Diario delle lezioni — Sistemi Operativi

Marino Miculan

A.A. 2002-2003

Indice

1 Introduzione ai Sistemi Operativi (30/9)

Administrivia: struttura del corso, di Lab S.O., esami, bibliografia,...

Cosa è e cosa fa un sistema operativo: prime risposte. Obiettivi di un S.O.: uso efficiente delle risorse, implementazione di una macchina astratta. Posizione del S.O. in un sistema di calcolo. Evoluzione dei sistemi operativi.

2 Evoluzione dei Sistemi Operativi (1/10)

Evoluzione dei sistemi operativi, con esempi di funzionalità introdotte via via (spool, multiprogrammazione, time-sharing, multiutente,...). Trasferimento di caratteristiche dei S.O.: dalle macchine più grandi a quelle più piccole, mano a mano che quest'ultime aumentano in potenza e prestazioni.

Richiami all'architettura dei calcolatori elettronici: componenti fondamentali, loro funzionamento di base.

3 Architettura degli elaboratori (7/10)

Comunicazione CPU-dispositivi di I/O: busy-wait, polling, interrupt, DMA. Interruzioni software e hardware: definizioni, esempi, gestione (vettore di interruzione, stack,...).

Protezione hardware: istruzioni privilegiate, i due modi di esecuzione della CPU (*user* e *supervisor*). System calls.

Componenti comuni dei sistemi operativi: Gestione dei processi, Gestione della Memoria Principale, Gestione della Memoria Secondaria, Gestione dell'I/O, Gestione dei file, Sistemi di protezione, Connessioni di rete (*networking*), Sistema di interpretazione dei comandi.

Servizi offerti dai sistemi operativi: Esecuzione dei programmi, Operazioni di I/O, Manipolazione del file system, Comunicazioni, Individuazione di errori. Addizionali: Allocazione delle risorse, Accounting, Protezione.

4 System call, e Struttura dei S.O. (8/10)

Implementazione dei servizi: le *system calls*. System calls: tipi, funzionalità offerte. Meccanismi di passaggio dei parametri tra codice utente e codice kernel: in registri, in tabelle puntate da registri, sullo stack.

Programmi di sistema: l'ambiente offerto insieme al sistema operativo. Tipi di programmi di sistema.

Struttura dei sistemi operativi: struttura semplice (DOS, Unix vecchia maniera); svantaggi. Struttura stratificata (OS/2): vantaggi in modularità. Macchine virtuali: robustezza, complessità implementativa, svantaggi nella comunicazione tra processi, esempi (DOS sotto Windows, JVM,...). Exokernel.

5 Microkernel; Meccanismi e Politiche (14/10)

Verso il microkernel: distinzione tra *meccanismi* e *politiche*. Idea di microkernel: spostare in user space tutte le politiche e lasciare in kernel solamente i meccanismi. Funzionamento di base di un microkernel. Vantaggi e svantaggi dei microkernel. Esempi di sistemi operativi basati su microkernel.

Struttura di Mach, Windows 2000, Mac OS X.

6 Intro a Unix; Processi (15/10)

Storia di Unix: dal papà MULTICS, alla divisione tra System V e BSD. Dallo standard POSIX a UNIX98 dell'Open Group. Ascesa di Linux e GNU. La situazione attuale.

Caratteristiche fondamentali del sistema UNIX, principî di progetto. Struttura a strati: hardware, kernel, librerie, applicativi. Struttura del kernel: a strati (tradizionale), modulare (moderna). Interfaccia per il programmatore. Alcuni esempi di chiamate di sistema.

Processi: concetto di processo, risorse. Switch di contesto.

7 Processi (22/10)

Processi: diagramma degli stati, process control block. Scheduling: switch di contesto, code di scheduling dei processi, migrazione dei processi.

Gestione delle interruzioni.

Scheduler a breve termine: controllo del time sharing. Scheduler a lungo termine: controllo del job mix. Processi CPU-bound e I/O-bound. Scheduler a medio termine: controllo della multiprogrammazione.

8 Processi in Unix; Thread (28/10)

Modelli di esecuzione (a kernel separato, a processo che esegue codice kernel, a microkernel).

Processi in UNIX: modello di esecuzione "ad iceberg", modello della memoria, creazione, terminazione. Ciclo fork-execve nella shell.

Strutture dei processi in Unix: process structure, user structure, text structure. Informazioni contenute. Posizione nella memoria. Real e effective UID/GID.

Cosa succede al *fork()*: cosa viene copiato, cosa no, cosa viene condiviso. Cosa viene sostituito con la *execve*.

