Control of Processes in Operating Systems: The Boss-Slave Relation

R. Stockton Gaines
Communications Research Division, Institute for Defense Analyses, Princeton NJ
and
The RAND Corporation, Santa Monica CA

Presented at SOSP-5, 1975

Abstract

This paper describes a boss-slave relationship between processes, different from the normal relationships between processes, which is useful for a number of purposes. These purposes include debugging of programs, analysis of process behavior, control of a process for security purposes, and simulation of a different operating system for a slave process. This mechanism was easily added to one operating system, and the ideas apply to a number of different operating systems.

This paper describes a boss-slave relationship between processes, different from the normal relationships between processes, which is useful for a number of purposes. These purposes include debugging of programs, analysis of process behavior, control of a process for security purposes, and simulation of a different operating system for a slave process. The relation described will not be applicable to all operating systems, but could usefully be added to many operating systems in which processes operate in a relatively autonomous manner. For systems such as the RC4000 described by Brinch Hansen [1], in which each process is really a subset of the address space of some other process, the extremely tight control each process exercises over its progeny would render the mechanism described in this paper superfluous. However, operating systems which allow processes to operate independently of their parents are amenable to process organizations which are not possible in systems with strictly hierarchical control of processes. The relation described in this paper is very useful in such systems. It has been implanted in the IDA Operating System [2], where it has proved to be quite valuable.

The term “boss” has been used in this paper in preference to other terms such as “master” in order that we may clearly distinguish between the activity of these processes and the activities that are part of the operating system proper, particularly the supervisor. In most systems, some code or some processes operate in a special privileged state, often called Master Mode or Supervisor Mode. The processes which we designate as bosses in this paper do not operate with privileged instruction execution or memory access status. Throughout this paper we will use the term supervisor to refer to that part of the operating system which controls the activities of processes and provides an environment in which “normal” processes in the system operate.

The plan of this paper is the following: First we will describe the boss-slave relation, and mechanisms in the operating to support it, in a general way. Included in this description will be some details of the implementation in the IDA Operating System [2] which will serve to amplify the general ideas being put forth. Then some uses of the boss-slave relation which have proved valuable in our experience will be described, together with possible additional uses of the boss-slave relation. These ideas will also be contrasted with previous work.

General description of the boss-slave relation

For the purposes of this paper, we suppose that a process consists of a set of information called a description block[1], describing the process to the supervisor, and a memory image. The memory image is difficult to define in a general way (though it is easy to define it precisely for any given operating system). Roughly it corresponds to the set of addresses a process can reference directly when it is running on a processor without invoking the supervisor of the operating system, and a copy of that information at other times (when the process is blocked, or active but not actually running). We consider that when a process is not running on a processor, the values of the machine registers which are used by or affect that process are stored as part of its description block.

We suppose that processes in the system are normally in one of two states, which we will call active and blocked. An active process is one that can proceed with its computation when assigned a processor, and a blocked process is one waiting to be made active by the operating system when some event occurs, such as a message or signal from another process, the system clock reaching a certain value, etc..

A slave process differs from a normal process by having recorded in its description block the name of its boss, and a set of interrupt conditions. When the slave process is in the active state, it is treated by the supervisor just as a normal process is. That is, when assigned a processor, it proceeds with its computation in a normal way. However, when the slave makes a supervisor call, and thus interacts with the supervisor, that interaction is controlled by the interrupt conditions.

The essential interrupt conditions are:

Error
Start of a Supervisor Call
Termination of a Supervisor Call
Enter Blocked State
Exit from Blocked State

[1] In some operating systems, the information about a process is distributed in tables throughout the operating system. We will still think of this information conceptually as a unit, called the description block.

These are identified by bits in the field containing the interrupt conditions, and any combination of these bits may be set at any time.

We allow a process to enter a new state, in addition to blocked or active, called suspended. The supervisor will automatically suspend a slave process when a condition arise for which the slave has an interrupt condition bit set. A suspended process will remain suspended until some other process switches it to an active or blocked state, and is essentially an inert process within the system until then.[2]

A boss process is identified as such by the fact that its name appears in the boss field of the description block of some slave process. A boss may have more than one slave. When a slave is suspended by the supervisor because it reached an interrupt condition, two things must occur:

(1) the boss must be notified that this happened.

(2) the memory image and description block of the slave must be made accessible to the boss process.

