

Sistemi operativi anno 2021-2022 registrazione: 2° semestre

2022-02-05

slide 00-intro.pdf

Ieri abbiamo visto queste belle cose abbiamo visto cosa abbiamo fatto. Il Sistema Operativo è un livello di astrazione e l'importante è che fornisce un API in maniera tale che il processo utente venga confinato controllato in maniera tale che il processo corrente possa, anche qui c'è la metafora con il mondo reale, convivere civilmente con processi che condividono l'ambiente di lavoro. Il Sistema Operativo è un gestore di risorse, rende tutto più facile. Macchina Estesa, dipende dall'hardware, rende i programmi portabili. Abbiamo lavorato nello storia, volendo tornare qui (allo storia) un attimo perché "ciò che avviene in passato non è detto che non sia utile anche nel futuro". Questa (00-intro.pdf pag 22 Generazione Z (1955-1965)) che sembra un'architettura antica, desueta in realtà è usata ma non in sistemi general-purpose per esempio chi di voi ha giocato con un arduino sa che non si può dire che abbia un sistema operativo, ma proprio un monitor, oppure chiamato loader per l'arduino. Cioè l'arduino altro non è che un microcontrollore, adesso se ne usano di vario genere, compreso adesso c'è anche l'arduino con il nuovo chip della raspberry pie, comunque quello classico è un atmega

di qualche genere diciamo per molti arduini è il 328 e che sarebbe un debole sistema però per inserire, per caricare il programma avrebbe bisogno di un device esterno. L'arduino è di fatto un microcontrollore con in più un loader, un loader che viene precaricato, che viene mantenuto in una eeprom (più che in una eeprom in una non volatile... una eeprom). Arduino atmega non è una macchina di Von Neumann è una macchina di Harvard, quindi ha due memorie separate, una per i programmi e una per i dati. Nella memoria per i programmi che è

non volatile, per essere un arduino viene precaricato un piccolo programma, che può essere simile a questo monitor, che all'accensione o al reset per pochi secondi si ferma e guarda se sta arrivando dati dalla seriale emulata su USB e se in questi pochi secondi arrivano dati li interpreta come un programma da caricare, sempre nella parte programma oltre il loader. Quindi assomiglia molto a questo (gen. 2) vedete il compilatore Fortran se voi avete l'arduino e il primo programma tipico dell'arduino blink, viene caricato blink e fa quello che deve fare poi resettate/riaccendete/caricate un altro programma e fate altre cose. Quindi come vedete queste architetture ritornano in altri ambienti, non si può dire che sia un sistema operativo

è molto embrionale come sistema.

(va a pag 37 dicendo che abbiamo visto le cose in mezzo...)

Ecco abbiamo visto tante funzionalità che possono avere i sistemi operativi. I sistemi operativi si riconoscono dall'interfaccia delle system call dalla loro API. E abbiamo visto che ci sono delle funzionalità che ci possono essere o non essere e che se ci sono o non ci sono si evince dal fatto se l'interfaccia delle systemcall di questi sistemi prevede o meno funzionalità chiamate specifiche per queste funzioni. Se manca tutta la classe di systemcall per definire chi è l'utente del processo corrente e cambiare l'utente del processo corrente il sistema non sarà multi-user, ma sarà pensato per fare programmi senza protezione tra utenti e così via. Oltre alle cose classiche ci sono altre funzionalità del sistema operativo che possono esserci o non esserci e queste caratterizzano sistemi operativi per usi specifici, per esempio i sistemi operativi possono supportare o no il PARALLELISMO (pag 37). Cosa è il parallelismo? Lo dice la parola, la capacità di fare più operazioni contemporaneamente. Un processore standard normale, un singolo CORE, fa una cosa sola in un'unità di tempo. Quindi abbiamo visto che i sistemi operativi capaci di funzionare su singolo processore o eseguono codice del KERNEL o eseguono codice applicativo e possono dall'uno all'altro o da Kernel a user

