Vediamo la funzionalità delle Istruzioni Privilegiate

Immaginiamo di avere un insieme delle istruzioni che la macchina può interpretare: tale insieme può essere suddiviso in due sottoinsiemi, uno di essi conterrà le Istruzioni Privilegiate e l’altro le istruzioni Non Privilegiate.   
Cos’è questo privilegio? È uno stato che viene registrato nel registro di Stato del Processore (attraverso un bit o eventualmente più, se ci sono più livelli nel privilegio, nel registro di Stato). Prendiamo il caso più semplice: immaginiamo di avere un solo bit dedicato a identificare il privilegio nel Registro di Stato -> 0 vuol dire assenza di Privilegio, 1 vuol dire presenza di Privilegio.

Le istruzioni non privilegiate sono SEMPRE eseguibili (indipendentemente che il bit di privilegio nel registro di Stato sia 0 o 1). Le istruzioni privilegiate non sono sempre eseguibili, ma solo quando il processore si trova in uno stato privilegiato (Ossia solo quando il bit di privilegio è uguale a 1). Quando il bit nel registro di Stato è uguale 0 e il processore prova a eseguire un’istruzione privilegiata avviene una segnalazione di errore, segnalata attraverso una trap.

Abbiamo già introdotto il meccanismo di Interrupt e Trap e il meccanismo di memoria a virtuale a Segmentazione/Impaginazione (a noi interessa principalmente la Segmentazione).

Questi ingredienti messi insieme ci permettono di far funzionare il nostro processore sotto il controllo di un sistema operativo, senza perdere l’efficienza nell’esecuzione del codice.

Le interrupt e le Trap sono inviate al processore, tuttavia la loro gestione non può avvenire a RAM vuota, cioè prima/durante l’accensione del calcolatore (perché gli handler sono come funzioni salvate in RAM). Neanche il meccanismo di Paging può avvenire all’accensione della macchina, perché le tabelle di traduzione risiedono in memoria RAM, quindi durante l’avvio devono ancora essere scritte.

Per andare a scrivere nella RAM quando il meccanismo della memoria virtuale non funziona correttamente (perché non è avvenuto il bootstrap) serve poter aggirare il meccanismo di virtualizzazione del processore (ossia aggirare l’MMU) e andare a inserire i valori direttamente in RAM tramite indirizzi fisici. Se si riesce a far ciò il processore potrà funzionare normalmente, quindi questa “scorciatoia” usata in fase di avvio è proprio il caso che venga cancellata: ridando il controllo della conversione degli indirizzi all’MMU si eviterà che un qualsiasi programma possa sporcare le zone di memoria fondamentali.

Per questo motivo è importante avere il meccanismo delle istruzioni privilegiate: durante la fase di avvio, le istruzioni privilegiate accedono alla memoria RAM attraverso indirizzamento fisico. Dopo che è avvenuta la fase di bootstrap, il bit di privilegio nel Registro di Stato viene portato a 0 per impedire l’esecuzione di altre istruzioni di questo tipo (quindi il privilegio delle istruzioni privilegiate è di bypassare l’MMU).

Quindi, di solito, durante la fase di bootstrap il processore parte col bit di privilegio uguale a 1 (avviene quindi un reset del processore). Attraverso le istruzioni privilegiate vengono inizializzate le porzioni della memoria che contengono gli interrupt e i trap handler e le tabelle di traduzione. Avvenuta questa inizializzazione il bit di privilegio viene portato a 0. Se si provasse successivamente ad eseguire un programma che contiene istruzioni privilegiate scatterebbe la segnalazione di una trap: ciò potrebbe portare all’aborto del programma, se però l’esecuzione di tale istruzione fosse necessaria, la trap causerebbe lo switch temporaneo del bit di privilegio da 0 a 1 per permettere l’esecuzione. Questo è permesso perché il nostro modello di funzionamento è quello di avere un Nucleo di Sicurezza (Security Kernel) che serve per completare la gestione della sicurezza del sistema (tale Nucleo di Sicurezza comprende anche i gestori degli interrupt e delle trap).

