**APPUNTI SISTEMI OPERATIVI**

**Sistemi Multiprocessori:**

* Simmetrici:
  + I core hanno memoria comune
* Numa (non uniform memory access):
  + Ogni core ha memoria propria

**Evoluzione dei SO:**

* Primi calcolatori:
  + Privi di SO
  + Programmatore operatore interattivo
* Prima generazione (50-60):
  + virtualizzazione dell'I/O, librerie di controllo dei device
  + separazione del programmatore dalla macchina tramite l' operatore
  + monitor residente: lavoro da console, programma permanente
* Seconda generazione (60 -65)
  + indipendenza tra programmi e dispositivi usati ( logical I/O)
  + multiprogrammazione e time sharing
* Terza generazione (65 -75):
  + SO unico
  + Risorse virtuali
  + Sistemi multifunzione
* Quarta generazione(75-85):
  + Macchine virtuali
  + Sistemi multiprocessori e distribuiti
  + User interface
* Quinta generazione(85-95):
  + Personal computer
  + Reti locali, internet, web
* Sesta generazione(95-05):
  + Elaborazione distribuita, peer to peer, servizi online, e-commerce

**Tecniche di gestione di un sistema di calcolo:**

* Monoprogrammazione:
  + Processi sequenziali
* Multiprogrammazione:
  + + processi nello stesso momento
  + Gestione batch (esecuzione accorpata di + processi)
  + Time sharing

**Spooling:**

* Prima i programmi vengono caricati in memoria poi vengono passati alla cpu (+ veloce)

**Modello interazione cpu dispositivi I/O:**

* Ciclo fetch-execute
  + Istruzione – calcolo – output
* Cpu:
  + Registri di IR(instrucrion r.) e PC (program counter)
* Tramite controllore dispositivo:
  + Registri di controllo, di stato e dei dati
  + Buffer con flag BUSY e READY
* Dispositivi:
  + Possono essere in 3 stati: libero-pronto-occupato

**Throughput:**

* capacità di trasmissione effettiva che viene usata (programmi/tempo)

**Interrupt**

* E’ una richiesta al sistema operativo da parte di un processo in esecuzione (per indicare che un’azione è stata eseguita)

**Interrupt da timer:**

* E’ un interrupt periodico generato da un timer hardware. Utilizzato come alternativa agli interrupt dei singoli dispositivi
  + Meno efficiente perché può causare overhead dovuti al salvataggio e al ripristino del programma interrotto

**Multiprogrammazione**

Il sistema operativo gestisce le interruzioni tramite la interrupt routine (che porta a termine l’azione richiesta dall’interrupt)

**Interruzioni:**

* Interruzioni hardware - Sincrone (interne al programma)
  + Device interrupt
  + Timer interrupt
  + Powerfail sense interrupt
* Interruzioni software - Asincrone (esterne al programma)
  + Eccezioni o trap: per violazioni illegali
  + System call: chiamate di sistema per accedere a funzioni del SO o per trasferire il controllo al SO
* Dopo l’interrupt routine (e quindi il cambiamento di programma) il SO utilizza algoritmi di scheduling per determinare il processo successivo
* Lo stato del processore al momento di un interrupt viene salvato in una stack.
* Gli interrupt possono avere diversi livelli di priorità.

**Protezione:**

* Per evitare che un processo modifichi dati di un altro processo
* Politiche (cosa) e meccanismi (come) per controllare accessi alle risorse
* Hardware identifica errori e vengono segnalati tramite le *trap*
* In presenza di errore o violazione si effettua un *dump* della memoria
* Diritti di accesso e domini di protezione per i processi nei confronti di risorse
  + Lista delle capabilities: lista di oggetti e operazioni consentite su tali oggetti riferiti ad un dominio
  + Capability (chiave): token di autorità immodificabile dai processi
* Protezione per funzionalità della CPU:
  + Kernel mode
  + Supervisor mode
  + User mode

Per passare da modo a modo si necessita di una interruzione

* **Protezione della memoria**:
  + Tramite registri limite:

L’area di memoria dedicata ad un processo viene confinata così da fare in modo che il processo sia isolato.