caricando uno stato o da user a Kernel perché avviene una richiesta del processo a un altro evento asincrono. Come possono essere i sistemi paralleli? I sistemi paralleli possono essere Single Instruction Multiple Data (SIMD) detti anche sistemi vettoriali. Quindi questi sistemi fanno più cose contemporaneamente, ma in realtà eseguono la stessa istruzione su dati diversi, su vettori, su sequenze di dati in modo che l'operazione venga fatta nello stesso momento su tanti dati. O se no ci sono le Multiple Instruction Multiple Data (MIMD) che sono quelle che eseguono programmi differenti su dati differenti. Quindi per esempio se avete un sistema multicore, un sistema multicore è una macchina parallela di tipo MIMD. E voi conoscete dei sistemi SIMD? Li avete mai visti? Pensate che sono dei processori che per come sono fatti velocizzano le operazioni che devono essere fatte contemporaneamente su grandi quantità di dati. Non vi viene in mente un esempio? Bepo!

Studente: forse nella lavorazione delle immagini sui tanti pixel

Sì, e chi fa l'elaborazione delle immagini gestendo tanti pixel? Le SCHEDE VIDEO, e infatti queste schede video poi talvolta vengono utilizzate chiamandole con il nome di GPU (Graphics Processing Unit) per fare elaborazioni dove la stessa operazione deve essere fatta su grandi quantità di dati. Per esempio l'implementazione

dell'evoluzione di una rete neuronale. Per cui abbiamo un vettore che ci dà tutti gli stati di eccitazione dei vari neuroni, abbiamo la matrice con i pesi che ci indica quanto lo stato di un neurone debba propagarsi allo stato di un altro e per far evolvere la rete bisogna che questi calcoli vengano svolti su tutti i neuroni che compongono la rete neuronale. E quindi: una macchina SIMD è molto comoda per questo. Un altro modo di classificare macchine parallele è quello di guardare quanti CORE/processori ci sono. Ci possono essere sistemi a basso parallelismo, pochi processori in genere molto potenti, o sistemi massicciamente paralleli, un gran numero di processori che possono anche avere potenza non elevata

(pag 38) Un altro tipo di classificazione dei sistemi paralleli sono sistemi TIGHTLY COUPLED, quindi sono sistemi dove i core collaborano all'interno di un architettura che prevede il bus di comunicazione a memoria condivisa, è questo il modello dei sistemi Personal Computer Multicore, dove avete ... che ne so... 2, 4, 8 core, però questi CORE condividono il bus, la memoria. ok? E invece sistemi LOOSELY COUPLED sono sistemi dove ogni unità di elaborazione è un processore con la sua memoria (memoria privata), e i suoi canali di comunicazione ed è collegato con altri processori, secondo una determinata TOPOLOGIA. ok? Normalmente i sistemi tightly

coupled usano pochi processori, i sistemi loosely coupled usano tanti processori. Perché secondo voi non si fanno sistemi tightly coupled con 256 processori / core ? Prepol

studente: Perché più aumentano i processori più sarà il numero di processori che nello stesso momento chiederanno l'accesso alla memoria e quindi...

Esattamente! Perché essendo un bus a memoria condivisa diventa collo di bottiglia. Quindi non si ottengono prestazioni perché diventa elemento critico il bus perché è vero che si può spendere di più e tentare di ottenere dei bus più performanti più veloci però c'è un limite fisico alla cosa. Mentre invece ci sono state sperimentazioni, per esempio la CONNECTION MACHINE era una macchina che aveva 65.536 processori collegati fra loro con una topologia ad IPERCUBO. Che cos'è un CUBO?

"Non si capisce un cubo" :D no... (lasagna) ...

dimensioni	1	2	3	...	N
cubo	• - •	! - !	! - !	! - !	e poi potete continuare!

