

## *The Plessey System 250*

### *4.1 Introduction*

The second attempt to build a capability-based hardware addressing system was made by the Plessey Corporation in the United Kingdom. Plessey's System 250 [England 72a, England 74], examined in this chapter, was not only the first operational capability hardware system but also the first capability system sold commercially.

Initially the Plessey 250 was not designed as a capability system. Maurice Wilkes of the University of Cambridge had learned about capabilities during several visits to the University of Chicago and had included a capability description in his book on timesharing systems [Wilkes 68]. Wilkes sent a draft of his book to Plessey's Jack Cotton who incorporated capability concepts into the System 250. Because of the strong resemblance between the System 250 and the Chicago effort, Bob Fabry (who had worked on the Chicago Magic Number Machine) later acted as a consultant for Plessey.

Unlike the systems examined thus far, the Plessey 250 was not intended to be a general-purpose timeshared computer. Instead, it was designed as a highly reliable, real-time controller for a new generation of computerized telephone switching systems [Cosserat 72, Halton 72]. The reliability goal was very stringent: mean time between failures of 50 years [Hamer-Hodges 72]. Meeting this goal required that the system be easily configured, tested, and modified while operating in the field. Service improvements or performance upgrades would have to be performed while the system was operational. Such

needs led to a multiprocessing design that allowed connection of many processors and memories, as well as traditional and specialized I/O devices.

Although capabilities were used primarily for memory addressing and protection in the Plessey 250, the designers viewed the capability mechanism as a means of restricting the effects of faulty hardware and software components. Fault isolation was a major concern in a multiprocessing environment where several processors had access to a shared memory. One faulty processor could potentially damage another processor's computation. Capability addressing facilitated sharing among processors, while also restricting each processor's domain to the segments for which it possessed capabilities. The Plessey 250's designers also found that capabilities were useful in structuring the operating system [England 72b, Cosserat 74]. Layering and data abstraction were important aspects of the Plessey operating system design.

#### **4.2 System Overview**

The multiprocessing architecture of the Plessey 250 allows connection of up to eight processors with up to eight storage modules through separate per-processor data paths. Each storage module consists of up to 64K 24-bit words. Multiprocessing is symmetric, and any processor can perform any function if another processing component fails. Peripherals are connected and controlled through interfaces that allow the addressing of devices as memory. That is, device registers can be read and written by standard LOAD and STORE instructions, and no special I/O instructions are needed.

The Plessey 250's hardware and operating system support a segmented memory space. A segment can contain capabilities or data, but not both. The system has a general register architecture with eight 24-bit data registers (D0-D7) and eight 48-bit capability registers (C0-C7). To access data in a memory segment, a program must load one of the capability registers with a capability for that segment. Programs can freely copy capabilities between capability segments and capability registers using standard hardware instructions.

#### **4.3 Capability Addressing**

A Plessey 250 capability permits its possessor to access an object in the system, where an object is a logical or physical resource. The most basic object is a memory segment, and

hardware instructions can operate directly on segments through segment capabilities. Capabilities can be stored in capability segments or capability registers, as noted above. For each program, one of its capability registers (C6 by convention) points to a *Central Capability Block* for the program. The Central Capability Block is a capability segment that is the root of a network of program-accessible segments. The closure of this network completely defines the program's execution domain.

A capability in the Plessey 250 has two formats, depending on whether it is stored in a capability segment or a capability register, as shown in Figure 4-1. When stored in a 48-bit capability register, a capability contains three fields:

- A *base address*, which contains the primary memory location of the segment. (The high-order bits specify the storage module or interface, and the low order bits specify the storage element within the module or interface.)
- A *limit*, indicating the size of the segment.
- An *access rights* field, specifying the type of operations permitted on the segment by the owner of the capability. (The six unary-encoded access rights are: execute, write data, read data, enter capability, write capability, read capability.)

This 48-bit capability format, which includes a memory address and limit, is used only when capabilities are loaded into capability registers. When stored in a capability segment, a capability is 24 bits and contains only the access rights field and an index into a central system data structure, the *System Capability Table* (SCT). Each processor has an internal register that contains the address of the SCT. The SCT, which corresponds to the CAL-TSS Master Object Table, holds the base and limit information for all memory segments. In this way,

