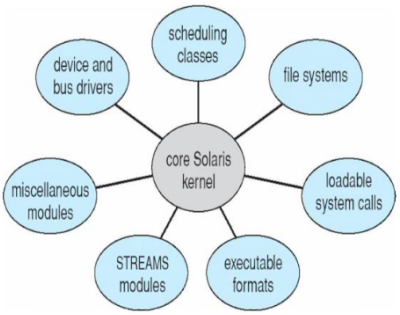
12 Maggio 2023

ARCHITETTURE DI CALCOLO LEZIONE 16

Introduzione e gestione dei sistemi operativi

Approccio modulare di Solaris



Tra la varietà di SO proposti, rientra anche Solaris, un sistema operativo non creato per il grande pubblico, creato da Sun Microsystems. Ispirandosi alsistema solare, prevedeva un kernel centrale(figurativamente il Sole) attorno al quale si disponevano tutte le altre componenti non centrali del sistema operativo. Viene detta architettura a stella, ma di fatto è un'architettura a due strati. Una caratteristica unica di questa architettura era l'estensibilità, cioè era possibile realizzare servizi aggiuntivi a seconda della macchina specifica.
L'organizzazione modulare lascia la possibilità alkernel di fornire i servizi essenziali, ma permette anchedi implementare dinamicamente servizi aggiuntivi,specifici per il particolare sistema di calcolo.

Sistemi Ibridi

La maggior parte dei sistemi operativi non adottano un modello "puro". I modelli ibridi combinano diversi approcci implementativi allo scopo di migliorare le performance, la sicurezza e l'usabilità. Ad esempio, i kernel di Linux e Solaris sono fondamentalmente monolitici, perché mantenere il SO in unico spazio di indirizzamento garantisce prestazioni migliori, sono però anche modulari, per cui le nuove funzionalità possono essere aggiunte dinamicamente al kernel.
Windows è perlopiù monolitico, ma conserva alcuni comportamenti tipici dei sistemi microkernel, tra cui il supporto per sottosistemi separati (detti personalità) che vengono eseguiti come processi in modalità utente e permettono di utilizzare Windows per sistemi embedded.

Android

È un sistema operativo per l'ambiente desktop, che si fonda sul kernel LINUX modificato. Sviluppato dalla Open Handset Alliance − guidata da Google e costituita da oltre 30 compagnie, tra cui ASUS, HTC, Intel, Motorola, Qualcomm, T−Mobile, Samsung e NVIDIA − gestisce una grande varietà di piattaforme mobile ed è open−source. La scelta del kernel

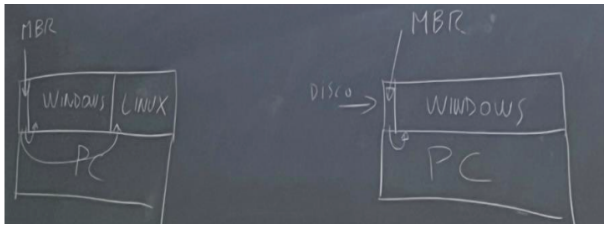
Linux è legata alla sua leggerezza, prestanza ed affidabilità. Le differenze tra Linux standard e Linux di Android:

- Nell'ambiente mobile il display è più piccolo, con minimizzazione della capacità di aprire contemporaneamente più finestre. Picture to Picture: la possibilità di avere un'immagine nell'altra immagine che è la modalità più semplificata per aprire più finestre contemporaneamente nel mobile)
- Meno spazio per memorizzazione, processori più lenti
- Il range di periferiche nel mobile è minore, con una conseguente maggiore semplicità nella loro gestione
- Consumo energetico ridotto grazie alla modalità di risparmio energetico (cambio degli algoritmi dello scheduler del SO).

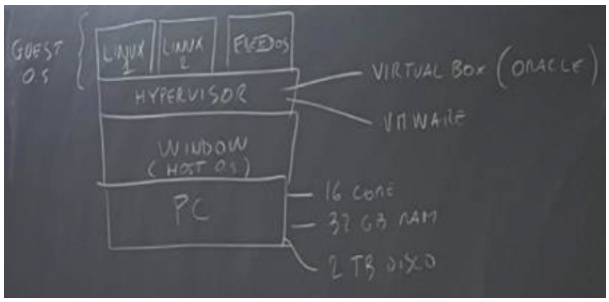
L'ambiente Android è costituito da un insieme di librerie che hanno le funzioni tipiche del SO, e da una macchina virtuale ART (Android Run Time), che è l'ambiente di esecuzione delle app. Le app sono sviluppate in Java (linguaggio interpretato e non compilato); la scelta di un linguaggio interpretato, più lento e meno efficienze, è legato alla maggiore semplicità e portabilità. Disponibile anche l'interfaccia Java nativa (JNI) per bypassare ART, e scrivere programmi Java con accesso più diretto all'hardware.