 Questo è un cubo. Ma questo è un particolare cubo.
E il cubo di dimensione 3. Esistono cubi di TUTTE le dimensioni. Il cubo di dimensione uno \bullet . Cubo di dimensione 2 \square . Cubo di dimensione 3 \boxtimes . E poi potete continuare. Come potete creare il cubo di dimensione $N+1$? Per fare il cubo di dimensione $N+1$ prendete 2 cubi di dimensione N e collegate tra loro i nodi corrispondenti. Quindi qui abbiamo un cubo di dimensione 1 \bullet , qui ne ho preso 2 \bullet e li ho collegati assieme \boxtimes . Qui ho preso 2 cubi di dimensione 2 \square e li ho collegati assieme \boxtimes . Se volessi passare alla dimensione 4 già ancora si può vedere nello spazio fate due cubi concentrici, uno dentro l'altro e collegate i nodi. OK? Questo è correlato ai numeri binari. Qui \bullet avete un bit 0 0 0 1. Qui \boxtimes avete fatto due copie e quindi il primo bit vi rappresenta a quale copia volete accedere e il secondo bit, il bit meno significativo è il bit all'interno del cubo precedente. Per fare un cubo in più il primo bit vi indica a quale cubo del livello precedente scegliete e poi avete l'indirizzo nel cubo più basso. Quindi collegando i processori secondo un IPERCUBO con 65.536 punti, vuol dire che è un cubo di dimensioni 16 (infatti $2^{16} = 65.536$). La cosa bella è che ogni processore

può parlare con ogni altro processore facendo non più di 16 passi, nonostante ci siano 65.536 processori.

La connection machine è fallito, perché secondo voi?

L'idea della connection machine era quella di prendere dei processori a bassa potenza e ne mettere insieme dei numeri enormi. (un esempio di loosely è la connection machine).

Anche se un processore non è una valvola, anche i processori hanno la loro possibilità di guastarsi, e mettendone insieme 65.536 uno ha la probabilità di guastare un processore 65.536 volte quella di un processore singolo. Questo è uno dei fattori e l'altro è mentre con un sistema / i sistemi che abbiamo / i nostri portatili che hanno N core questi sistemi vengono utilizzati spesso anche con programmi che non sono adatti all'elaborazione parallela, ma perché avete bisogno di fare più cose quindi mettete un processo che non è parallelo che funziona su un processore mentre un altro funziona in quell'altro per utilizzare proficuamente delle architetture di questo genere tutti gli algoritmi che andate a scrivere devono essere pensati per funzionare lì sopra.

Perché non ha senso mandare lì sopra l'esecuzione di un processo sequenziale imperativo standard.

Perché userà un processore e ne avete ... e andrà piano come un singolo processore. ok? Quindi comunque esistono sistemi loosely coupled

anche magari con meno processori e... o forse oggi rinascono ma rinascono nella forma di SISTEMI DISTRIBUITI (nag 40 poi torniamo a 39). Cioè invece di pensare a una macchina singola con tanti processori si dice "prendiamo tanti sistemi come sistemi singoli (con processore, memoria... come tanti personal computer) e poi facciamo in modo di considerare tutto questo enorme insieme di macchine convenzionali come un sistema unico". ok? Ed è l'architettura che stanno usando per fare i nuovi supercomputer. avete sentito Leonardo?... Se poi andate anche molto più vicino all'INFN in area morassutti (ci si arriva a piedi) avete in mente dove è il dipartimento di fisica? Se andate a vedere i siti online ci sono sistemi di elaborazione che i fisici usano tantissimo per fare elaborazioni delle tracce degli acceleratori, elaborano dati che vengono dal CERN. Vedete che lì avete distese e distese di "rack" con nei "rack" dei server, ma quei server non sono dei server singoli: quei server vengono visti come un unico sistema di elaborazione che coinvolge tutte queste macchine. E quindi se volete la differenza tra sistema massivamente parallelo (loosely coupled) e sistema distribuito è una differenza labile, il concetto è simile, solo che mentre

si pensava di fare una topologia, una struttura, una macchina specifica nei sistemi distribuiti vengono viste tante macchine convenzionali come un sistema unico. Qual'è il problema? qui (connection machine) veniva studiata una rete specifica per l'elaborazione, in questo caso tutta la topologia di collegamento, la parte di connessione tra processori viene fatta con sistemi specifici di rete. Qual'è il problema nei sistemi distribuiti? Siccome sono sistemi di rete e non delle architetture di bus interne come quelle loosely coupled, hanno maggiore latenza e questo si ripercuote sugli algoritmi e sullo studio di come sfruttarli appieno. Questo è più o meno una descrizione hardware di come sono fatte le macchine.

Come impatta tutto questo sui sistemi operativi?