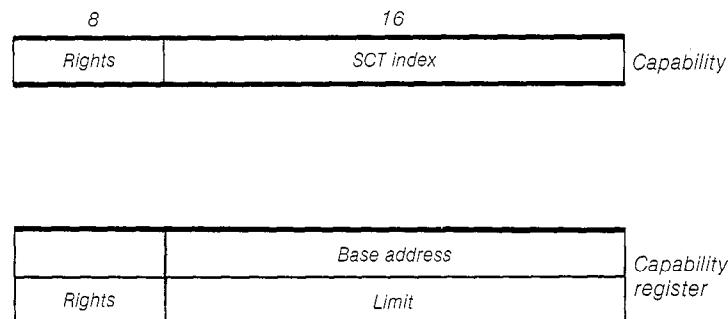
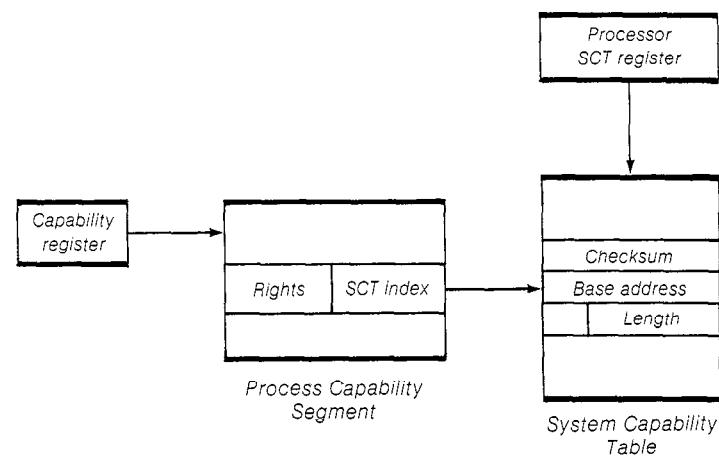


Figure 4-1: Plessey System 250 Capability Formats

physical addressing information is centralized and relocation of segments is simplified. There is one SCT entry for each object in the system. Because access rights for an object are stored in the capabilities, different processes can possess capabilities permitting different access rights to the same segment.

A program executes a LOAD CAPABILITY instruction to transfer a capability from a capability segment to a capability register. Figure 4-2 shows how the capability register is formed. The hardware first examines the SCT index in the specified capability in memory. This index selects the SCT entry for the segment, which is three words in size and contains a 24-bit checksum and some special flag bits in addition to the base and length fields. The 48-bit capability register is then constructed from the rights field in the capability and the base and limit information found in the selected SCT entry. The capability segment from which the capability is loaded must itself be addressed by a capability register, as shown in the top left portion of Figure 4-2.

When a program loads a capability register, the SCT index from the loaded capability is saved in a process-local data structure called the *Process Dump Stack*. The dump stack is a two-part process data structure containing fixed space for copies of data and capability registers, and a stack used to save information on procedure invocations. When a program executes a STORE CAPABILITY instruction to move a capability from a register to a capability segment, the saved SCT index is used, along with the rights field in the register, to construct the capability



in memory. The Process Dump Stack is thus used to hold the SCT index for each capability stored in the eight capability registers.

Because the SCT is shared by all of the processors, the relocation of a segment or the modification of any SCT entry must be synchronized. If several processors try simultaneously to modify a single SCT entry, the entry could be placed in an inconsistent state. In order to prevent this, the Plessey 250 has a facility to trap programs accessing a particular entry. Thus, a processor updating an SCT entry can prohibit other processors from using the entry until the modification is complete.

#### **4.4 Capability Register Usage**

Of the eight general-purpose capability registers, several have reserved uses. The first five capability registers, C0-C4, can be freely used by the program to address any memory segments to which the program has access. Register C5 points to a data structure used to store dynamically allocated elements associated with the current process execution. C6, as has been mentioned, contains a pointer to the process's Central Capability Block. This block defines all of the instruction, data, and capability segments associated with the current process. Register C7 contains a capability for the currently executing code segment.

In addition to the eight program-accessible capability registers, each processor has five special-purpose capability registers. These registers hold capabilities that address the following segments:

- The Process Dump Stack that contains backup register values.
- The System Capability Table that contains base/limit values for all storage segments in memory.
- The Start-up Block used for restarting the system after failures.
- The System Interrupt Word that indicates what devices need attention.
- The Normal Interrupt Block that contains device interrupt information.

#### **4.5 Inform and Outform Capabilities**

The Plessey 250 operating system provides a virtual segment interface to programs; that is, a program can address its segments independent of whether they are located in primary

or secondary memory. Secondary storage is totally transparent to the program. The operating system determines which segments are held in primary memory and which are held on disk storage. When a program attempts to access a segment that is not in primary memory, a trap occurs and the operating system then loads the segment from disk.

