[redacted] Institute of [redacted]

Project MAC

Machine Structures Group No. 9.

[redacted] Group No. 9 [redacted]

# Processes, Spheres of Protection and Independent Computations.

[redacted]

Earl C. Van Horn

Table of Contents

[redacted]

A time-shar[redacted]grammed computer [redacted]. This means that the system is executing conc[redacted]tly several [redacted]dent computations. The key word here is "independent". It is [redacted]tant that we have a precise notion of what we mean when we speak of [redacted]endence among computations. This memo will explain a particular [redacted]int toward the concept of independence and explore some of the [redacted]usions to which we are led when we adopt this viewpoint.

This memo will consider only totally independent computations, that is, [redacted]tations which do not communicate with one another at all. The structure [redacted] dynamics of communicating computations will be treated in a subsequent [redacted].

In a multiprogrammed system the various physical entities of the [redacted] are, at any instant, assigned among several independent computations. [redacted]ample, one computation might have assigned to it a processor, a [redacted] of core memory registers, a tape drive, a printer, and several [redacted] on a drum.

The following four aspects of independence have been discussed in the [redacted]erature and elsewhere.

1. A computation must not write into an independent computation.
2. A computation must not be written into by an independent computation.
3. A computations must not read an independent computations.
4. A computation must not be read by an independent computation.

These same four aspects can be defined more formally and precisely as follows.

1. A computation must not alter any physical entity assigned to an independent computation.
2. A computation must not have any of the physical entities assigned to it altered by an independent computation.
3. A computation must not alter any of the physical entities assigned to it as a function of the state of physical entities assigned to an independent computation.
4. The state of a computation's physical entities must not be used by an independent computation to alter the latter computation's physical entities.

A brief examination of these four points reveals that, in a system where independent computations are treated uniformly and symmetrically, points 1 and 2 are equivalent, and points 3 and 4 are equivalent. Hence, we really have only two distinct aspects to the problem of total independence among computations, namely those aspects given in points 1 and 3 above.

There is no doubt that point 1 ought to be included in any formal definition of the concept of independence. However, there has been some debate as to whether point 3 is essential to the concept of independence. The assertion to be made here is that point 3 is also essential to the concept of independence. If point 3 were omitted, then an undebugged computation which happened to read an independent computation could not be counted on to produce the same results when executed again and again

with the same input data. One important difficulty with this situation has been pointed out by Prof. Corbato, namely that such behavior appears exactly like transient hardware failure to the user and is extremely difficult to diagnose. This is not the only difficulty with the omission of point 3, however. In practice, it is impossible to determine if a program is completely free of bugs. Therefore, in practice, if point 3 is omitted, no computation could be counted on to produce repeatable results.

It is an essential property of digital computations that, barring hardware failure, the output of a computation ought to be solely a function of the input to that computation, and of the initial state of the physical entities assigned to that computation. This property is so important that we will give it a name: the repeatability property.

In order to guarantee the repeatability property to the computations in a multiprogrammed system, we must include point 3 in the concept of independence. The adoption of point 3 means also that the state of one computation is kept secure from access by independent computations. This feature is important in some applications of time-sharing.

## A Mechanism for Achieving Independence among Computations

### A. Restrictions on Processors

Let us now examine the mechanism by which a computation can alter or sense the state of a physical entity. Let us assume that the system consists of one or more physical devices called processors, and that no physical entity of the system can be altered or sensed except as a result of the action of a processor. That is, a processor can perform altering and sensing operations.

itself, and/or it can initiate these operations in other devices.

Under this assumption the statement that a computation alters or senses a physical entity means that a processor executing an instruction sequence of that computation causes that physical entity to be altered or sensed.

We had previously seen that in order for two computations to be independent, each must be restricted in the set of physical entities which it can alter or sense. We now have transformed the problem of restricting a computation into the problem of restricting a processor. That is, in a multiprogrammed computer system, which can concurrently execute independent computations, each processor must be restricted in the set of physical entities of the system which it can alter or sense;, a set which which each processor is restricted is just the set which is assigned to the computation whose instruction sequence that processor is executing.

B. Logical Entities

It is desireable to digress slightly at this point in order to introduce and explain the distinction between logical and physical entities. From time to time and for various reasons, the system reallocates physical entities to serve different purposes. This reallocation is carried on asynchronously with respect to the computations using the entities being reallocated. So that the instruction words and other logical entities of a computation need not be modified when such a reallocation takes place, a physical system device, called the name-address map, is interposed between a computation and the physical entities that it must ultimately alter or sense. The computation

[redacted] behaves as if it is altering or sensing logical entities whose [redacted]signations, or names, are in variant under the above-mentioned [redacted]location. When a computation wishes to alter or sense a logical entity, [redacted] name-address map determines which physical entity currently enbodies [redacted] referenced logical entity, and the system actually alters or senses [redacted] physical entity thus found. Examples of logical entities are [redacted]ory words and logical tape drives. The corresponding physical entities [redacted] memory registers and physical tape drives.