Le macchine virtuali

La macchina virtuale si frappone fra il SO della macchina ed il nuovo sistema operativo che si vuole installare.
Vi sono due metodi per utilizzare SO diversi sulla stessa macchina:



•Installarli tutti nell'hard disk, per poi richiamarli tramite il programma di bootstrap (che si trova nel MBR→master boot record) ad ogni avvio del computer, dando all'utente la possibilità di scegliere quale di essi usare.
•Installare una macchina virtuale che, tramite un hypervisor o gestore delle macchine virtuali (es. MVWARE, VIRTUAL BOX), permette di supportare più SO (Guest SO).



Una macchina virtuale è un ambiente che riproduce una macchina fisica con caratteristiche che sono un sottoinsieme delle caratteristiche della macchina fisica ospitate (Host OS) (es. macchina fisica: 16 core, 32 GB di RAM, disco da 2 TB; macchina virtuale: 4 core, 8 GB di RAM, disco da 100 GB).
Può essere utile fare uso di una macchina virtuale perché:

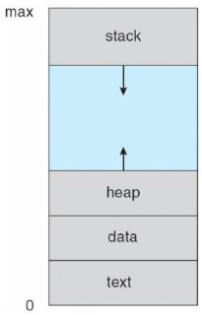
- si vuole installare un SO diverso da quello supportato dalla macchina (es. windows su dispositivi Apple)
- si vuole sviluppare un software in un ambiente sandbox, ovvero un ambiente virtuale “isolato” dal resto del computer
- è possibile spostare una macchina virtuale da una macchina fisica ad un’altra macchina fisica con uno stesso hypervisor; ciò è alla base del funzionamento dei cloud e del consolidamento, ovvero il raggruppamento del maggior numero di macchine virtuali nel minor numero di macchine fisiche, che può avvenire a patto che le macchine virtuali
    - possano essere ibernate
    - possano essere trasferite sulla rete (i tempi dipendono dalla larghezza della banda della rete)

In sintesi
una macchina virtuale fornisce una identica interfaccia a diverse architetture hardware e più macchine virtuali possono essere installate su uno stesso computer.
Il sistema operativo crea l'impressione di diversi processi ognuno in esecuzione su un proprio processore usando la propria memoria.

Gestione dei processi

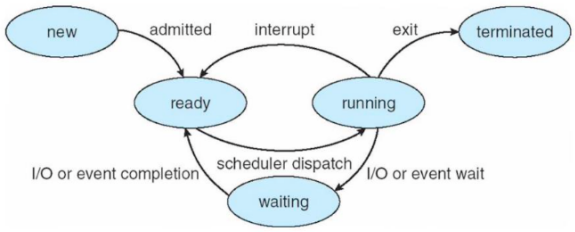
La parola che descrive un programma in esecuzione più variare a seconda del sistema operativo; per cui in un contesto di:
- Sistemi batch si parlerà di job
- Sistemi time-sharing si parlerà di processi utente o task
- Sistemi che supportano il multithreading si parlerà di thread

Un processo è definito come un programma in esecuzione o, in alternativa, come la sequenza di eventi che avvengono in un elaboratore quando opera sotto il controllo di un particolare programma. Un processo include:
- Una sezione di testo (il codice del programma da eseguire)
- Una sezione dati (le variabili, ovvero i dati su cui vengono fatte le elaborazioni)
- Lo stack (dati temporanei generati quando si avvia una routine: parametri per i sottoprogrammi, variabili locali e indirizzi di rientro, un record di attivazione)
- Heap (letteralmente "mucchio", grande quantità memoria dinamicamente allocata durante l'esecuzione del task, es. array, liste, alberi, tabelle hash)
- Un program counter (registro contenente la prossima istruzione da eseguire) ed il contenuto dei registri della CPU.



Il programma è un'entità "passiva" memorizzata su disco (un file eseguibile), che, una volta in esecuzione (quindi, caricato nella memoria principale), si trasforma in un processo (entità “attiva”). L'esecuzione di un programma viene attivata dal doppio click del mouse in una GUI (Graphical User Interface), o immettendo il nome del file eseguibile da linea di comando. Al processo viene assegnato un valore di minimo e di massimo di spazio occupabile in memoria.