(pag 39) Parliamo di sistemi paralleli, sistemi multicore, sistemi tightly coupled. I sistemi operativi più diffusi usano MULTIPROCESSING SIMMETRICO (SMP). Cosa significa? che tutti i processori vengono gestiti nello stesso modo. "Non c'è un processore più quale degli altri" citando Orwell :D. Tutti i processori eseguono una copia identica del sistema operativo e eseguono processi. Quindi qual'è il problema? L'abbiamo già in parte visto, che questi vari processori vivono come il processore singolo quindi eseguono processi, succede una cosa, viene chiesta una systemcall, arriva un interrupt, vuole eseguire il Kernel, esegue il Kernel e fa Però il problema è che se ci sono più processori

che vogliono eseguire il Kernel, questi processori hanno strutture dati in comune. E quindi occorre fare in modo che si coordinino nell'accedere alle strutture dati. E, ripasso dal primo semestre, siccome i vari core i vari processori hanno disabilitazione degli interrupt indipendenti occorre utilizzare altre davorerie, altre soluzioni per creare sistemi di mutua esclusione, ed è per questo che sono nati gli spin-lock. Quindi per fare dei sistemi operativi SMP occorre che i processori forniscano delle istruzioni per consentire spin-lock, quali test & set, atomic swap, e la coppia di istruzioni lock-load store-conditional che è la coppia di istruzioni che consente di fare test & set su processori RISC. I processori RISC non possono fare due cose in un colpo solo quindi sono due istruzioni, ma sono fatte in maniera che la seconda fallisce se è successo qualcosa tra la prima e la seconda. L'altra alternativa è MULTIPROCESSING ASIMMETRICO quindi si può pensare di avere un processore master che gestisce il Sistema Operativo, gestisce l'assegnamento dei processi ai vari processori, e gli altri processori sono solo slave che eseguono i singoli processi come viene indicato dal processore master. (pag 41) SISTEMI DISTRIBUITI invece si comportano in modo diverso, appaiono come un'unica macchina. Un esempio di Sistema Operativo Distribuito lo potete pensare come quello del nostro laboratorio. Se accedete al laboratorio esistono tante macchine, le riconoscete perché di fianco al login di ogni macchina

c'è scritto il nome alla quale vi collegate. Sono nomi di personaggi d'opera Però quando accedete (anche remotamente via SSH) alla fine non vi interessa niente a quale macchina voi avete fatto accesso. Il sistema è tutto allineato. Avete la vostra home directory dovunque voi siate (non come i sistemi delle aule, in cui abbiamo una home directory staccata in ogni aula quindi le nostre preferenze...). E ciò che non vedete è che ci sono dei sistemi di installazione, aggiornamento di tutto il sistema distribuito che sono condivisi. Quindi se c'è da aggiornare il sistema non è che i tecnici del nostro dipartimento vadano di macchina in macchina ad aggiornare il sistema, ci sono dei meccanismi per tenere tutto il sistema distribuito allineato con quello che serve. Quindi in questo caso quello che ci serve dal punto di vista dei Sistemi Operativi per dare questa situazione è ovviamente condivisione efficace del file system, poi occorre database condivisi per quello che sono utenti, password, gruppi e così via, possono esserci meccanismi per distribuire il carico fra computer se volete utilizzarli per fare elaborazione massiva/massiccia. (pag 42) Andiamo in un'altra classe di sistemi in realtà poco conosciuti, i sistemi REAL TIME, penso che abbiate sentito questo termine solo che di solito non vengono definiti correttamente. REAL TIME non significa veloce, può essere tranquillamente lento, dipende dal problema che dovrete risolvere. La definizione esatta di sistemi real-time la trovate qui (pag 42, i linki sono disponibili sulla pagina del corso):

Definizione: Sistemi real-time

Sono i sistemi per i quali la correttezza del risultato non dipende solamente dal suo valore ma anche dall'istante nel quale il risultato viene prodotto.