The details of how the boss will be notified that a slave who was suspended will depend on the particular operating system. The boss should be informed that the slave was suspended, the name of the slave, and what condition caused the slave to be suspended. In order that the boss may have complete control over the slave, the boss must be able to examine and modify the slave's memory image. The boss would also need to access the slave's description block. The boss should be able to access the slave's memory image and description block to the same extent that it could access its own. Then the boss will have the same control over the slave that it has over itself.

A boss process has several actions it can take with regard to a slave. First, it can create the slave process. In the IDA operating system this is the only way process becomes the slave of another process. In other implementations, it might be desirable to allow a process with appropriate rights to designate that another process become its slave. Also it might be appropriate for a boss process to be able to release a slave. We will later discuss one possible use for the latter two features.

A boss process can suspend a slave process by means of a supervisor function for this purpose. Thus a slave can become suspended either because an interrupt condition was recognized by the supervisor or because it was put in this state by its boss. A boss can also resume a slave. To resume a slave, its boss must specify the new state (blocked or active) of the slave. If the slave is being switched to the active state, the boss must also

---

[2] The notion of a suspended state was first discussed by Lampson [3]. He suggested that a process should be suspended while the program it was running was being debugged, which is one of the uses we have in mind also. He gave no details about how a process might actually enter or leave the suspended state, however.

specify where in the computation the slave should resume its activities. When the slave is resumed, the boss also specifies which interrupt condition bits should be set.

An important characteristic of the boss-slave relationship is that the boss be able to examine and modify the slave's memory image and description block when it is suspended. This ability is necessary in order that the boss can have complete control over the slave. The limits on this should be the same as the limits that apply to the boss's own memory image and description block. Though processes are not permitted to modify the description blocks directly in most operating systems, they can do so indirectly by means of supervisor calls. For example, in some systems the list of files currently opened by a process is part of the information in its description block, and this is modified by the supervisor calls opening and closing files. A boss may indirectly modify the description block of a slave by modifying the memory image of the slave so that it makes an appropriate supervisor call, and by setting the appropriate interrupt condition bits, regain control of the slave after the supervisor call completes.

Implementation of the boss-slave relationship in the IDA Operating System

This section covers some of the details of the implementation of the boss slave relationship in the IDA Operating System. Familiarity with reference [2] would be helpful to the reader in places but the main ideas do not require it.

The supervisor was modified so that for a slave process, a check is made in several places to determine whether it should be suspended or not. These checks occur in the Error Handler, and before and after the Supervisory Computer (which executes the supervisor calls on behalf of processes) is invoked. At these points, the Interrupt Condition Bits of the slave are checked, and the process is switched to the suspended state if appropriate.

The supervisor of the IDA Operating System will release most of the space in central memory occupied by a process by creating a file and copying the process's memory image into it (except for a small piece, called the Communications Area, which is used for communications with the supervisor and other processes). This normally occurs when deemed desirable by the Memory Manager. However, it also happens when a slave process is suspended, so that the memory image of the slave will be available for examination and modification by its boss.

The way a boss is notified that a slave has been suspended depends on the information in the Interrupt word in the boss's description block. The boss may, by means of a supervisor function, designate an address in its memory image to which control should be passed if one of its slaves is suspended. When a slave is suspended, the boss's machine registers at the time of the suspension are stored in its Communications Area, and the boss is set active executing at the address stored in the Interrupt word. Information about the reason the slave was suspended, and the address its computation had reached at the time of the suspension, are also placed in the boss's Communications Area. At the same time, the Interrupt word in the boss's description block is cleared.

If the slave is suspended at a time when a boss's Interrupt word does not contain an address to pass control to, this is treated as an error for the boss. The Error Handler is invoked and an error message is passed to the boss indicating that one of its slaves was suspended. As above, the boss is also give information about why the slave was suspended, and about its state at the time of suspension.

There are supervisor functions available to the boss which will allow it to examine the contents of the slave's description block and to read and write the slave's Communication Area. The supervisor function which permits a process to write another process's Communications Area has been modified so that when transmitting process is the boss of the receiving process, the name of the transmitting process is not recorded in the slave's Communications Area, as is normally done. (The boss can, of course, place this information where it would normally go, if it wishes to make it appear that it had sent the slave process a message in the normal manner.) Also, the boss is allowed to transmit information to the slave process's Communications Area at any time, without the restrictions that usually apply to the process-to-process communications mechanism.

It is evident that a boss process can get into trouble if it is not careful how it handles its slaves. It is advisable for a process with more than one slave to ensure that only one slave is unsuspended (i.e., either active or blocked) at a time. The boss process should set its Interrupt address before it removes a slave from the suspended state. In some circumstances it might be desirable to allow several slave processes of one boss to be unsuspended at the same time, and queue up reports of suspension so that the boss will not be interrupted at an undesirable point in its activities. This is not done in the IDA Operating System.