Thread. Concetti fondamentali: separazione tra unità di allocazione risorse e unità di esecuzione, cosa è del processo e cosa invece del thread.

9 Thread (29/10)

Thread a livello utente (ULT) e a livello kernel (KLT). Caratteristiche, differenze, vantaggi e svantaggi. Esempi di utilizzo (word processor, http server). Scheduling a livello utente e a livello kernel, comportamento su macchine multiprocessore, costo di creazione/distruzione. Thread CMU, POSIX, Mach. Implementazione dei thread. Informazioni di thread e di processo. Esempi di implementazione. La soluzione Solaris: i Light Weight Process, loro utilizzo e flessibilità.

10 Thread; Scheduling di breve termine (4/11)

Scheduling di breve termine: concetti base, obiettivi, parametri da ottimizzare. Primi algoritmi: First Come-First Served, Shorted Job First, Shortest Remaining Time First, Round Robin a code multiple, con/senza feedback.

11 Scheduling di breve termine (11/11)

Schedulazione garantita. Schedulazione a lotteria.

Cenni a scheduling multiprocessor: tipi (omogenei, eterogenei), pinning, problemi, soluzioni (AMP, SMP).

Sistemi real-time: differenze tra *hard* e *soft* real-time, caratteristiche dello scheduling soft real-time. Processi schedulabili. Algoritmi per realtime: Rate Monotonic Scheduling, Earlier Deadline First.

Riduzione del tempo di latenza: kernel non prelazionabile, con punti di prelazione, pienamente prelazionabile.

Lo scheduling di Unix tradizionale: code multiple round robin, con feedback. Code a priorità negativa per i processi in modalità kernel.

12 Esempi di Scheduling di breve termine. Programmazione concorrente (12/11)

Scheduling di SVR4: principî, classi di default. Esempio: lo scheduling di Solaris. Comandi *priocntl* e *dispadmin*.

Scheduling di Linux: supporto per real-time, politiche FIFO, RR, OTHER.

Scheduling di Windows 2000.

Programmazione concorrente. Processi cooperanti. Vantaggi della suddivisione dei task tra più processi. Svantaggi: problemi di sincronizzazione e condivisione. Esempio: il produttore/consumatore su un buffer limitato condiviso. Race condition: definizione, caratteristiche, esempio nel produttore/consumatore sulla variabile condivisa.

13 Problema della Sezione Critica (18/11)

Problema della sezione critica. Requisiti di una soluzione. Prime soluzioni: disabilitazione degli interrupt. Soluzioni software: alternanza stretta, algoritmo di Peterson, algoritmo del fornaio.

Supporto hardware alla sincronizzazione. Istruzioni di Test&Set.

Evitare i cicli busy-wait: *sleep* e *wakeup*. Esempio del produttore-consumatore.

14 Semafori (19/11)

Semafori. Esempi: mutua esclusione per sezione critica, produttore-consumatore, sincronizzazione. Implementazione. Mutex. Necessità della memoria condivisa.

Deadlock e starvation: definizione.

Monitor: definizione, esempio, problematiche implementative.

Passaggio di messaggi: definizione, problematiche implementative. Comunicazione sincrona vs. asincrona. Esempio: produttore-consumatore.

Barriere.

I Grandi Problemi Classici: cinque filosofi.

15 Problemi Classici; IPC di Unix: semafori (25/11)

I Grandi Problemi Classici: lettori/scrittori, il barbiere.

I meccanismi IPC di System V. I semafori in Unix: creazione (chiamata *semget(2)*), utilizzo (chiamata *semop(2)*). Descrizione funzionamento e caratteristiche dei semafori interi (multivalore) e multipli (array di semafori).

16 IPC di Unix: shmem, mmap, Solaris (26/11)

La memoria condivisa: creazione (chiamata *shmget(2)*) e utilizzo (chiamata *shmat(2)*). Descrizione funzionamento e caratteristiche della memoria condivisa: volatilità, protezioni. Esempio: i programmi `pline` e `cline`. Race condition possibili in `pline`, `cline`, loro risoluzione con semafori nei programmi `sempline` e `semcline`.

Implementazione alternativa: il memory mapping attraverso la chiamata *mmap(2)*. Differenze rispetto alla memoria condivisa. Esempio: i programmi `mmpline` e `mmcline`.

Monitoraggio stato IPC: i comandi *ipcs(1)* e *ipcrm(1)*.

Primitive di sincronizzazione di thread in Solaris (cenni): semafori, mutex adattativi, variabili condition, lock di lettura/scrittura.