Each segment has an associated disk address that is assigned when the segment is created. A segment's disk address is used as its unique identifier, because two segments cannot have the same disk address. When a program creates a new segment, the operating system assigns the secondary storage address for the segment, allocates an SCT entry for the segment, and returns a capability for the segment to the program. The operating system initializes the SCT entry to indicate that no primary memory has been allocated. When the program first attempts to reference the segment, a trap occurs and the operating system allocates primary memory and stores the memory address in the SCT entry.

Because all segments on the Plessey 250 are potentially long-lived, the SCT could grow to enormous size if it had to address every segment in existence. To constrain the size of the SCT and maintain high memory utilization, the Plessey operating system allows SCT entries to be reallocated. At different points in its lifetime, an object may be addressed by different SCT entries. If a segment has not been referenced for a long period of time, the segment can be moved to secondary memory (an operation known as *passivation*), and its SCT entry can be used to address a newly created segment. Later, if the passivated segment is needed, it can be returned to primary memory and an SCT entry (most likely a different one) is allocated.

Reallocation of SCT entries is complicated by the fact that capabilities in memory contain SCT indices. If a segment's SCT entry is reallocated while capabilities for that segment are still in use, those capabilities would erroneously address a different segment. Thus, an object's SCT entry cannot be changed as long as capabilities that address the object are in memory.

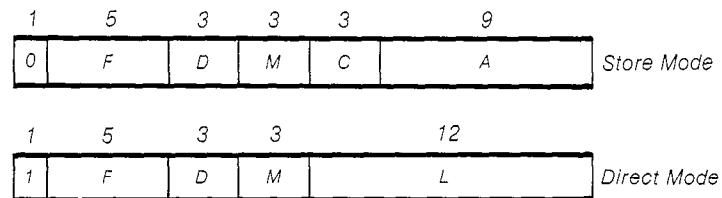
To allow SCT entries to be reallocated, the Plessey operating system uses a different format for capabilities that are stored on disk. Capabilities in primary memory are known as *inform* or *active* capabilities; these capabilities contain an SCT index. Capabilities in secondary memory are known as *outform* or *passive* capabilities; each of these capabilities contains a unique identifier, which is the object's disk address. When a

capability segment is moved from primary to secondary memory (or the reverse), the operating system changes the form of all the capabilities in that segment.

By changing capabilities from inform to outform, the operating system reduces the number of active references to the SCT. When a segment is passivated, its SCT entry is retained as long as active capabilities exist for that segment. If a segment stays passive for a long time, it is likely that the capabilities for that segment will eventually be passivated also, allowing the SCT entry to be reused. A special operating system process, called the garbage collector, periodically searches the capabilities in primary and secondary memory. The garbage collection process will cause an SCT entry to be deallocated if no active capabilities exist for that entry or will cause an object to be deleted if no capabilities exist at all for that object.

#### 4.6 Instructions and Addressing

A Plessey 250 instruction occupies a 24-bit word and is represented in one of two formats, as shown in Figure 4-3. The first bit of the instruction selects the instruction mode. *Store mode* instructions are used to access storage locations. The instruction specifies a capability register addressing the segment, a 9-bit offset into the segment, and an optional index register



*F* Function (operation code).

*D* Data register.

*M* Data register to be used as address modifier (index).

*C* Capability register.

*A* Address offset.

*L* Signed literal (if *L*=0 then *M* defines the second register of a two-register instruction).

Figure 4-3: System 250 Instruction Formats

modifier. The primary memory address for the operation is calculated by adding the base address contained in the capability register to the sum of the 9-bit literal and the index register contents. This address is validated using the limit field in the capability; the type of access requested is verified against the capability access rights field. *Direct mode* instructions do not require memory access and are used for loading a 12-bit literal or for register-to-register operations.

#### **4.7 Protected Procedure Calls**

The Plessey 250 System, unlike most traditional computers, has no privileged mode of operation. The operating system relies only on the protected procedure mechanism for its protection. This mechanism is available to any process and allows a process to add to the facilities supplied by the standard operating system.

A protected subsystem is built by creating a Central Capability Block in which the subsystem will execute. The Central Capability Block serves the same function for the subsystem as for any process: it contains capabilities for code, data, and capability segments available to the executing process. Some of the capabilities in the Central Capability Block are *execute* capabilities for the procedures that implement subsystem services. To make these procedures accessible, the subsystem passes an *enter* capability for its Central Capability Block to appropriate users. The possessors of the enter capability can call any of the procedures defined by execute capabilities in the block, but cannot access capabilities in the block.