Base; Base + Limite. Se indirizzo generato non rientra negli estremi allora errore di indirizzamento (trap)

**Memoria virtuale:**

* è un'architettura di sistema capace di simulare uno spazio di memoria centrale maggiore di quello fisicamente presente o disponibile. Viene eseguita mappando gli indirizzi (in memoria logica non si hanno dati ma indirizzi a dati in memoria fisica)
* Perché?
  + Solo una parte del programma è necessaria in memoria per l'esecuzione
  + Lo spazio di indirizzamento logico può quindi essere molto più grande di quello fisico
  + Consente agli spazi di indirizzamento di essere condivisi da più processi
  + Consente un più efficiente creazione dei processi
  + Più programmi eseguono concorrentemente
  + Meno I/O è necessario per caricare o avvicendare (swap) processi
* Come avviene la traduzione da indirizzo logico a indirizzo fisico?
  + Tramite la **MMU** (Memory Management Unit)
* **Implementazioni**:
  + **Demand Segmentation** (segmentazione su richiesta ormai obsoleta):
    - Indirizzo con 2 elementi**: Segment** - **Offset**
    - Memoria virtuale divisa per **compartimenti** (es data, codice etc)
    - Tradotta tramite *segment table*: viene tradotto il segmento virtuale in quello fisico, poi sommato con l’offset da l’indirizzo fisico

Es: segment = n° pagina, offset = n° riga

* + **Demand Paging** (paginazione su richiesta):
    - Memoria logica divisa in blocchi detti **pagine**
    - Memoria fisica divisa in blocchi detti **frame**
    - Indirizzo con 2 elementi**: page number** – **page offset**
    - Tradotto tramite *page table*: il n° di page viene tradotto nel frame corrispondente, e sommato all’offset da l’indirizzo fisico

Es: page = n° pagina, offset = n° riga

* + - Tradotto anche con **TLB** (translation look aside buffers): una special cache HW di ricerca
    - Protezione implementata tramite un bit di protezione *Valid-invalid*

**System Call:**

* Costituiscono l'interfaccia tra un programma in esecuzione ed il S.O.
* Categorie principali:
  + controllo dei processi e dei job
  + manipolazione dei file e dei dispositivi
  + gestione delle informazioni
  + comunicazione
* In windows: Windows API

**Struttura dei SO**

* **Struttura a livelli monolitica:**
  + Users
  + Librerie di sistema, shell, compilatori…
  + Kernel (scheduling, drivers, file di sitema etc)
  + Hardware
* **Struttura a microkerkel**:
  + Kernel più piccolo che comprende solo le funzioni più basiche (scheduling, comunicazione tra processi…) che comunica con l’hardware, i programmi applicativi, i file di sistema e i driver
* **Sistema a macchine virtuali**:
  + Usando lo scheduling della CPU e la tecnica della memoria virtuale, si possono creare macchine virtuali, una per ogni processo. Si consegue il massimo livello di protezione, a scapito dell'efficienza.

