**Programmazione di Sistemi**

**Appunti**

Lezione 1:

Sistemi Operativi:

**Come viene suddivisa la memoria**

Memory Menagement

Protection

Logical Memory

Physcal Memory

MMU

Variable

**BackGround**

Programma C into memory

Va sulla memoria secondaria

Acesso diretto su memoria principale o registri -> CPU

File Eseguibile su disco

Spostare eseguibile dal Disk alla Memoria di Sistema

Cosi da accesso diretto

Di disco tanto, sulla memoria abbiamo poco

Disco molto piu lento della memoria

Problem: Bisogna eseguire tanti programmi, che vanno spostati da Disk sulla memoria della CPU

CPU divisa in diversi Task quando lavorano

Ogni processo non deve andare a influire sui altri, Protection of Memory

Ex: val di a e val b vanno sposate su CPU per essere eseguibile

Processo1 -> ha accesso solo a questo indirizzo, non ad altri

Come: Per ogni processo sono disponibile 2 registri

Base and Limit Registers

Per ogni processo diciamo diamo il range

Define ranger of legal address

Base specifica la memoria piu piccola in cui puoi avere accesso

Limit Register -> Fino dove puoi avere accesso (range)

**Hardware Address Protection**

CPU ogni accesso, va a controllare l’indirizzo, vede se è piu grande o piu piccolo di base + limit a cui possiamo accedere

Cosi ogni processo cerca di accedere a un indirizzo di memoria di un processo, base e limit

**Address Binding**

Programma su disk, va portato in memoria cosi CPU la esegue

Quando creaiamo un app ci da l’indirizzo che possiamo chiedere indirizzo symbolic

Va da Disko su Memoria -> Dove c’è spazio

Quando chiamiamo il programma , il Compiler va su relocatable address, con un operazione di Address blinding che lo manda in un registro

**Multistep Processing of a User Program**

Compile Time , convert into machine code.

Load Time, dopo la creazione va in memoria l’eseguibile

Run Time, Il programma viene eseguito -> Durante la run time mettiamo in memoria ed eseguiamo

Il blinding of instrucitions in queste tre fasi ( scelta nostra ) va in un indirizzo di memoria fisico vero e proprio

Blinding in Compile Time : Absolute Code

B- in Load Time: Relocatable code

B- in Execution time: need hardware support for adress maps (base and limit registers)

Movimenti per gestire da Disk a Memory

**Logical vs Physical Address Space**

Nel address generated la CPU crea Logical Address

Dall’altra parte cè la Main Memory , bisogna spostare il Task nella CPU sulla Memory che il suo registro è nella Physical Address

Binding – l’indirizzo logico va mappato e poi vanno spostati sulla memoria fisica (nella RAM) e per eseguibile vanno spostate nella RAM per essere eseguibili.

Ho bisogno di qualcosa in mezzo per mappare l’indirizzo logico dichiarato dalla CPU all’indirizzo fisico sulla Memoria disponibile

MMU -> Fa questo lavoro, dato che abbiamo bisogno di velocità

MMU = Run-Time mapping from the virtual to physical addresses

**MMU**

CPU -> logical address -> MMU -> physical address -> Physical Memory

**Memory-Management Unit**

Processo 1 puo avere accesso a Base, inidirizzo minimo o massimo che puo accedere

Base register diventa Relocation Register

Viene sommato

CPU -> logical address (346) -> relocation register (1400) -> MMU

physical address (1436) -> Memory

**Dynamic Loading**

Processo bisogna spostarlo su RAM

Abbiamo bisogno delle informazioni ( function, librerie.. ecc..)

Bisogna mettere solo quello che serve

Portiamo Loading di informazioni utili : Dynamic Loading

**Dynamic Linking**

Static Linking -> Modulo completo con tutto cio che serve

Dynamic Linking Libreries (DLL) -> Librerie Condivise A e B fanno il Loading di una libreria dato che ce lhanno in comune la condividono

Ex: collection of executable code and data...