Di tale Nucleo di Sicurezza ci si deve quindi fidare, per questo motivo va debuggato il più possibile fino a renderlo praticamente perfetto (o il più sicuro possibile). Se tale Nucleo di Sicurezza presentasse delle debolezze o degli errori potrebbe causare gravi problemi di funzionamento o danneggiare la macchina.

Ci sono poi le Applicazioni, che contengono istruzioni NON privilegiate e che sono bloccate in caso contrario: in tal caso vengono fatte passare nel Nucleo di Sicurezza che gestisce la situazione (in genere sospendendo l’esecuzione dell’Applicazione). Ciò rende ammissibile avere errori nelle applicazioni, basta che il Security Kernel sia stato programmato bene. Tutto ciò avviene grazie alle Trap, perché procedono a bloccare l’esecuzione di un Applicazione e fanno portare il bit di Privilegio a 1, permettendo l’esecuzione delle istruzioni del Nucleo di Sicurezza.

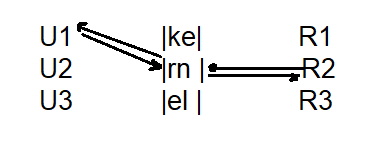
Non ha quindi senso avere un meccanismo della memoria virtuale se non ho il meccanismo delle istruzioni privilegiate, così come non ha senso avere il meccanismo delle istruzioni privilegiate senza il meccanismo delle trap.

Un Kernel minimale, che faccia solo la parte di virtualizzazione dell’hardware, è detto MicroKernel. Un tale Kernel, sebbene non presenti errori, permette di fare ben poche cose: Più features si aggiungono a un sistema operativo e più cose è permesso fare con esso, tuttavia aumentano anche i possibili errori nel codice, e quindi aumentano le vulnerabilità. Quindi, realizzato un Kernel, è conveniente cercare di semplificarlo per trovare eventuali errori e sistemarli.

Per implementare le altre funzionalità che non sono state inserite nel nucleo di sicurezza vengono fatti girare in background anche dei Server (per esempio nel caso del MicroKernel tutte le funzionalità non fondamentali ma molto utili agli utenti vengono implementate separatamente come Server -> ciò torna utile perché in caso di errori in un Server questi hanno conseguenze spiacevoli ma non catastrofiche, come invece lo sarebbero errori nel Kernel). Un esempio di server può essere il filesystem.

Parliamo della sicurezza di un Sistema: Essa può essere quantificata sulla base di alcuni principi fondamentali, chiamati “Principi di Denning”:

1. Separazione tra utente e risorse: Quando un Utente (o meglio un processo avviato da un’utente) vuole accedere a delle risorse, il Kernel deve mettersi in mezzo e decidere se recuperare le informazioni richieste dall’utente e restituirgliele o no. Analogamente se si prova ad utilizzare la memoria RAM il Kernel si mette in mezzo e permette all’utente l’accesso solo a determinate pagine di memoria.



1. Trovare l’anello debole della catena: Se la sicurezza nel sistema dipende a un tot. Di aggeggi presenti nel sistema, il grado di sicurezza complessivo dipende dalla (cioè è uguale a) componente più debole. È quindi necessario agire innanzitutto su un punto debole e ottimizzare/debuggare prima quello (agire prima sulle componenti più forti è uno spreco).
2. Principio di Economia/Ridondanza di Controllo: è una strategia per arrivare alla realizzazione di un sistema sicuro. “Economia” Consiste nel semplificare le cose il più possibile in modo da avere la maggior consapevolezza possibile del funzionamento del sistema (Anziché scrivere un programma lungo 10.000 righe di codice ne scrivo uno equivalente o quasi di poche centinaia di righe: su questo secondo programma datami a disposizione una sufficiente quantità di tempo posso garantire di risolvere tutti i bug). La “Ridondanza” vuol dire aggiungere cose in più ma che siano tutte indipendenti tra loro, e se uno dei sistemi non sembra abbastanza sicuro aggiungere un ulteriore meccanismo di controllo solo per quel sistema (es: meccanismo di autenticazione in due passaggi nell’home banking). Questi due meccanismi vanno in senso opposto: uno vuol dire **togliere** delle cose e sfruttare la maggior comprensione del funzionamento per rendere tutto più sicuro, il secondo vuol dire **aggiungere** meccanismi sperando che per qualunque minaccia almeno uno dei sistemi di protezione funzioni (e in maniera consistente). La Ridondanza è come se, avendo un anello debole nella catena, anziché sostituirlo io aggiunga altri anelli a supporto di quello del materiale peggiore.
3. Il Principio del Minimo Privilegio (abbastanza complicato da mettere in pratica): Consiste nel fare in modo che tutti gli utenti abbiano un livello di privilegio minimale e dare privilegio di accesso a una risorsa dolo se questa è strettamente indispensabile. Questo perché se nel codice del sistema sono presenti errori, ogni cosa che concedi in più all’utente diventa una possibile vulnerabilità (un esempio può essere di dare accesso a certe aree solo in lettura e non in scrittura, onde evitare modifiche indesiderate). Nella realizzazione a livello hardware minimo privilegio vuol dire che le istruzioni privilegiate devono essere limitate al massimo (un MicroKernel è costruito in osservanza del principio del minimo privilegio).