Questo sarà costituito dal text (il codice), dai dati e dal heap e dallo stack, che crescono l'uno contro l'altro (non viene assegnata loro una quota di memoria predefinita). Se lo stack e l'heap si "scontrano", si verifica lo stack-heap overflow, ovvero è stata consumata tutta la memoria destinata a queste due componenti.
Mentre viene eseguito, un processo è soggetto a transizioni di stato (rappresentate da un automa a stati finiti), definite in parte dall'attività corrente del processo ed in parte da eventi esterni, asincroni rispetto alla sua esecuzione. I possibili stati sono:



•New (nuovo): il processo viene creato
•Running (in esecuzione): se ne eseguono le istruzioni
•Waiting (in attesa): il processo è in attesa di un evento
•Ready (pronto): il processo è in attesa di essere assegnato ad un processore
•Terminated (terminato): il processo ha terminato la propria esecuzione

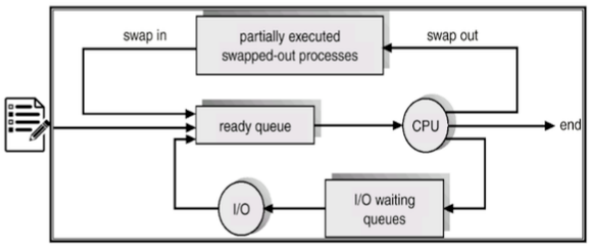
Esaminiamo nello specifico ciascuna transizione tra i diversi stati:

- Lo stato new corrisponde alla creazione di un nuovo processo (al caricamento del codice da disco in RAM).
  La transizione dallo stato new a ready avviene quando il SO (scheduler a lungo/medio termine) ammette il nuovo processo nella coda d'attesa attiva per la CPU (inclusione nella ready queue). La ready queue non funziona secondo il principio del "first in, first out", ma si basa sulle priorità;
- La transizione dallo stato ready a running avviene ad opera di un componente dello scheduler (dispatcher) quando, in seguito al blocco del processo in esecuzione, il processo viene scelto, fra tutti i processi pronti, per essere eseguito (dispatch).
- La transizione da running a ready, chiamata revoca o prerilascio, avviene:
  - nello scheduling a priorità, quando arriva al sistema un processo con priorità maggiore
  - nei sistemi a partizione di tempo, per esaurimento del quanto
  - al verificarsi di un interrupt esterno (asincrono)
- La transizione da running a waiting avviene per la richiesta di un servizio di I/O al SO (o per l'attesa di un qualche evento), mentre il passaggio da waiting a ready avviene al termine dell'evento. Non esiste il passaggio da waiting a ready in quanto, al termine dell'evento che ha interrotto l'esecuzione del programma, andrà in running il programma con maggiore priorità della ready queue.
- La transizione dallo stato running a terminated può avvenire per:
  - terminazione normale, con chiamata al SO per indicare il completamento delle attività
  - terminazione anomala:
    - Uso scorretto delle risorse (superamento dei limiti di memoria, superamento del tempo massimo di utilizzo della CPU, etc.)
    - Esecuzione di istruzioni non consentite o non valide (trap)

  In entrambi i casi, le risorse riusabili, previamente allocate al processo terminato, vengono riprese in carico dal SO per usi successivi.

Gli scheduler, responsabili di gestire le transizioni tra stati, possono essere di tre tipologie:

- **A breve termine** (o CPU scheduler): seleziona, tra i processi pronti, quelli che devono essere eseguiti→invocato molto frequentemente (millisecondi)



•**A medio termine** (o swapper): schematizzato nell'immagine asinistra, gestisce i processi pronti inmemoria centrale in alcuni sistemitime-sharing, ovvero rimuove i processi dalla memoria (swap out) per riportarli in memoria (swap in) quando sarà possibile. Questo migliora l'utilizzo della memoria in caso di un'alta richiesta di esecuzione di processi.

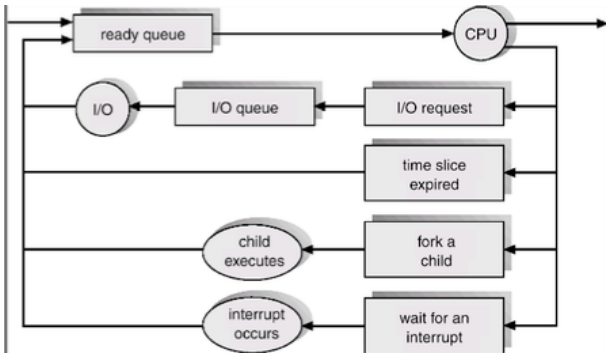
- **A lungo termine** (o job scheduler): presente solo nei sistemi di tipo batch, seleziona i processi da inserire nella ready queue, mantenendoli vicini ad un numero ideale che permette di raggiungere la massima efficienza; per cui controlla il grado di multiprogrammazione→non è invocato spesso (secondi, minuti).