**Macchine virtuali:**

* Usando lo scheduling della CPU e la tecnica della memoria virtuale, si possono creare macchine virtuali, una per ogni processo. Si consegue il massimo livello di protezione, a scapito dell'efficienza.
* Creano l’illusione di processi multipli, ciascuno in esecuzione sul suo processore privato e con la propria memoria virtuale privata, messa a disposizione dal proprio kernel del SO, che può essere diverso per processi diversi
* presentare all’utilizzatore una visione delle risorse del sistema diversa da quella reale.
* Tipi:
  + A livello di processo: per isolare i processi
  + **VMM**: virtual machine monitor
  + **VMM di sistema:**
    - Una singola piattaforma hardware viene condivisa da più sistemi operativi, ognuno dei quali è installato su una diversa macchina virtuale
  + **VMM Ospitato:**
    - Il VMM viene installato come un’applicazione sopra un SO esistente, che opera nello spazio utente e accede all’hardware tramite le system call del S.O. su cui viene installato. Es: VM oracle
* **Emulazione**:
  + eseguire applicazioni (o SO) compilate per un'architettura su di un’altra
  + uno strato software emula le funzionalità dell'architettura; il s.o. esegue sopra tale strato
* **Virtualizzazione**:
  + definizione di contesti di esecuzione multipli (macchine virtuali) su di un singolo processore, partizionando le risorse.
* **Vantaggi virtualizzazione:**
  + Più SO sulla stessa macchina fisica
  + Isolamento tra gli ambienti
  + Sicurezza
* **Virtualizzazione a livello di SO:**
  + Una modalità di virtualizzazione in cui il kernel consente l'esistenza di molteplici istanze isolate nello spazio utente
  + Tali istanze (**container**) rendono disponibili ai programmi contenuti solo i contenuti e dispositivi che sono state assegnati all'istanza.

**Processi**

* **Svolgimento di operazioni:**
  + **Algoritmo**: Procedimento logico che deve essere seguito per risolvere il problema in esame
  + **Programma**: Descrizione dell'algoritmo tramite un opportuno formalismo ( linguaggio di programmazione )
  + **Processo**: (sequenziale): La sequenza di eventi cui dà luogo un elaboratore quando opera sotto il controllo di un particolare programma. E’ parte di un programma
* **Processi e cpu:**
  + Alcuni processi dispongono di un processore privato e pertanto
  + Altri processi condividono un processore comune, la CPU
  + Ai processi che non dispongono di un processore privato associamo un processore virtuale
* **Stati dei processi:**
  + In **esecuzione**: se attivo sul processore
  + **Pronto**: se dispone di tutte le risorse e le condizioni logiche per eseguire ma non dispone del processore
  + In **attesa**: se non dispone delle risorse e delle condizioni logiche per essere eseguito
* **Descrittore di processo o Process Control Block (PCB):**
  + area di memoria, associata al processo e contenente tutte le informazioni proprie del processo
* Tipi di interazioni tra processi:
  + Competizione
  + Cooperazione
  + Interferenza
    - Di primo tipo: quando un processo modifica lo stato di un altro processo indesideratamente
    - Di secondo tipo: quando due processi entrano in competizione per una risorsa (mutua esclusione)
* Composto da **Thread** (percorso, filone):
  + 1 solo thread: *Processo sequenziale*
  + 2 o più thread: *Processo multi-thread*
* Sezione critica: La sequenza di istruzioni con le quali un processo accede e modifica un insieme di variabili comuni

**Modelli di interazione tra processi:**

* **Modello ad ambiente globale** (a memoria comune):
  + costituito da uno o più processori che hanno accesso ad una memoria comune
  + ogni interazione avviene tramite oggetti contenuti nella memoria comune
* **Modello ad ambiente locale** (a scambio di messaggi):
  + Il sistema è visto come un insieme di processi ciascuno operante in un ambiente locale non accessibile direttamente a nessun altro processo.
  + Ogni forma di interazione tra processi (comunicazione, sincronizzazione), avviene tramite scambio di messaggi
  + Non esiste il concetto di risorsa accessibile direttamente ai processi:
    - alla risorsa è associato un processo servitore
    - la risorsa viene passata da un processo ad un altro sotto forma di messaggi
  + Il modello a scambio di messaggi rappresenta la naturale astrazione di un sistema privo di memoria comune (sistema distribuito), in cui a ciascun processore è associata una memoria privata.