A technique has been used for some time on several different machines with memory protection consisting of a base register and length register. This technique was first publicly described in a paper by Hargraves and Stern [4] (see also [5]) for the GE 600 machines, and was there called the squeeze technique. In these machines, the base register and the length register (which we will refer to as the bounds registers) define the memory address space of a process. The squeeze technique consists of a supervisor call by means of which a process could cause the bounds registers to be set to a subset of the address space of the process and then have control passed to a point in the subspace. When the code that was executing inside the subspace of the original address space reach the point where attempted to make a supervisor call or do any other action which would cause activity on the part of the supervisor, such activity did not take place. Instead, control was returned to the original process with the bounds registers reset to the original address space. By this means the process could act as the operating system of a program. That is, the process would reserve part of its address space to hold the program, and contain code simulating the desire operating system in the remainder of the address space. The most important difference between squeeze mode and the boss-slave relationship described here is that the boss-slave mechanism permits the controlling code and the controlled code to execute as independent processes.

These processes can be scheduled independently by the supervisor, and the boss process can go about various kinds of activity while its slave process is in existence. The squeeze mechanism simply makes one part of the code being executed in a process dependent and under control of another part of the code within the same process. It does not provide the degree of independence achievable with separate processes, though it does protect the controlling code from the controlled code.

Uses of the boss-slave relationship

The most obvious use of the boss-slave relationship, and indeed the one that was the original motivation for its implementation, is in debugging. No debugging code except code necessary to generate breakpoints (which can be done simply by generating an error of a standard form) need be included in the slave's memory image when a program it is a executing is being debugged. Thus, the slave's memory image can look almost exactly the same during debugging as it will when a normal (non-slave) process is executing it. This has several advantages compared with other debugging techniques. One is that under some circumstances the program being debugged may have size requirements which are incompatible with the introduction of a debugging package. Even if debugging packages could be incorporated without trouble, as they might be in some virtual memory systems, the process would have to do I/O operations of some sort to report debugging information, and this may interfere with the performance of the program being debugged. And there may be other undesirable effects of the debugging package on the performance of the program being debugged if the debugging package operates as part of the memory image of the process executing that program. Finally, in some systems it may be difficult to protect the debugging package itself or the machinations of the other code in the process.

A word about breakpoints is in order. Breakpoints are an essential element of interactive debugging. These are achieved for a boss which is debugging a slave by planting code which will generate an error where a break point is desired. Some machines allow the setting of a harder register specifying a breakpoint when a processes is placed in execution, so that one breakpoint could be obtained without modifying the slave's memory image. But if several breakpoints are desired there does not seem to be any alternative to modification of the slave's memory image. Therefore, it is still possible that the performance of a program while it is being debugged may be different from its performance under normal execution. However, the interference of the debugging activities is generally greatly reduced when most of the debugger code resides in the boss, in contrast to the potential interference when the debugging package is part of the same memory image as the program being debugged.

The debugging of debugging packages is facilitated by the boss slave mechanism. Given an initial debugging tool using this relationship, more advanced debuggers can be developed by using the initial debugger in a process that is the boss of a slave executing the new debugger. This slave will itself be the boss of a process executing a test program for debugging the debugger. The boss-slave mechanism is also particularly valuable for

debugging in the case where two or more processes are executing programs which require them to communicate with each other.

A use of the boss-slave relationship similar to debugging occurs in the analysis of the behavior and performance of processes. Such things as the number and types of supervisor calls, statistics on the use of files, and amount of interprocess communications, and time between I/O operations are easily gathered. Such information may be quite difficult to gather in other ways.

Since the boss has general control over slave, it can limit the slave's activities. This leads to a very important use of the boss-slave relation to enhance the security of an operating system. Using this relation, it is possible to run secure processes in a basically insecure system. That is, even though most of the programs run by the processes in the system might be able to violate the desired rules for access to and modification of information in a system, a boss process could execute a program in such a way that it could not violate the intended security restrictions in the system. This comes about as follows.

The boss process can limit the interactions of the slave process with the supervisor as severely as it desires. In particular, it can check all attempts on the part of the slave process to access information via the operating system, and decide whether they are valid or not. Such checking is performed in addition to checking in the operating system itself. This transforms the security problem from one in which the entire functioning of the operating system, or those element of it which conceivably could affect the ability of processes to access information stored in the system, must be verified, to a problem of verifying that the boss-slave mechanism works properly, and that the code the boss is executing actually performs the checks it is supposed to. Since the boss program can be tailored as specifically as desired to the acceptable activities on the part of the slave is controlling, the task of verifying that it will perform correctly maybe much easier than the task of verifying that the operating system protection and security mechanisms as a whole perform properly.