Up to this point we have talked about retricting computations, and hence [redacted]essors, to physical entities. We can now, without contradicting what [redacted] have previously said, talk about restricting computations and processors [redacted] logical entities as well as physical entities. This change is simply a [redacted]tional convenience rather than a basic alteration of concepts, since [redacted] logical entity is associated with a single physical entity through the [redacted]-address map.

C. Processes

We now wish to define the concept of a process. Consider first an [redacted]truction sequence, which is a set of instructions. Each of these [redacted]tructions specifies an ordered set of actions to be performed upon [redacted]cal or physical entities. In addition, each instruction specifies a [redacted] of possible successor instructions. When an instruction is executed, [redacted] than one member of its set of possible successor instructions is [redacted] to be the next instruction to be executed, on the basis of the state

of the computation at the time of execution. If the successor set of an instruction is empty, or if no next instruction is chosen during its execution, then that instruction is said to be a process-terminating or terminal instruction. Every instruction sequence has exactly one first or entry instruction.

A process is defined to be a site of execution within an instruction sequence. For a more vivid understanding of this definition, consider that an instruction sequence can be depicted in terms of a graph, or flow-chart, as is exemplified in Figure 1. Here the nodes denote instructions, and therefore symbolize both a series of actions as well as a successor decision. The branches leading from a node link the node to all of its possible successors. The half-arrow labeled "E" leads to the sequence's entry instruction, while the half-arrows labeled "T" eminate from terminal or potentially terminal instructions of the sequence.

As the instruction sequence of Figure 1 is executed by a processor, we can imagine that the node being executed is lighted up. As successive instructions are executed, successive nodes become lighted, in the manner of an animated electrical sign. After watching the lights go on and off for a while, we begin to perceive that something is moving through the graph, lighting lights as it goes. Or perhaps we perceive that there is only one light and that this light is itself moving through the graph. In any case, this something that moves through an instruction sequence we choose to call a process.

The problem of defining a process is similar to the problem a physicist faces when he attempts to define the notion of a wave. Clearly something moves, and yet everything tangible stays in one place (on the [illegible]te of interest).