Ex: Shared by multiple programs simultaneously – Shared Libraries

**Contigous Allocation**

Main Memory viene diviso in 2, la prima viene usata per assegnare e memorizzare il codice di Sistema Operativo, la seconda parte dove cè tutto il resto che è disponbiile per mettere diversi Task

Come metterli?

Abbiamo la memoria disponibile, appena arriva il processo 5 (ex) va nel primo spazio disponibile

Processo 8 (ex) va in quello dopo ec.. Sempre a catena in successione a quella disponibile

Tempo di esecuzione finito, va spostato nel disco e va in un altro indirizzo e cosi via

Se non cè spazio, va ad aspettare che si libera il processo per allocarne uno nuovo

2° Modo:

Prima che arriva il processo lo dividiamo in diverse sezioni, lo dividiamo

Quando arriva processo 5 (ex) lo allochiamo noi in una partizione specifica, in base alla divisione in byte che noi gli abbiamo dato

Si fa di solito nei processi grandi magari che richiedono tanti byte disponibili

Cerchiamo una memoria sufficiente in pratica per allocarli la memoria per aumentare l’efficienza

Abbiamo un processo, prendiamo tutte le informazioni e tutte insieme come un singolo modulo le allochiamo nella memoria fisica

Processo 1 ha bisogno di 4 byte -> e troviamo una parte della memoria adatta e li allochiamo uno dopo l’altro

CPU -> logical address -> limit register (verifica operazione legale)

E ->Relocation register -> physical address -> memory

Poi succede:

**Variable Partition**

Dopo p1, il buco che creano gli indirizzamenti specifici creano un HOLE ovvero un buco di memoria disponibile della memoria che eventualmente possiamo utilizzare per allocare un processo.

**Dynamic Storage-Allocation Problem**

Dobbiamo decidere dove allocare il processo, in quale parte della memoria che è disponbile ovvero 3 temi:

First – Fit -> mettiamo nel primo spazio hole abbastanza grande dove allocare il processo ( metodo più veloce )

Best – Fit -> Su tutta la memoria disponibile, dobbiamo andare a cercare il modulo piu piccolo ma comunque sufficente per il nostro processo, in maniera da rendere il tutto piu efficente ( cerchiamo in tutta la memoria )

Worst – Fit -> Allochiamo cercando in tutta la memoria, (il meno veloce) bisogna trovare il modulo abbastanza grande per il processo (sempre l’hole) , ma rispetto agli altri cerchiamo il più grande possibile . Di solito cerchiamo uno abbastanza grande per allocare altri processi

**Exercise:**

Given 6 memory partitions of 300kb, 600kb, 350kb, 200kb, 750kb and 125kb.

How would the first fit-, best-fit, and worst-fit algothims place processes of size 115kb, 500kb,358kb,200kb and 375kb (in order) ?

-:

P1 115kb, P2 500kb.... ecc..

P1 115kb va in 300kb di partion e rimane un modulo di 185kb disponibile in memoria

P2 500kb va in 600kb quindi nel 2° partion M2 e quindi 100kb disponibili ancora in memoria

P3 358kb va in (1 non abbastanza,m2 non abbastanza... ) arriviamo a M5 e lo mettiamo dentro, quindi 392 kb rimangono in M5

P4 200kb va... in M3 , e rimangono ancora 150 kb in M3

P5 375 va... M5 che avanzavano 392kb e lo inseriamo li dato che è il primo slot di memoria disponbile, e avanza comunque ancora qualcosa.

Tutti i processi sono stati inseriti senza aspettare una locazione di memoria disponibile, quindi sono stati lisci dato che sono andati tutti bene. “ FIRST-FIT “

Nel caso un processo debba aspettare, deve aspettare il primo slot di memoria disponbile per l’allocazione, sempre in base al metodo che stiamo utilizzando per applicarli. ( i 3 cosi lassu)

Come allocare i processi nella memoria disponbile???

Usiamo altre teniche, tipo dividendo in moduli piu piccoli.