Vi dico che i sistemi real-time sono i sistemi per i quali il risultato è corretto non solo se è corretto nel proprio valore, ma se è corretto nel tempo in cui viene prodotto. OK? Quindi tecnicamente "se io progetto un sistema per dire ... devo trovare la risposta alla vita, l'universo in ogni cosa ma non ci deve mettere meno di 200 anni" è real time. Perché se quello mi risponde a 199 anni la risposta è sbagliata perché gli avevo creato un vincolo temporale, ora chiaramente questo è uno scherzo con citazione di Douglas Adams, ma ... perché ovviamente quello che interessa è avere un risultato entro un determinato tempo. Però non è detto che sia... l'importante è che sia deterministico, non ci interessa che sia veloce, che sia veloce abbastanza per il problema che andiamo a risolvere. (pag 43) Infatti abbiamo due tipi di sistemi real-time: HARD real-time e SOFT real-time. Quelli hard real-time sono quelli con effetti catastrofici se il vincolo temporale non viene rispettato. Purtroppo in questi giorni è tornata per vicinanza geografica alla mente il problema della centrale di Černobyl'. Una centrale nucleare ha un sistema di controllo, un sistema di controllo che come pensate è un sistema realizzato

con dei sistemi di elaborazione quindi materia nostra. Questo sistema non è poi... nelle centrali a Fissione... non è poi un sistema tanto elaborato perché ci sono dei meccanismi di rallentamento della reazione nucleare e quindi se la temperatura diventa troppo elevata occorre limitare, occorre ridurre la velocità della reazione. Nelle vecchie centrali avevano delle barre di grafite che venivano messe, inserite o estratte, dal nucleo di uranio. Ora però queste operazioni fatte nell'arco di decine di minuti, anche... non so esattamente i tempi ma non millisecondi, microsecondi. Quindi anche un errore potrebbe farcela per solo fare il calcolo necessario però deve essere garantito, perché se o non arrivo in tempo perché si blocca il sistema oppure non hanno previsto che poterà arrivare l'acqua del maremoto come FUKUSCHIMA gli effetti sono catastrofici, ma non sono le centrali nucleari gli unici ambienti. Se avete visto, spero da lontano, gli apparati della terapia intensiva, sono apparati con a bordo degli elaboratori. Non è il caso che questi sfapino le tempestiche delle loro attività, che gli infusori diano troppo o troppo poco farmaco o si blocchi tutto aspetti che qualcuno dia Ctrl+Alt+Delete, perché alla macchina si può fare ma alla persona collegata NOI Non solo ma se pensate agli zeri, ai grandi zeri di linee, spero che non

pensiate che la cloche del pilota sia collegata con dei carri agli alettoni. La cloche non è altro che un sofisticato joystick attaccato a dei sensori che danno input a un sistema di elaborazione, ed è il sistema di elaborazione che poi pilota tutte le superfici di volo. Tutti questi sono esempi di sistemi hard real-time.

I Sistemi Operativi per i sistemi hard real-time sono completamente diversi da quelli visti fin ora.

Anche i processi dei sistemi hard real-time non sono neanche scritti in C, sono scritti in linguaggi fatti apposta dove per esempio ogni loop ha un numero massimo di iterazioni consentite se no si chiama la routine di emergenza, perché non ci si può scherzare e tra l'altro non sono sistemi operativi in cui dici adesso lanci un processo. L'insieme dei processi che quel sistema deve elaborare viene stabilito a priori, viene calcolato lo Scheduler statico in modo che siamo sicuri che tutti facciano quello che devono fare, e lo si fa in condizioni non critiche, quando la centrale nucleare è ferma, quando il sistema di controllo della terapia intensiva non ha nessun malato collegato, quando l'aereo è a terra. A quel punto si carica tutto, si mette in funzione, magari si mette in funzione anche con sistemi replicati per full tolerance e si lancia tutto il sistema così com'è ok? Quelli che in realtà sono molto più diffusi sono i sistemi SOFT real-time. I Sistemi Operativi per Personal Computer

sono tipicamente SOFT real-time. Quindi se avete un processo che sta facendo rendering di un frame video, o ancora peggio di frame audio, che sono più fastidiosi gli audio del video, quelli con un sistema di priorità si fa in modo che raramente possono sbagliare la loro tempistica. Però capite bene che è una gamma di problemi completamente diversi. Tipicamente i sistemi SOFT real time sono best-effort cioè se voi prendete un sistema fatto benissimo, studiato con cura e lo sovraffarcicate state tranquilli che il vostro filmato salta un po' e l'audio è singhiorzante.

Comunque mi preme dirvi che real-time non è l'esecuzione veloce.