Ad ogni processo è associata una struttura dati del kernel che memorizza tutte le informazioni necessarie a gestire i processi, detta PCB (Process Control Block). È importante che i dati contenuti nel PCB siano accessibili esclusivamente al kernel.
Le informazioni contenute in un PCB sono:

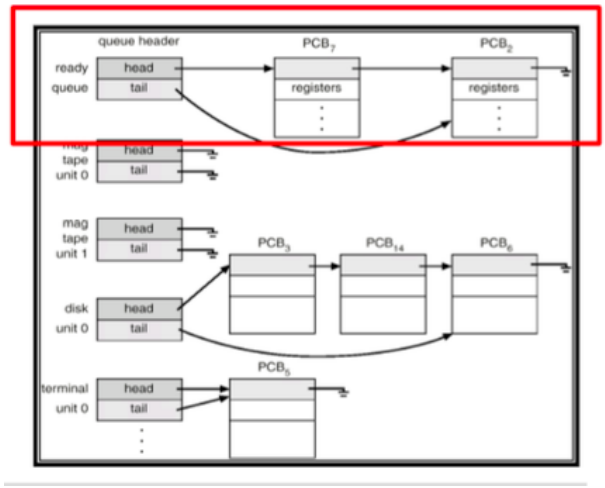
1. Stato del processo
2. Nome (numero) del processo
3. Contesto del processo: program counter, registri della CPU→copiati dalla CPU nel PCB e poi nuovamente dal PCB alla CPU (accumulatori, registri indice, stack pointer, registri di controllo)
4. Informazioni sullo scheduling della CPU (priorità, puntatori alle code di scheduling)
5. Informazioni sulla gestione della memoria allocata al processo (registri base e limite, tabella delle pagine o dei segmenti)
6. Informazioni di contabilizzazione delle risorse: utente proprietario, tempo di utilizzo della
   CPU, tempo trascorso dall'inizio dell'esecuzione, etc.
7. Informazioni sull'I/O: elenco dispositivi assegnati al processo, file aperti, etc.

Scheduling dei processi

Il problema dello scheduling nasce quando si passa dall'uso di sistemi monoprogrammati a sistemi multiprogrammati.
Lo scheduler più usato è quello time-sharing, che si fonda sull'idea delle time-slice ed effettua il round-robin, ossia alterna i programmi in esecuzione nella CPU ad ogni quanto di tempo, con l'obiettivo di rendere più interattiva l'esperienza dell'utente.
Lo scheduling necessita la presenza di code:



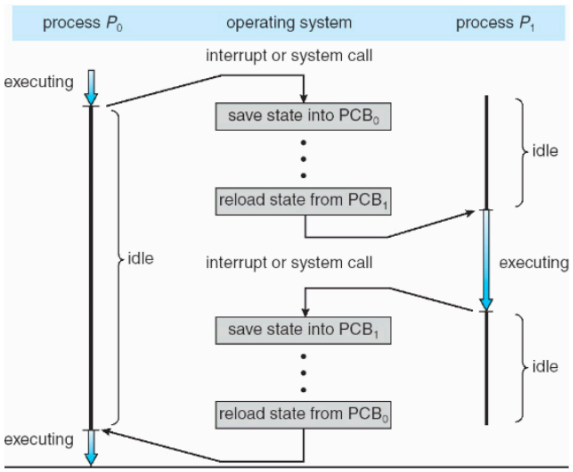
•Coda dei processi: l'insieme di tutti i processi nel sistema (processi new, processi ready, processi waiting, ...).
•Ready queue (coda dei processi pronti): l'insieme dei processi in memoria centrale pronti per essere eseguiti.
•Coda del dispositivo: l'insieme dei processi in attesa di usare un dispositivo (es. stampante). Ne esiste una per ciascun dispositivo.



I processi passano da una coda all'altra durante l'esecuzione in funzione dello stato che stanno attraversando.
I processi presenti in memoria centrale, che sono pronti e nell'attesa d'essere eseguiti, si trovano in una lista detta coda dei processi pronti (ready queue). Questa coda generalmente si memorizza come una lista concatenata: un'intestazione della coda dei processi pronti contiene i puntatori al primo e all'ultimo PCB dell'elenco, e ciascun PCB comprende un campo puntatore che indica il successivo processo contenuto nella coda dei processi pronti.