To call a protected procedure, a process executes a CALL instruction, specifying an enter capability for a Central Capability Block and an offset in that block. The offset must locate an execute capability for a procedure to be called. The CALL instruction saves the instruction pointer and registers C6 and C7 (defining the Central Capability Block and Current Code Block) on the Process Dump Stack. Register C6 is then loaded with a capability for the Central Capability Block specified in the call; the C6 capability is given *read* access, permitting the called procedure access to *any* of the objects addressed by the central block. Register C7 is loaded with the capability for the instruction segment containing the procedure specified in the call.

Thus, the called procedure executes in its own domain as defined by its Central Capability Block. It is protected from

the caller, and the caller is protected from the procedure. A RETURN instruction restores the process to the previous domain by restoring the state of C6, C7, and the program counter from the stack.

#### 4.8 Operating System Resource Management

The Plessey 250 operating system is constructed as a set of protected subsystems that manage various types of *resources*. A segment is one type of resource that users can create and manipulate through capabilities. Other logical resources, such as files and interprocess communication ports, are also accessed by capabilities. Unlike segment capabilities, which are operated on by hardware instructions, logical resource capabilities are enter capabilities that allow the user to request services for the resource.

The resources supported by the Plessey 250 operating system are:

- storage segment
- process
- user
- job
- text file
- symbol directory
- data stream
- synchronizing flag

The last resource type listed, the synchronizing flag, is used both for interprocess communication and for synchronization. Processes that share capabilities for a flag can send messages to the flag or wait for message reception. At any point, a flag can have either a message queue or a queue of processes waiting for new messages. Processes can also wait on multiple flags for one of several events to be posted.

Users gain access to operating system services through a *Central Facilities Block* that contains enter capabilities for system resource allocation routines. Using these routines the caller can create any of the supported system resources. The creation routine returns an entry capability for the resource that can be used to manipulate it.

The actual representation of a resource is defined by the Central Capability Block pointed to by the enter capability returned to the user. The Central Capability Block contains execute capabilities for procedures that manipulate the resource.

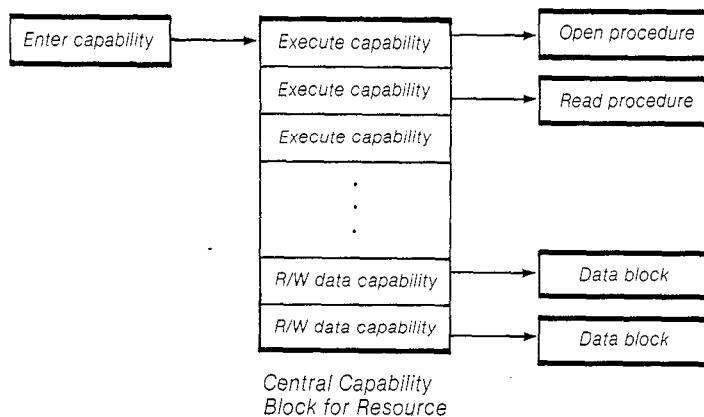


Figure 4-4: Protected Procedure Resource Subsystem

It also holds capabilities for segments that contain data structures describing the state of the resource. For example, Figure 4-4 shows a Central Capability Block created for a single file object. The Central Capability Block contains capabilities for file procedures and capabilities for file data segments. Note that a separate Central Capability Block will exist for each resource (e.g., file) in the system; however, all resources of the same type will share the same code segments.

#### 4.9 Input and Output

The use of capabilities in the Plessey 250 I/O system is similar to capability usage in storage accesses. Input/output devices are controlled by special device registers that exist in the physical address space. To access device registers, a process must have a capability for that memory space. For each device, one device driver process possesses capabilities for the device registers. This process can execute on any processor and still perform its I/O functions.

Any processor must be able to handle device initiation and completion. Because of this requirement, standard interrupts are abandoned in favor of a polling scheme using shared memory. Approximately every 100 microseconds, each processor examines certain I/O status words that are addressed through two of the five special capability registers (the System Interrupt Word and Normal Interrupt Block). The contents of these locations indicate whether or not any action needs to be taken and on behalf of what device. Other processors must be locked while the examination is made.

Several facts make the Plessey System 250 an important computer system:

1. It is the first functioning computer to use capability addressing.
2. It is the first capability-based computer produced by a commercial manufacturer.
3. It is designed to meet critical real-time performance and reliability needs.
4. It applies capabilities to a multiprocessor environment.

The Plessey 250 is similar to both the Chicago Magic Number Machine and the CAL-TSS system. The use of capability registers as user-loadable segment/base registers is borrowed from the Chicago project, while its addressing resembles the CAL-TSS mechanism. When combined, these features result in a capability design with the following attributes:

1. When stored in user segments, capabilities do not contain physical addresses, but instead contain an index into a central mapping table.
2. Capabilities can be stored on disk and are converted to a different form when copied to disk.
3. A segment is represented by a unique identifier, which allows conversion between inform and outform capabilities.