Es: sistemi server

* + Può essere realizzato anche in presenza di memoria comune, che viene utilizzata per realizzare canali di comunicazione

**Modello di interazione in ambiente Globale:**

* Problema della mutua esclusione:
  + Tempificazione dell'esecuzione dei singoli processi
  + Inibizione delle interruzioni del processore sul quale sono eseguite le sezioni critiche durante l'esecuzione di ciascuna di esse
  + Strumenti di sincronizzazione (semafori, produttore-consumatore)
* Sincronizzazione dei processi mediante **Semafori:**
  + Sono uno strumento di sincronizzazione generale e flessibile al problema della mutua esclusione e ad altri problemi di sincronizzazione
  + Un semaforo è una variabile intera non negativa “s”
  + Al semaforo è associata una lista di attesa Qs nella quale sono posti i descrittori dei processi che attendono l'autorizzazione a proseguire nell’esecuzione
  + Utilizzati tramite le loro primitive:
    - wait (s) & signal (s)
  + wait e signal Devono essere indivisibili:
    - Si può ottenere indivisibilità inibendo le interruzioni durante l'esecuzione di wait e signal (solo su uno stesso processore)
    - Nel caso di sistema multiprocessore occorre considerare wait e signal come sezioni critiche brevi e proteggerle mediante un meccanismo di più basso livello denominato lock

|  |  |
| --- | --- |
| wait(s):  begin  if s = 0 then  <il processo viene sospeso e il suo descrittore inserito in Qs>;  else s:= s - 1;  end; | signal(s)  begin  if <esiste un processo in coda> then  <il suo descrittore viene rimosso da Qs e il suo stato modificato in pronto>;  else s := s + 1;  end; |

* **Lock e Unlock:**
  + Meccanismo di più basso livello
  + x indicatore associato alla classe di sezioni critiche (inizializzato a 0):
    - x = 0 nessuna sezione critica in esecuzione
    - x = 1 una sezione critica in esecuzione
  + lock e unlock devono essere indivisibili:
    - Occorrono speciali istruzioni dei processori come TSL (Test and Set Lock) e CMPXCHG (Intel)

|  |  |
| --- | --- |
| lock (x):  begin  repeat until x = 0; // test del valore di x  x := 1; // modifica del valore di x  end; | unlock (x):  begin  x := 0; //indivisibile  end; |

* **Produttore consumatore:** Buffer in grado di contenere N messaggi, a cui accedono il processo P(produttore) per l’inserimento di un messaggio ed il processo C(consumatore) per il prelievo di un messaggio

**Modello di interazione in ambiente Locale:**

* Tramite scambio di messaggi:

|  |
| --- |
| type messaggio = record  origine: …;  destinazione: …;  contenuto: …;  end; |

* + Messaggio:
  + ad ogni processo sia associata una **coda** per i messaggi in arrivo;
  + le primitive di comunicazione usate dai processi sono:
    - **send** (m): inserisce il messaggio m nella coda del destinatario
    - **receive** (m): preleva un messaggio dalla coda o sospende il processo se la coda è vuota.
  + Con lo scambio di messaggi si ha:
    - Comunicazione
    - Sincronizzazione
  + La mutua esclusione non è più un problema, perché tutte le risorse sono private.
  + **Buffer di comunicazione:**
    - È necessario un processo gestore **(buffer\_control**) della risorsa buffer che serve i processo produttori P e i processi consumatori C
  + Classificazione:
    - Per **Designazione** dei processi sorgente e destinatario di ogni comunicazione
      * Designazione **diretta** (o esplicita)
        + Simmetrica
        + Asimmetrica
      * Designazione **indiretta** (o implicita)
        + **Mailbox**
        + **Porte**
      * **Naming statico**: solo tramite canali preassegnati
      * **Naming dinamico**: canali preesistenti possono essere assegnati dinamicamente in base all’occorrenza
    - Per tipo di **sincronizzazione** tra i processi comunicanti:
      * **Sincrona**
      * **Asincrona**
      * **RPC**