Così abbiamo finito i lucidi che abbiamo iniziato ieri (00-intro.pdf (registrazione: 2022-02-25 pt1 minuto 45:40)) e cominciamo a ripassare l'architettura (01-arch-hw.pdf) in particolare guardiamo quelle parti di architettura che ci interessano per lo sviluppo dei sistemi operativi. Vi segnalo <https://nandgame.com> questo è un gioco a livelli che insegna l'architettura degli elaboratori, come dicono i miei figli...? "Io ho incominciato il gioco e sono stato sopra tutto il pomeriggio perché non riuscivo più a sciararmi". Qual è la parola giusta? ... intrappa... Sei arrivato a costruire completamente dallo zero logica a un elaboratore capace di

eseguire programmi. Ritenni se vi piace quando lo avete provato (non fatevi adesso se no non stareste attenti alla lezione).

bot: nand to tetris

prof: non ho capito la battuta.

Questa (pag 2) è la nostra benemerita macchina di Von Neumann.

Forse si può dire "virale" del gioco... (registrazioni: 48:20)

C'è la CPU che ha all'interno l'unità di controllo e l'unità aritmetico logica. C'è un bus, attaccato al bus ci sono non i dispositivi ma i controller dei dispositivi e c'è la MMU (Memory Management Unit \Rightarrow unità di gestione della memoria) che collega alla memoria Principale.

Sai conosco il libro nand to tetris, se volete nand to tetris è molto più professionale e usa linguaggio di descrizione di hardware. Voi avete usato nand to tetris al corso (di architettura)? Il gioco mi consente di portare conoscenze meno formalizzate di quelle di nand to tetris anche a studenti della scuola secondaria (\Rightarrow 1° grado medie, 2° grado licei, itis...) ok.

(pag 3) Avete visto l'anno scorso architettura dei processori, concetti base relativi alla memoria e linguaggio assembly? Avete visto un linguaggio assembly?

Studente: quello di nand to tetris.

Consiglio, comunque bene o male, come effetto collaterale, quest'anno vedrete l'assembler di mips. Quando vedete in esecuzione il programma vedete la transcodifica

delle istruzioni mips. Tra l'altro il pezzo che fa il reverse-assembly delle istruzioni mips è proprio un pezzo della macchina Mmips che ho fatto con le mie manine... :D. ok.

(pag 4) INTERRUPT! Questo è un concetto chiave fondamentale e ancora oggi vedo che spesso anche all'esame viene sbraitato.

L'**INTERRUPT** è un meccanismo che permette l'interruzione del normale ciclo di esecuzione della CPU.

La CPU (Central Processing Unit) è qui mi appello alle conoscenze dell'anno scorso, miracolosamente se qualcosa non è chiaro fermatemi e rivediamo tutto quello che c'è da rivedere. Le CPU vivono la loro vita

caricando l'istruzione corrente, e come fanno a caricare l'istruzione corrente? Hanno un registro che si chiama

Program Counter o Instruction Register, sono sinonimi.

Lo buttano sul bus accendendo il bit del leggi. La memoria risponde mettendo sul bus dati l'istruzione.

L'istruzione viene caricata nel registro di decodifica.

A quel punto, passo successivo, il processore analizza l'istruzione caricata e (lo vedete benissimo anche su nand to tetris, o con l'architettura dei libri di Tanenbaum) e quello che fa è configura i vari componenti per fare/ eseguire l'istruzione voluta. Se pensate all'architettura di un processore, i vari bus interni al processore mi piace vederli come i binari dei modelli ferrovieri. "Dove vanno i dati?", "Dove vanno presi i dati?"

dipendono da come sono stati settati gli scambi.
Praticamente la decodifica dell' istruzione altro non è che fare in modo che dalla stringa dell' istruzione vengano settati gli scambi dei bus della CPU perché al passo successivo possa essere eseguita l' istruzione, nel caso alla fine l' istruzione fa lo store dei risultati, e così via.
Quindi come il motore a 4 tempi: aspirazione, compressione, scoppio e scarico questa (CPU) fa:

carica l' istruzione, decodifica l' istruzione, esegue l' istruzione e poi torna da capo. Quando arriva in fondo e vuole tornare da capo fa un passaggio in più. Dice:

