

Table of Contents

The Kernel	127
Queues	131
The BUFFER Data Structure	131
The Queue Header	132
The PUT Subroutine	134
The GET Subroutine	135
The Q.REAR Pointer	135
Concurrent Processes	138
The Process Control Block	138
The CP Current Process Pointer	140
The READY Queue	140
Requesting Kernel Services	144
The Kernel Service Request Handler Routine	144
Coding Kernel Service Requests	147
Context Switching	149
The PUSH and POP Subroutines	149
The SW Kernel Service Routine	153
The Null Process	154
Semaphores	157
The P Kernel Service Routine	158

The V Kernel Service Routine	160
Mailboxes	162
The SEND Kernel Service Routine	163
The RECV Kernel Service Routine	164
Process Creation and Destruction	167
The PCBS Free PCB Mailbox	167
The CRP Kernel Service Routine	168
The STP Kernel Service Routine	169
The EOP Kernel Service Routine	169
Reentrant Programs and Multiple Processes	172
Initialization	175
The Initialization Process	175
User Process Initialization	176
Kernel Initialization	178
Debugging the Initialization	179

1. The Kernel

The kernel is a small set of subroutines which can be used to provide an environment in which real-time, multiple process applications may be executed. The term "kernel" refers to the fact that these subroutines constitute the inner-most level of software in a simple layered operating system. The kernel contains routines necessary for the creation, scheduling and destruction of processes, process synchronization, and interprocess communication.

The kernel manipulates variables which are global throughout the entire system. For this reason, it is imperative that only a single sequential section of code, either process or interrupt service routine, be allowed access to the kernel at a time. Otherwise the kernel would be prone to the same race conditions that it is intended to prevent.

To insure that only one program can access the kernel at a time, interrupts are automatically disabled whenever the CPU is executing code inside the kernel. This permits the kernel subroutine that is called to execute to completion without interruption. Since, in this implementation, there is only a single CPU which fetches and executes one instruction at a time, no other program can be executing code in the kernel simultaneously.

Some computers have more than one CPU or "engine" sharing memory and executing instructions concurrently. Examples range from the DEC Rainbow personal computer, which contains both a Zilog Z80 and an Intel 8086 microprocessor, to the IBM 3031AP mainframe, which contains two 370 engines. For such machines, our simple technique for enforcing exclusive access to the kernel is not sufficient. Machine instructions which permit busy waiting must be used to serialize kernel access. This also assumes that some sort of hardware memory interlock, even on multiprocessor memories, is available.

Even our technique of disabling interrupts has a flaw. Some peripheral devices are smart enough to access and modify the processor's random access memory directly, rather than requiring the execution of machine instructions by the CPU. This method of I/O data transfer is called Direct Memory Access or DMA (on DEC systems it is often referred to as Non-Processor Request or NPR). A DMA device could read or overwrite kernel data structures even

