Laboratorio parte 2.

Questa settimana lavoreremo sul file hello1.c (l’evoluzione di hello0.c). Questa variante è incompleta e va completata da noi. Per capire quali sono le istruzioni mancanti è far sì che l’esecuzione di hello1 restituisca lo stesso output di hello0. La manipolazione da fare è riconoscere il primo carattere dopo uno spazio e renderlo maiuscolo (ovviamente può avvenire solo se tale carattere è una lettera, nessuna trasformazione è necessaria su numeri e segni di punteggiatura).   
L’implementazione della funzione deve riconoscere il primo carattere non spazio e fare la conversione in maiuscolo.  
Complicazione in più: cercare tra le diverse varianti possibili quella che richiede il minor numero di caratteri nella scrittura del codice sorgente. Tutte le istruzioni mancanti devono essere scritte sulla stessa riga senza andare a capo. Perde chi usa più di 110 caratteri in totale nella riga (compresi i due commenti che non possono essere eliminati né ridotti). È consigliato riutilizzare il più possibile codice già esistente (meglio sfruttare la stdlib in c), quindi per ottenere il risultato è necessario usare le funzioni della libreria: leggi la documentazione.

Parte la stringa, ci si chiede se il primo elemento è uno spazio o un carattere di tabulazione, se è così si passa al carattere successivo fintanto che non trovo un carattere diverso, a tal punto ci si deve chiedere se quello che si trova è un carattere alfabetico minuscolo oppure no (numero punto ecc.): se lo è si converte a maiuscolo e si stampa con un a capo in fondo.   
Chi trova un metodo per fare ciò in meno di 100 caratteri vince.

Abbiamo visto l’implementazione delle memorie Cache. Il loro nome deriva dal francese ed è infatti storpiato alla versione inglese (originariamente si pronuncia Caché), il suo significato è “nascosto” (perché, pur essendo nascosta, la cache permette al processore un più veloce accesso ai dati).

Rivediamo adesso i supporti hardware alla virtualizzazione delle macchine. Abbiamo parlato di Interrupt e Trap, di memoria virtuale (MMU, segmentazione, paginazione) e di Istruzioni privilegiate. Queste tre cose sono funzionalità offerte a livello hardware da tutti i sistemi attualmente in uso e servono per realizzare la funzionalità di virtualizzazione.  
Il processore di suo ha la capacità di essere avviato per eseguire un programma e va avanti seguendo un’istruzione dopo l’alta secondo il flusso di esecuzione e le istruzioni di salto, andando dalla prima alla ultima istruzione. Una volta avviata l’esecuzione di un programma non c’è possibilità di ripensamento (per passare a cose più urgenti), ma ciò è risolto dalle Interruzioni. Esse segnalano interventi esterni al processore: quando il processore riceve la segnalazione di un’interruzione interrompe il programma in corso e avvia l’esecuzione del gestore delle interruzioni (che fa parte del Nucleo del Sistema Operativo). Questo è il modo che viene utilizzato da tutti i dispositivi di input/output.  
Il meccanismo delle Trap è gestito in maniera simile, anche lì c’è un handler che fa parte del Kernel; tuttavia la Trap è sollevata da una situazione causata dal programma in esecuzione (quindi non da interventi esterni).

Come memoria virtuale abbiamo visto la tecnica della segmentazione, coadiuvata dalla tecnica della paginazione. Ad un programma vengono in genere attribuiti quattro o poco più segmenti (codice, dati statici, stack, heap). Il meccanismo della memoria virtuale richiede la presenza del meccanismo delle Trap, poiché è tramite esso che vengono segnalati eventuali errori di traduzione e quindi la segnalazione del programma di cercare di uscire dal proprio segmento. Tale traduzione viene svolta dall’MMU, che usa le tabelle di traduzioni presenti in RAM o, se disponibili, nella piccola memoria associativa che ha vicino, il TLB.  
Quando viene generata una Trap di questo tipo in genere viene fatto terminare il programma e si manda all’utente un segmentation fault.

Le istruzioni privilegiate si basano su un bit di stato che segnala la differenza dello stato tra privilegiato (durante l’avvio/bootstrap o durante la gestione da parte del Kernel) e quello normale della macchina. Lo scopo di queste istruzioni e quello di bypassare il meccanismo di traduzione, così che il sistema operativo possa inizializzare le tabelle di paginazione, di segmentazione e i gestori delle interrupt e delle trap tramite il sistema operativo. Queste istruzioni usano quindi indirizzi fisici e non virtuali. Dopo la fase di avvio il processore passa da una fase di privilegio a una di NON Privilegio e tutte le istruzioni successive potranno essere coperte in caso di errore e di uscita dai propri limiti: in questo modo un programma buggato lanciato in esecuzione potrà fare dei danni solo all’interno dei segmenti (liberi) che gli sono stati assegnati: avviene così la separazione degli spazi di memoria dedicati ai vari programmi ed è per questo motivo che è sicuro lanciare più programmi contemporaneamente.

Perché tutte queste funzionalità funzionino correttamente è necessario un buon sistema operativo. Per realizzare un tale sistema, che quindi deve essere sicuro, fondamentali sono i principi di Denning. In questo caso per sicurezza si intendono aspetti diversi:

Il primo aspetto che deve essere garantito è la riservatezza delle informazioni. Il sistema deve essere in grado di dare l’accesso ai dati alle persone che ne hanno diritto e soprattutto deve Impedire a chi non ne ha diritto di accedere ai dati. In genere tale forma di riservatezza è implementata tramite un sistema di riconoscimento dell’utente. Una versione minimale dell’identificazione sicura degli utenti è la conoscenza di una password segreta.