"È avvenuto un INTERRUPT?", o meglio l' interrupt alla fine arriva al processore come un filo o più fili. Uno di questi bit è a 1 o sono tutti a 0? Se vede che c' è un interrupt il processore è organizzato per prendere l' indirizzo particolare, che è stato messo apposta per questa funzionalità, e metterlo nel program counter. Quindi in realtà il processore al ciclo successivo avrà fatto un salto a sua insaputa. Il programma non ha detto in quel momento di fare il salto, ma siccome è avvenuto un interrupt viene cambiato il program counter, ovviamente il valore precedente del program counter lo deve salvare da parte se no non riesce a tornare indietro.
Se gli INTERRUPT sono MASCERATI questa operazione non viene fatta. E quindi anche se ci sono

questi fili accesi per dire "guarda che c'è un interrupt" il processore continua con l'istruzione successiva, e così via. Perché vengono introdotti gli interrupt?

L'interrupt fondamentale, l'interrupt per il quale è stato creato il concetto di interrupt è l'interrupt di Input / Output (I/O).

Ricordate che gli interrupt o sono quelli veri INTERRUPT HARDWARE sono comunicazioni che vanno dai controller dei dispositivi al processore, ok? Quindi il caso più tipico, l'abbiamo visto anche con la funzione scenica, la teatralizzazione di inizio corso, l'interrupt di I/O più importante è quello che viene mandato dai controller per dire "ho finito l'operazione" di I/O.

Facendo così il Sistema Operativo può attivare tante operazioni di I/O nei vari controller e poi mettersi a fare altre cose perché quando uno dei dispositivi avrà finito l'operazione manderà l'interrupt. Poco disinteressarsi del fatto che ci siano operazioni di I/O in corso perché verrà PRONTAMENTE AVVERTITO, prontamente è importante, nel momento della fine dell'operazione di I/O. Ma domanda tipica

è "Senza interrupt si può fare multiprogrammazione?"

Sì, ma la farà inefficiente. Perché l'unico modo che c'è per vedere se l'unità di I/O ha finito è chiederglielo.

Via bus il processore può sempre dire "Hei! Unità di I/O hai finito? Cosa stai facendo?". Ma facendo così si dovranno utilizzare solo tecniche di POLLING: "Hai finito?" "Hai finito?" "Hai finito?"... In un sistema monotask, quando mettete micro-sleep nell'

arduino per fare il bit di accendi / spegni, quello perde tempo, quello sta lì a ciclare a dire "Hai finito?" "Ho finito?" ... non ha altro da fare! Ma se io usassi dei meccanismi del genere (*polling*) su processori da Personal Computer ovviamente perderei un sacco di prestazioni. O perdo prestazioni, chiedendo troppo spesso "Hai finito?", o perdo prontezza cioè mi accorgo dopo del tempo che l'unità di I/O ha finito, alla fine perdo prestazioni lo stesso. Ecco gli INTERRUPT possono essere sia hardware che software, se volete in molti autori, anche a me piace chiamare gli interrupt hardware INTERRUPT, gli interrupt software TRAP. Gli interrupt software è il meccanismo per l'idea di usare lo stesso meccanismo degli interrupt per poter codificare degli eventi causati dal processo in esecuzione. Quindi occorreva una maniera per poter fare in modo, ve l'ha detto anche ieri, che il processo in esecuzione vive in ambiente confinato e controllato. ... far in modo che se il processo in esecuzione tentava di fare una corbelleria DAVOLERIA... :D: divisione per zero, accesso in memoria dove non può, uso di istruzione illegale... inventarsi tutto quello che può fare un processo di male... Non scoppiasse tutto, ma attirasse l'attenzione del Kernel (del S.O.). Allora l'idea è stata "noi abbiamo già un meccanismo che serve per attirare l'attenzione del

Sistema Operativo quando avviene gli interrupt del controller, gli interval hardware. Utilizziamo lo stesso strumento per gestire quando succedono degli errori, delle situazioni anomali causate dal processo in esecuzione. OK? Quindi gli interval hardware sono generati dai controller e normalmente sono attività asincrone che non sono relative al processo in esecuzione in questo momento. Gli interval software invece utilizzano lo stesso meccanismo degli interrupt per errori del processo in esecuzione.