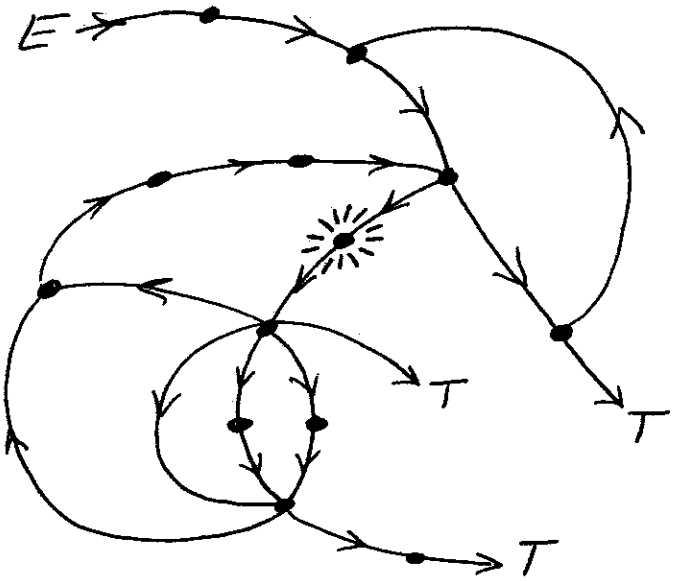


**Figure 1.** An instruction seq uence, which is being traversed by one process.

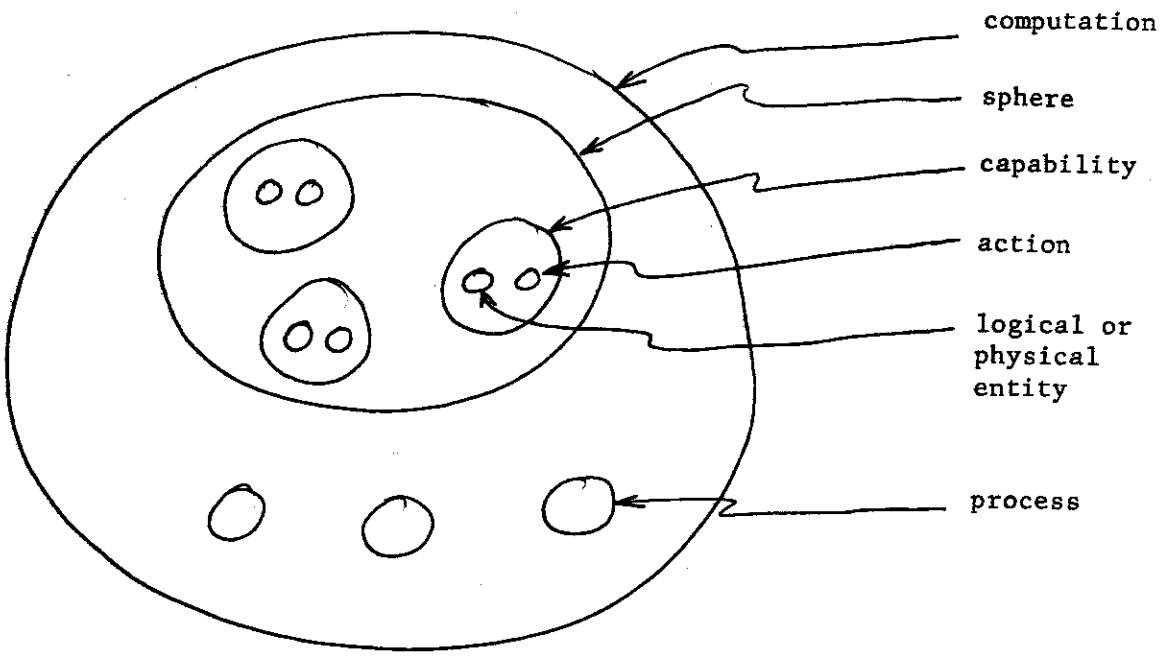


**Figure 2.** A computation.

The following questions can now be asked: "Why introduce the idea of a process? Why not say that it is a processor which is moving through the instruction sequence?" The answer to this question is that there is no relationship between the number of processors in the system, and the number of processes that are active in the system at any instant. It is true that a process does not advance through a sequence except through the functioning of a processor. But a processor can temporarily suspend a process in order to advance some other process. In this case, the suspended process sits motionless in its sequence, with its light still on, as it were. In fact, in a multiprogrammed system, one of the principal tasks of the system is to schedule the available processors among the processes to be executed.

D. Restrictions on Processes

We have assumed that we can place restrictions on computations by placing restrictions on processors. Let us now assume that restrictions are actually placed on processes, and that a processor always observes the restrictions of the process that it is executing. Under these circumstances, we can discuss the problems of changing the restrictions on computations by considering only the problems of changing the restrictions on processes. We thus avoid considering such irrelevant details as the number of processors in the system, and the manner in which they are commutated among the processes to be executed.

E. Spheres and Computations

In order to discuss process restrictions in a more succinct and precise way, we introduce the concept of a sphere of protection, which is often called simply a sphere. A sphere is a set of capabilities

A capability is a set consisting of a physical or logical entity, and an action which can be performed on that entity. The existence of a capability within a sphere means that a process associated with that sphere can carry out the action of that capability on its physical or logical entity. The following are examples of capabilities: (1) the ability to access a certain segment for reading only, (2) the ability to transfer control to a certain relative name within a certain segment. (3) the ability to type a character on a certain logical teletype, and (4) the ability to place new capabilities within a certain sphere.

We now are in a position to formally define a computation. A computation is a set which consists of a number of processes, and a sphere. The sphere of a computation contains only the capabilities which the system allows the processes of that computation to have. Figure 2 shows the set structure of a typical computation. A process is a member of only one computation, and likewise a sphere is a member of only one computation. A sphere is associated with each process. This sphere is the sphere belonging to the computation of which that process is a member.

F. Mechanizing Spheres

In order to achieve independence among computations, the system must, each time a process attempts to alter or sense an entity, ask, "May this process do this?" In order to answer this question the system must determine if the required capability is in the sphere associated with that particular process. If it is, then the system can perform the requested action. If it is not, an exceptional condition exists which causes some executive process outside of the current computation to be resumed. All of this means that the system must have available to it

[redacted]tion describing the contents of the spheres that exist in the [redacted].

[redacted] checking of process actions that was mentioned above can be [redacted] by system hardware, or by a supervisory program. CTSS uses both [redacted]. Memory references are checked using an adder and a boundry [redacted]. I/O requests are checked by the supervisor program.

[redacted] the system must be able to ascertain which sphere should [redacted] to perform a check, there must be associated with each [redacted] a designator called a sphere number, which identifies the [redacted] of the process.

[redacted] check on a process' actions invokes an examination of the [redacted] sphere. For example, a boundry register can be loaded from [redacted] contained in a sphere, and subsequent memory references can [redacted]lly invoke a check using this boundry register. The boundry [redacted] is then a kind of look-aside for the sphere. A boundry register [redacted] an extraction of certain information contained in the [redacted]. If the capability from which the boundry register was loaded [redacted] subsequently changed, the boundry register must be reloaded or [redacted] as having illegal contents.