Un secondo aspetto richiesto molto importante è l’integrità. I dati devono essere mantenuti integri (memorizzato un file questo deve mantenere la stessa configurazione per quando lo si ricercherà nuovamente). L’integrità può essere a rischio quando qualcuno commette un errore che porta alla modifica di qualche dato: in questo caso a livello hardware l’integrità è garantito dal sistema di memoria virtuale.

Vi è poi come caratteristica l’affidabilità dei dati e del sistema. Un pc che non può essere acceso è inutile, così come lo è a batteria scarica: tutti i dati in esso memorizzati non sono disponibili. Per garantire questo aspetto la soluzione dipende dalla causa che provoca la mancanza di affidabilità: per ovviare alla batteria scarica, ad esempio, è necessario avere un caricatore per il calcolatore, per non rischiare di perdere i dati il sistema ti permette di fare dei backup. L’affidabilità consiste nell’avere a disposizione dei mezzi per riparare eventuali “guasti” del mio sistema.

Ci sono casi in cui l’affidabilità non basta, si scende nel concetto di disponibilità: essa indica lo stato corrente del sistema. Per garantire questo aspetto è necessaria la ridondanza; se si guasta il caricabatterie deve esserci la possibilità di comprare un altro caricabatterie che possa sostituire quello attualmente in uso.

Un sistema sicuro dovrebbe avere tutte quelle quattro caratteristiche. I principi di Denning sono un modo per ottenere quei livelli di sicurezza.

Il **primo principio** è il sistema di **virtualizzazione**. Si vede il sistema come una serie di utenti, che possono essere sia persone che avviano programmi che altri programmi, un nucleo di sicurezza (Kernel del OS) e delle risorse (memoria RAM, processore, stampante ecc.). L’idea è che gli utenti non accedano mai direttamente alle risorse del sistema, ma possono soltanto accedere al nucleo, che fa da intermediario tra l’utente e le risorse. Il risultato di ogni operazione viene quindi rimandato indietro all’utente sempre dal nucleo del sistema operativo (per questo motivo il principio di virtualizzazione è anche chiamato di “mediazione”). La memoria virtuale funziona esattamente così (il processore fa a utente e l’MMU fa da traduttore). Da notare che il nucleo della transazione (attraverso segnalazioni di Trap) può negare all’utente di ricevere i dati che sta richiedendo.

Il **secondo principio** è il concetto di **anello debole della catena**. Quando si progetta un Nucleo di sicurezza lo si fa introducendo una serie di componenti (ad esempio l’MMU, che però ha bisogno della memoria virtuale e del meccanismo delle Trap e delle istruzioni privilegiate); tali componenti sono realizzati in tempi diversi e talvolta da persone differenti. Se qualcuno commette un errore banale in uno dei componenti il livello di sicurezza complessivo è, secondo il principio dell’anello debole, equivalente a quello della componente peggiore: dunque il componente che presenta l’errore abbassa notevolmente il livello di sicurezza, non importa quanto gli altri siano efficienti. Se si deve effettuare un processo di debugging, si devono quindi spendere le risorse sul miglioramento della componente che fa più schifo (se ho una catena di acciaio ma ha un anello di plastica questa si spezzerà eventualmente all’anello di plastica).

Il **terzo principio** è quello di **Economia/Ridondanza di controllo**. Quando dobbiamo agire su un sistema operativo abbiamo due alternative. Una è quella della ridondanza: in una catena d’acciaio con un anello di plastica si aggiunge nella stessa posizione di quello di plastica anche un anello di legno (quindi è necessario spezzarli entrambi per spezzare la catena). In pratica, la ridondanza consiste nell’aggiungere misure di sicurezza che permettono di resistere a sollecitazioni di tipo diverso. Per esempio, si può aggiungere un lettore di impronte digitali come sistema di riconoscimento. Un eventuale attaccante dovrà così tener conto della presenza di più dispositivi di sicurezza.   
L’alternativa, ossia l’Economia di controllo è un approccio opposto: anziché aggiungere cose ne si tolgono. Si elimina tutto quello che non serve e si ottiene come vantaggio di semplificare il sistema. Quando un sistema è più piccolo e semplice diventa più facile modificarlo, capirlo, trovare eventuali errori e correggerli: il sistema diventa più sicuro perché si sono tagliati i bug (è la trasformazione dell’anello di plastica in uno di acciaio). Se un meccanismo di password è debole perché il controllo è progettato male, si riduce la complessità del programma fino a capire perfettamente come funziona così da poter implementare un algoritmo migliore.  
Tra le due alternative quale conviene scegliere? Dipende, se si hanno abbastanza tempo e risorse per riprogettare il sistema operativo conviene scegliere la strada dell’Economia, con l’aggiunta di eventuali altri servizi tramite server, altrimenti, per risparmiare, si segue il principio della ridondanza, programmando patch che, alla fine, dovrebbero riuscire a comprendere tutti i casi possibili di attacco e avere una risposta abbastanza sicura per ciascuno di essi.

L’**ultimo principio** di Denning è il **minimo privilegio**. Sebbene sia quarto nell’elenco è il più importante.