(pag 5)

bot: quindi quando un programma interrompe la sua esecuzione per core dumped ad esempio l'esecuzione viene terminata da una trap?

Sì, se c'è segmentation fault... tralasciando allora... ponete di fare un bel programma che prende un indirizzo 0, assegna un puntatore a 0 e tenta di stampare ciò che c'è a quel puntatore.

Tra l'altro ho notato, non me lo ricordo, ma ho visto che pmpcs usa il valore di NULL a -1, secondo me sarebbe bene cambiarlo e lo riportiamo a 0 perché è quello standard voluto dal C. Secondo me se cambiate la costante e ricompilate tutto dovrà funzionare tutto ugualmente, senza che succeda niente. Lo cambiamo tra fase 1 e fase 2 così vediamo che tutto funzioni. Dicono segmentation Fault lo

causa il processo in esecuzione quindi "io sono il processo in esecuzione. Sono in user-mode, posso fare tutto quello che mi consente le istruzioni aritmetico logiche e l'accesso alla mia memoria. Faccio questa istruzione in cui tento di accedere alla memoria con indirizzo sbagliato. Il ciclo di CPU carica decode esegue, l'esegui crea la trap. Al passo successivo viene generata la trap e come se fosse un interrupt si passa all'esecuzione di un indirizzo del Kernel, che quindi incomincia a... Il Kernel si risveglia, praticamente, normalmente è dormiente e viene svegliato quando avviene un interrupt per una trap. Il Kernel parte, guarda quello che è successo, dice ah guarda "il processo X ha tentato di accedere ..." e lo vede guardando i registri della CPU. A quel punto dice "cosa devo fare adesso?". Ed è il Kernel che mette in piedi, parliamo di UNIX, quell meccanismo di mandare al processo il SEGNALE. Quindi il Kernel dice ah, siccome io sono UNIX e c'è scritto nelle regole che per mandare "se un processo fa un errore di indirizzamento nella memoria devo dare segmentation fault" il Kernel chiama quella funzione per mandare un SEGNALE quella chiamata dalla systemcall Kill di un altro processo. D'esecuzione non viene terminata dal trap, il passaggio è più lungo, il trap causa la generazione del segnale.

Non è detto che una segmentation Fault termini il processo.
Voi potete mettere nel processo un gestore di segmentation Fault che comandi le cose e vi faccia continuare l'esecuzione.
Perché può essere catturato il segmentation Fault. Il fatto che poi il segmentation Fault non venga catturato, si comporta come qualsiasi altro segnale non gestito che prevede la terminazione. Prego!

Studente: Non mi è chiaro come viene generata la trap?

Perché ... quando ... in caso di accesso in memoria la CPU esegue soltanto un'istruzione, ci vorrebbe un lato software che capisce che quell'indirizzo è sbagliato.

Sì, sì è la MMU che comunica ... È ancora un po' più complesso di così, ci abbiamo un capitolo di gestione della memoria fatto apposta per questo... provo a dire...

Quando si scrive sul bus un indirizzo e alla MMU è stato indicato quale Tabella delle pagine... quale risoluzione usare in questo momento. La MMU guarda in una cache, che si chiama Translation Lookaside Buffer (TLB), non so se l'avete visto ad architettura, c'è una piccola cache che tiene le ultime risoluzioni fatte, in realtà a livello di pagina/frame non a livello di istruzione, comunque se lì non c'è la MMU manda una trap al processore in ogni caso. Possono darsi due casi. Può semplicemente esserci che, siccome la MMU non può tenere tutta la mappa della memoria in memoria

sarebbe troppo pesante per la MMU, semplicemente può succedere che quella risoluzione non sia al momento nello mappa, allora il Sistema Operativo è lui che ricarica la parte mancante e fa riportare e tutto va bene. Se alla prima trov della MMU, perché non trova nella cache che ha a bordo la risoluzione, arrivo al Sistema Operativo e dice "meh! Questo è un indirizzo che non ho neanche nella tabella quella estesa che gestisco io" a quel punto il Kernel genera il segnale. Comunque parte da una segnalazione della MMU. Divisione per 0 o istruzione illegale vengono generati direttamente dal processore.

pausa (min 1:07:45).