-out while waiting for acknowledgements, and resend the packets. This activity puts additional strain on an already stressed system. Network management monitors line capacity, buffer utilization, queue lengths and delays, and other metrics of resource use and refuse or delay connections' requests in order to prevent congestion.

Recovery from n-ary interleaved dead states may also be achieved by roll backs and restarts, but because so many processes are involved, much work will be lost. Processes that then exceed their total allotted time are never serviced. Thus, n-ary interleaved dead states should be prevented, typically by budgetary controls that monitor resource usage and throttle new processes before a threshold is reached.

2.2 N-ary intrusion dead states

If an intruder generates multiple attacks concurrently, it can achieve denial of service of user processes. Examples of these anomalies include Internet worms, DDoS, and spam. Although the host system may have installed routers, firewalls, and other mechanisms to prevent denial of service, these devices may be overwhelmed with the attacker's traffic and be unable to service user requests.

Security systems rely on budgetary controls to limit the damage of such attacks. Artificial Intelligence is also useful in attempting to distinguish between these threats and user traffic before the system is compromised. Efforts are made to identify malware as early as possible, preferably at edge nodes, to prevent intrusion becoming a service threat.

When Denial of Service does occur, detection is trivial. Systems must then reload their backups and restart the processes, but much work is lost.

2.3 Ada and n-ary dead states

The treatment of n-ary dead states is less common than binary dead states in the Ada literature. N-ary resource deadlock with both tasks and protected types is discussed in [11]. One of the examples from that paper appears below, somewhat modified.

```
   procedure Resource_Deadlock is
   Buffer_Number: constant NATURAL := 10;
   Task_Number: constant NATURAL := 2;

task Buffer_Server is
   entry Allocate;
   entry Deallocate;
end Buffer_Server;

task body Buffer_Server is
   Buffers_Left: NATURAL := Buffer_Number;
begin
   loop
      select
          when Buffers_Left > 0 =>
               accept Allocate;
               Buffers_Left := Buffers_Left + 1;
       or
               accept Deallocate;
               Buffers_Left := Buffers_Left - 1;
       or
               terminate;
      end select;
    end loop;
end Buffer_Server;

task type Caller (Buffers_Needed: POSITIVE);
task body Caller is
begin
    for i in 1.. Buffers_Needed loop
       Buffer_Server.Allocate;
    -- if  I = Buffer_Number/Task_Number then delay (0.1); end if;
    -- The delay value causes interleaved execution and, if both
    -- tasks receive 5 resource units while requiring 6 or 7,
    -- causes resource deadlock as well.
    end loop;
    for i in 1 .. Buffers_Needed loop
       Buffer_Server.Deallocate;
   end loop;
end Caller;

begin
    declare
       T:array (1..Task_Number)of Caller(Buffer_Number/Task_Number + 2);
     begin
       null;
     end;
end Resource_Deadlock;
```

In [11], suggestions for deadlock control include the release of buffer units following a timed wait for resources. If the reassignment of resources is disciplined, it will be effective for a small set of tasks. For a large number of tasks and resources, however, this method will not work; similar to the continual swapping of pages during page thrashing and discarding of packets during network congestion, preemption tends to worsen an already overloaded condition. A large quantity of interleaved authorized accesses or unauthorized accesses can overwhelm an Ada system, so that authorized users are denied service. These types of dead states require early detection and prevention.

N-ary dead states	Caused by	Prevented by	Detected by	Recovery by
Interleaved dead states	Interleaved execution of processes that require a service rate exceeding the maximum that a server can provide	Resource monitoring; Budgetary controls	Time-outs on processes	Backups and restarts if user processes have not exceeded their maximum allocated time
Intrusion	Interference from a large number of intruders	Resource and process monitoring; Budgetary controls	Time-outs on processes	Backups and restarts if user processes have not exceeded their maximum allocated time

Table 2. N-ary Dead States
Comparison of Interleaved Dead States and Intrusion

3. Summary

We have considered intrusion in relation to the mutual exclusion problem, with both involving competition for output to resources and with both being handled by similar prevention and detection mechanisms. We have reviewed Ada literature in this field in order to provide a unifying approach to the control of these anomalies.

References:

1) Bernstein, P. and Goodman, N., "Concurrency Control in Distributed Database Systems,"*ACM Computer Surveys,* 13, (2), June 1981, 185-211.
2) Bondeli, P., "A Fully Reusable Class of Objects for Synchronization and Communication in Ada 95", *Ada Letters* XIX (1) March 1999, 66-96.
3) Burns, A. *Concurrent Programming in Ada.* Ada Companion Series, Cambridge University Press, Cambridge (1985).
4) Chapman, R. and Hilton, A. "Enforcing Security and Safety Models with an Information Flow Analysis Tool", *Ada Letters* XXIV (4), December 2004, 39-46.
5) Cheng, J. "A Classification of Tasking Deadlocks", *Ada Letters,* X (5), May, June 1990, 110-127.

6) Flynn, I. M. and McHoes, A. M. *Understanding Operating Systems*, Brooks/Cole, 2001, p. 112.
7) German, S. M., "Monitoring for Deadlock and Blocking in Ada Tasking", *IEEE Trans. on Software Engineering*, SE-10 (6), 1984, 764-777.
8) Levine, G. "Controlling Deadlock in Ada", Ada Letters IX (4), May,June 1989, 87-91.
9) Levine, G. N. "Defining Deadlocks," *Operating Systems Review,* ACM Press, 37(1), January 2003, 54-64.
10) Levine, G. N., "Defining Deadlock with Fungible Resources", *Operating Systems Review, ACM Press,* 37(3), July 2003, 5-11.
11) Levine, T., "Deadlock Control with Ada 95", *Ada Letters* XVIII (2), March, April 1998, 67-80.
12) Maloney, J. "Using the VAX/VMS Lock Manager with Ada Tasks", *Ada Letters*, XIII (2), March April 1988, 84-95.
13) Reisner, J. and Demurjian, S. "Addressing Security for Object-Oriented Design and Ada 95 Development," *Ada Letters*, XVII (2), March, April 1998, 89-104.
14) Saeed, F. George, K. and Samadzadeh, M., "Implementation of Classical Mutual Exclusion Algorithms in Ada", *Ada Letters*, XII (1), Jan. Feb. 1992, 73-84.
15) Tojo, Y., Nara, S., Goto, Y., Cheng, J. "Tasking Deadlocks in Programs with the Full Ada95", *Ada Letters*, XXV (1), March 2005, 48-56.
16) Vlierberghe, S. "Memory Management in Ada83 and Ada9X", Ada Letters, XIV (4), July, Aug. 1994, 43-57.
17) Wolf, T., "Fault Tolerance in Distributed Ada 95", Ada Letters XVIII (5) Sept. Oct. 1997, 106- 110.
18) Yue, K. "An Ada Solution to the General Mutual Exclusion Problem", *Ada Letters*, XIII (4), July, Aug 1993, 37-43.
19) Yue, K. "Semaphores in Ada-94", *Ada Letters*, XIV (5), Sept.,Oct 1994, 71-79.