Because capabilities in primary memory do not contain physical mapping information, they are small and can be compactly stored. Only when a capability is loaded into a register is it expanded to full 48-bit form. The disk address and disk number for a segment provide a unique name for the segment. Capabilities stored on disk contain a unique name, while capabilities stored in primary memory contain a table index. Plessey addressing differs from the CAL-TSS scheme, in which both the capability and the Master Object Table entries contain a segment's unique identifier.

Primary memory addresses are only stored in the SCT and in the capability registers of executing processes. When an executing process is pre-empted, its capability registers are not saved. The Process Dump Stack contains the SCT index and access rights for each capability register, from which the register can be regenerated when the process is activated. Therefore, to relocate a memory segment, the operating system need only search the System Capability Table and the current process capability registers for any active segment addresses.

The decision to handle virtual segments and provide a mechanism for storing capabilities on disk greatly simplified the design task and avoided many problems encountered in the CAL-TSS system. The system does not need special naming mechanisms for short-term objects that have second-rate status. All objects are potentially long-lived. Allowing long-lived objects makes garbage collection a necessity, and the Plessey system has a background process responsible for deallocating storage for segments with no remaining capabilities to address them.

The Plessey 250 uses capabilities to simplify multiprocessing. All processors in the system share a single primary memory space. A single table shared by all processors, the System Capability Table, contains primary memory addresses for all segments. Because a process's address space is defined by capabilities that refer indirectly to this table, a process can address its segments from any processor. No special action is required on the part of a processor to initialize a process's memory environment.

Capabilities also aid software error detection. Each process possesses capabilities for only those segments absolutely needed for its function. A process cannot address data outside of its domain; therefore, any errors are limited to that domain. Errors are frequently caught by the addressing mechanism, either as illegal accesses or segment length violations.

A new concern created by capability addressing is the maintenance of capability integrity. On a standard virtual memory system, for example, a 1-bit error in the transmission of a process virtual address is not likely to affect data outside the scope of the process. An error in the transmission of a capability, however, can affect any process in the system. Thus, all hardware involved in holding or transferring capabilities must be error-checked carefully.

The Plessey System 250 combines hardware and software support to provide a uniform view of system resources. All resources are addressed by capabilities; hardware executes operations directly on segment resources while software executes operations on other resources. From a program's point of view, all resources are addressed and manipulated in the same way. In the Plessey resource model, each resource in the system is represented by a Central Capability Block and addressed by an entry capability. The Central Capability Block defines the data segments that contain the state of the resource and the procedures that can manipulate the resource. Procedures are shared among all instances of objects of the same type. The entry

capability to a resource's capability segment permits calling of the resource manipulation procedures, but prohibits direct access to the resource data segments.

Because the operating system is implemented as a collection of resources and protected procedures, it is relatively easy to extend the operating system in a uniform manner. New protected procedures can be created and addressed through the System Capability Table. Such procedures can make new types of resources available to programs.

As implemented, the Plessey 250 protected procedure call has one weakness. Although a protected procedure call causes a domain change, the called procedure still has access to any capabilities left in registers C0 through C4 by the caller. Likewise, the capabilities left in these registers by the called procedure when it returns are available to the caller, presenting a potential security violation. The tradeoff is one of performance, because the registers are an efficient mechanism for passing parameters between a calling and called procedure. Procedures concerned with information leakage can explicitly clear these registers; however, that is an unusual burden to place on the caller of a procedure.

Finally, the Plessey 250 system integrates capability usage into the I/O system in a consistent manner. This is possible because of the memory-like nature of the I/O interface and because of the requirement for processor-independent I/O. However, since I/O devices are forced to be slaves, their power is limited and additional strain is placed on the processors using them.

The Plessey System 250 was not meant to be a general-purpose multi-user computer system but, rather, was intended for a very specific product area. The targeting of the product to a limited role probably provided the key to any success the System 250 has had—its simplicity. The Plessey 250 uses a small number of simple mechanisms to provide for protection from and isolation of failure. The Plessey System 250 is still in use today in military communications systems in the United Kingdom.

#### **4.11 For Further Reading**

The principal descriptions of the Plessey System 250's hardware and software are provided by [England 72b, England 74] and [Cosserat 74]. Several papers on Plessey 250 can be found in *The Proceedings of the International Conference on Computer Communications*, October 1972, and in *The Proceedings of the International Switching Symposium*, June 1972.