Il diagramma di accodamento spiega in modo semplice l'alternarsi delle diverse code in cui un processo andrà ad inserirsi (fork a child=creazioe di un processo figlio/sottoprocesso).
I processi possono essere classificati come:
- processi 1/0-bound— basso uso della CPU ed elevato uso dell'I/O.
- processi CPU-bound — elevato uso della CPU e basso uso dell'I/O.

È importante distinguere le due tipologie per poter utilizzare lo scheduler più adatto.



Quando la CPU passa da un processo all'altro, il sistema deve salvare il contesto del vecchio processo nella RAM e caricare il contesto, precedentemente salvato, per il nuovo processo in esecuzione. Il contesto è rappresentato all'interno del PCB del processo e comprende i valori dei registri della CPU, lo stato del processo e le informazioni relative all'occupazione di memoria. Il tempo di context-swicth è un sovraccarico (over- head) per la CPU; il sistema non lavora utilmente mentre cambia contesto (idle). Più sono complicati il SO e, conseguentemente, il PCB, più è lungo il

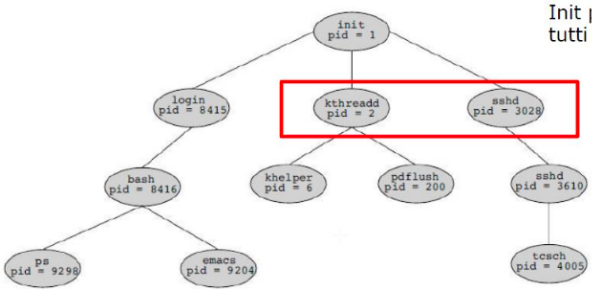
tempo di context-switch.
Lo schema rappresenta il passaggio dal processo P0 al processo P1 e viceversa. Context-switch per dispositivi mobili
Alcuni SO per mobile (es. le prime versioni di iOS) prevedevano un solo processo utente in esecuzione (sospendendo tutti gli altri).
A partire da iOS 4:

- un unico processo in foreground (primo piano), controllabile mediante GUI (Graphical User Interface)
- più processi in background, in memoria centrale, in esecuzione, ma impossibilitati ad utilizzare il display (quindi con "capacità" limitate)
- applicazioni eseguibili in background se:
  - realizzano un unico task di lunghezza finita (completamento di un download dalla rete)
  - ricevono notifiche sul verificarsi di un evento (ricezione di email)
  - impongono attività di lunga durata (lettore audio)

Versioni più recenti di iOS per iPad consentono di eseguire più app in foreground contemporaneamente con la tecnica split-view; per iPhone, la modalità PiP (Picture in Picture) permette la condivisione dello schermo solo per particolari app (es. FaceTime).
Android non pone particolari limiti alle applicazioni eseguite in background, grazie all'introduzione dei "servizi", cha hanno permesso di disaccoppiare l'interfaccia dalla funzionalità.
Un'applicazione in elaborazione in background deve utilizzare un servizio, un componente applicativo separato, che viene eseguito per conto della app (es. app per streaming audio - se l'app va in background, il servizio continua comunque ad inviare il file audio al driver).
I servizi non hanno un'interfaccia utente e hanno un ingombro di memoria moderato, rendendo questa tecnica estremamente efficace per il mobile multitasking.

Creazione di un processo

Un processo può essere creato, non solo dall'utente, ma anche da un altro processo. Il processo padre crea processi figli che, a loro volta, creano altri processi, formando un albero di processi (a destra); il processo progenitore in un sistema Unix è detto "init".



Un processo può essere padre, figlio, foglia (non ha figli) e/o orfano.
Un processo può essere padre, figlio, foglia (non ha figli), adottato (un processo figlio "sopravvive" alla terminazione del processo padre e viene adottato da un altro processo) e/o orfano (non ha padre).

La relazione padre-figlio è importante:

- nella condivisione delle risorse, dove si possono verificare tre scenari:
    - Il padre e i figli condividono tutte le risorse
    - I figli condividono un sottoinsieme delle risorse del padre
    - Il padre e i figli non condividono risorse
- negli approcci di esecuzione:
    - il padre ed i figli vengono eseguiti in concorrenza
    - il padre attende la terminazione dei processi figli

- nell'uso degli spazi degli indirizzi:
    - il processo figlio è un duplicato del processo padre
    - nel sistema figlio viene caricato un diverso programma