**Deadlock**:

* Indica una situazione in cui due o più processi o azioni si bloccano a vicenda, aspettando che uno esegua una certa azione che serve all'altro e viceversa
* **Grafico di allocazione risorse:**
  + Schema analitico per la gestione delle risorse
  + Frecce:
    - Richiesta: da processo a risorsa
    - Assegnamento: da risorsa a processo
  + Se il grafo se il grafo contiene un ciclo ci può essere deadlock
  + Se è presente una sola istanza per ogni tipo di risorsa e c’è un ciclo allora c’è deadlock
* **Condizioni di deadlock:**
  + **Mutua esclusione**
  + **Possesso e attesa** (hold and wait)
  + **Assenza di revoca**
  + **Attesa circolare**
* **Strategie per evitare il deadlock:**
  + **Ignorare il problema**
  + **Rilevare la presenza di deadlock e risolverlo**
  + **Prevenzione dinamica**
    - Algoritmo del banchiere
  + **Prevenzione Strutturale:**
    - Evitare una delle 4 condizioni (di solito l’attesa circolare)

**Scheduling della CPU:**

* La gestione delle risorse impone al SO di prendere decisioni sulla loro assegnazione in base a criteri di efficienza e funzionalità
* **CPU burst**:
  + la quantità di tempo in cui il processo utilizza il processore prima che non sia più pronto
  + Un programma I/O bound ha molti burst di CPU, brevi
  + Un programma CPU bound ha pochi burst di CPU, lunghi
* **CPU scheduler:**
  + Parte del S.O. che decide a quale dei processi (o thread ) pronti presenti nel sistema assegnare il controllo della CPU
  + Due tipi:
    - **Long term scheduler** (o job scheduler):
      * Determina quali processi dalla memoria di massa devono essere caricati in memoria principale pronti per l'esecuzione.
      * Controlla il grado di multiprogrammazione
      * Il criterio di selezione è basato su un mix equilibrato di job I/O bound e CPU bound
    - **Short term scheduler**:
      * Seleziona tra tutti i processi in memoria pronti per l'esecuzione quello cui assegnare la CPU
    - Medium therm:
      * Nei sistemi time sharing / interattivi non esiste long term scheduling.
      * Può risultare vantaggioso, talvolta, rimuovere alcuni processi dalla memoria e ridurre il grado di multiprogrammazione. I processi vengono successivamente reintrodotti swapping.
  + **Dispatcher**:
    - Passa effettivamente il controllo della cpu ai processi scelti dallo short therm scheduler
* Tipi di scheduling:
  + Non **preemptive**: Il processo una volta in esecuzione non può essere interrotto (al di fuori di azioni logiche come interrupt o azioni di i/o)
  + **Preemptive**: Il processo in esecuzione può essere interrotto e ripreso successivamente
* **Algoritmi di Scheduling**:
  + **First come first served (FCFS)**:
    - La CPU viene assegnata al processo che l'ha richiesta per primo.
    - Solo non preemptive
  + **Shortest job first (SJF)**:
    - A ciascun processo è associata la lunghezza del successivo burst di CPU. Quando la CPU è libera, essa viene assegnata al processo con il burst di CPU più breve.
  + **Priorità**:
    - Processi inseriti in code separate per diversi livelli di priorità
    - I processi nella coda con priorità più alta hanno priorità rispetto ai processi delle altre code
    - All’interno della coda la priorità segue la legge del FIFO
    - Priorità può essere statica o dinamica:
      * Statica: preattribuita e immutabile
      * Dinamica: può aumentare secondo criteri come il tempo di inattività del processo o la propria velocità
  + **Round Robin:**
    - Assegna un quanto di tempo prefissato ad ogni processo
    - La coda dei processi pronti è circolare e la CPU è assegnata a ciascuno dei processi per un quanto di tempo.
    - Un processo, se interrotto per l'esaurimento del suo quanto, viene inserito come ultimo nella coda dei processi pronti (preemption)
  + Possono esservi code di processi a più livelli che sfruttano diversi algoritmi, a esempio per i job foreground l'algoritmo R.R., per quelli background l'algoritmo FCFS.
  + Valutazione degli algoritmi:
    - Valutazione analitica (es: tramite tempo medio di attesa)
    - Modelli basati sulla teoria delle code.
    - Modelli basati su reti di Petri temporizzate.