Oggi la maggior parte dei Calcolatori moderni segue i requisiti di virtualizzazione di Popek e Goldberg.

Virtualizzazione non vuol dire soltanto la scrittura di un Kernel o di un MicroKernel. Potremmo parlare di virtualizzazione nel senso di realizzazione di macchine virtuali. Con questo termine si intende tipicamente l’uso della tecnica di virtualizzazione che abbiamo detto prima (MMU, sistema delle trap/interrupt con rispettivi handler e il meccanismo delle istruzioni privilegiate), per semplificare l’uso del sistema da parte di più programmi. L’idea è stata sviluppata un po’ di anni fa e oggi trova una sua applicazione su larga scala: essa permette di usare un certo sistema fisico (Cpu, RAM, Dispositivi I/O) e attraverso l’introduzione di un software che fa uso delle cose dette precedentemente simula la presenza di uno o più processori non necessariamente uguali al processore fisico. Quindi data una macchina fisica, noi possiamo avere una o più macchine virtuali (dotate del loro processore, della loro unità a disco, della loro RAM ecc.) che possono essere usate indipendentemente l’una dall’altra. Le componenti che vengono utilizzate per costruire le macchine virtuali sono dette Ipervisori (Hypervisors) e possono essere di diverso tipo: uno può fare da sostituto di un sistema operativo e far vedere una certa macchina virtuale all’utente, un’alternativa può essere di avere una macchina normale e avere un Ipervisore che mi fa vedere altre macchine virtuali.

Come fa un ipervisore a far vedere una macchina virtuale avendo virtualizzato una macchina fisica? L’idea è che il nostro nucleo (che può essere un microkernel o un ipervisore) usa dei meccanismi hardware per effettuarla: Quando il processore parte in modalità privilegiata inizializza lui i valori in RAM e, tolto il bit di privilegio, manda in esecuzione un altro codice non privilegiato che gli permette di riprendere il controllo della situazione attraverso gli handler quando avvengono degli errori. Una cosa che si può fare abbastanza facilmente è quella di programmare un interrupt data da un timer e far sì che dopo un tot di millisecondi ci sia un’interruzione. Dopodiché si fa andare avanti il programma applicativo e dopo un po’ lo si interrompe per far ripartire il nucleo: in Questo modo la nostra applicazione ha a disposizione una certa quantità di tempo (limitata) per poter utilizzare il processore. Così si possono simulare più processori virtuali: ognuno di essi ha una certa quantità di tempo per poter utilizzare il processore fisico (terminata la quale si passa ad eseguire un'altra macchina virtuale). All’aumentare delle macchine virtuali, la velocità di funzionamento sarà quindi proporzionalmente ridotta.

Il sistema operativo, dal canto suo, si aspetta di avere il bit di privilegio a 1, ma quando l’ipervisore prima di far partire il sistema operativo della vm azzera il bit di privilegio ciò causa la successiva interruzione dell’esecuzione tramite trap: quando entra in gioco il trap handler l’ipervisore può far finta di essere il sistema operativo e di eseguire al posto di esso l’istruzione privilegiata (il sistema operativo “ospite” viene quindi interrotto dalla trap). L’ipervisore fa quindi da tramite tra le risorse e il sistema operativo sulla macchina virtuale.