A process which is controlling another process for security purposes must be constructed carefully to ensure that the desired security is really achieved. An indication of the subtleties that may be involved as shown by the following example.

Suppose process A creates process B at its slave. If process B is allowed to create process C as its slave, then process A must never allow process B to release process C to run in an uncontrolled manner. That is, process A must carefully control process C as well as process B. This is not difficult to do once necessity is recognized. It is easy to ignore this necessity, however. In general, in security, there is more danger from things which are overlooked that from things that are not done right.

Since the boss can control the slave completely, it can be viewed as an extension or a replacement for the operating system, as far as the slave process is concerned. Both notions may have interesting consequences. It is possible for the boss process to simulate some entirely different operating system from the one that normally is seen by processes.

If this is done, the slave can run programs written for another system (i.e., programs which make supervisor calls that are appropriate for the operating system being stimulated by the boss). The slave process it can self be the boss (and operating system) for another slave process, and so on. The similarity between this idea and the recent developments in virtual machine systems is clear (see [6], for example).

It is interesting to consider new ways of designing operating systems which might take further advantage of the boss-slave relationship. If the boss-slave relation were implemented as a basic part of the operating system, it might be possible to reduce the size and complexity of supervisor by incorporating some of the functions normally carried out by the supervisor ass part of the code of a boss process. In such systems, processes might come in pairs - one element of the pair being the boss, and the other being the slave. This would make the boss in effect part of the supervisor of the system and the code that executes would be that necessary to provide control and support for the slave processes associated with it. The code the boss process executes may depend on the tasks or functions of the slave process, the rights slave process has within the system, and on other factors. It is often suggested that the supervisor of the operating system should be as small as possible, containing the fewest number of functions necessary to support the activities of the system. The suggestion here provides a possible mechanism for helping to reduce the size and activities of a highly privileged supervisor.

A similar notion that just described is to consider families of processes, each directed by a boss. An example of a possibly interesting family is a family of processes to manage a database. They could communicate with other processes in the system by means of interprocess messages. One boss could be involved to verify the accuracy of messages passed back and forth, and to check the activity of individual processes in the database family at critical points in their activities.

In a system in which processes can be released by their bosses and in which bosses can later take control of processes again, an interesting form of monitoring a process can be implemented. In such a system a process could be monitored when desired by a boss and allowed to freely execute without monitoring at other times. There is obviously some overhead to the boss mechanism unless various facets of this activity are implemented quite efficiently and it may be inappropriate in many cases to continuously monitor the activity of every process in the system. However, it would be appropriate to monitor the activities of some of the processes in the system for random intervals in order to verify that they're not attempting to do illegal things with respect to the rules of the use of the system by the processes. This would introduce an element of uncertainty about whether a process's activities would be noticed when it attempted to do things. This might discourage somebody desiring to violate the rules for use of the system, since it might introduce a greater probability than now exists in most systems that the activities of such processes would be detected. In addition, processes can be made slaves of a boss in order to monitor their activities with other information became available which indicated such monitoring might be advisable. It should be observed that when a boss process is used to monitor a slave process's activities, the activity of the boss process occurs in response to the activity of the slave process. This contrasts with other monitoring situations in which

a monitoring process must actively seek information rather than being driven by the activities which is supposed to monitor.

REFERENCES

[1] Hansen, P. B., The nucleus of a multi-programming system, Communications of the ACM, 13, 4, April 1970, pp. 238 - 250.

[2] Gaines, R. S., And operating system based on the concept of a supervisory computer, Communications of the ACM, 15, 3, March 1972, pages 150 - 156.

[3] Lampson, B. W., A scheduling philosophy for multi-processing systems, Communications of the ACM, 11, 5, May 1968, pages 347-360.

[4] Hargraves, R. F., Jr., and Stephenson, A. G., Design considerations for an educational time-sharing system, AFIPS Conference Proceedings, 34, 1969,Spring Joint Computer Conference), pages 657-664.

[5] Detlefsen, G. D., Direct use of old computer software under a new operating system by simulation of the old operating environment, MSc. Thesis, Dartmouth College, Hanover, New Hampshire, 1968.

[6] Goldberg, R. P., Architecture of virtual machines, AFIPS Conference Proceedings, 42, 1973, pages 309-318.