17 IPC di Unix: pipe, code di messaggi, segnali (13/1)

Le pipe: creazione (chiamata *pipe(2)*), utilizzo. Descrizione funzionamento e caratteristiche delle pipe. Applicazioni: situazioni produttore-consumatore. Esempio: il programma `whosort.c`. Named pipe: la creazione con *mknod(2)*.

Le code di messaggi: creazione (chiamata *msgget(2)*), utilizzo (chiamate *msgrcv(2)*, *msgsnd(2)*). Descrizione funzionamento e caratteristiche.

Segnali: cosa sono, da dove vengono. I segnali POSIX. Comportamenti di un processo in seguito ad un segnale. Esempio: un interrupt handler. Settaggio del signal handler, invio dei segnali, tracing del processo. Possibili usi di un segnale: ignorarlo, uscire in maniera pulita, riconfigurazione dinamica, (dis)attivare il debugging, riportare tabelle interne, implementazione di un timeout.

18 Deadlock I (14/1)

Introduzione ai deadlock. Risorse: prerilasciabili, non prerilasciabili. Protocollo di accesso ad una risorsa non prerilasciabile. Uso di semafori mutex per rappresentare le risorse. Problemi con più risorse: attese circolari. Problema di determinare una sequanza di allocazione sicura. Politiche di scheduling safe: FSCS. Non safe: Round Robin. Problemi: riconoscere la possibilità di deadlock, riconoscere un deadlock, risolvere un deadlock.

Definizione di Deadlock. Condizioni necessarie per il verificarsi del deadlock (mutua esclusione, hold&wait, mancanza di prerilascio, catena di attesa circo-

lare). Grafo di allocazione risorse. Uso ipotetico del grafo di allocazione risorse in uno scheduler.

Gestione del deadlock. Quattro alternative: fare niente, riconosce e curare un deadlock, evitare dinamicamente le situazioni pericolose, garantire che una delle quattro condizioni necessarie non valga.

Ignorare il problema: nel caso in cui i costi siano troppo alti.

19 Deadlock II (27/1)

Secondo approccio: identificazione e risoluzione dei deadlock. Algoritmi di identificazione per risorse singole e risorse multiple. Uso degli algorimi di identificazione. Risoluzione dei deadlock: prerilascio, rollback, terminazione.

Terzo approccio: eludere i deadlock. Traiettorie di risorse. Stati sicuri, sequenze sicure. Algoritmo del banchiere.

Quarto approccio: prevenzione dei deadlock. Falsificare una delle condizioni necessarie.

Approccio combinato. Blocco a due fasi.

20 Gestione della Memoria I (28/1)

La gerarchia della memoria. Obiettivi della gestione della memoria. Monoprogrammazione. Multiprogrammazione: Performance di utilizzo CPU, analisi della performance. Collegamento degli indirizzi di dati e istruzioni alla memoria fisica: compile time, linking time, run time. Caricamento dinamico. Collegamento dinamico. Spazio di indirizzi logici e fisici. La MMU: mappatura indirizzi virtuali in indirizzi fisici.

21 Gestione della Memoria II (3/2)

Allocazione contigua: partizione singola, partizionamento statico, partizionamento dinamico. Frammentazione interna. Algoritmi di allocazione: first-fit, next-fit, best-fit, worst-fit. Swapping.

Allocazione non contigua: paginazione, segmentazione. Esempi. Protezione e condivisione in ogni situazione. Supporto hardware necessario, traduzione degli indirizzi logici in indirizzi fisici in ogni situazione.

22 Gestione della Memoria III (4/2)

Implementazione della paginazione: hardware necessario, paginazione ad un livello, costi e degrado delle performance. Translation lookaside buffer: hardware, software. Calcolo del tempo di accesso effettivo. Principio di località. Paginazione a più livelli. Tabella delle pagine invertita. Segmentazione paginata: la soluzione MULTICS (segmentazione con paginazione a 1 livello), la soluzione IA32 (segmentazione con paginazione a 2 livelli).

23 Memoria virtuale I (10/2)

Memoria virtuale. Paginazione on demand: benefici, costi. Page fault: definizione, gestione. Sostituzione di pagina: costi, limiti architetturali. Performance. Copy-on-write, memory-mapped I/O.

Algoritmi di rimpiazzamento: FIFO (anomalia di Belady). Algoritmi di stack.

Algoritmi OPT, LRU, NFU. Implementazioni a contatori e a stack di LRU. CLOCK e CLOCK migliorato.

24 Memoria virtuale II (11/2)

Thrashing. Principio di località. Gestire il thrashing: modello del working set, limitazione del page fault. Implementazioni dell'algoritmo del working set. Algoritmo WSClock.

Sostituzione globale verso locale. Allocazione dei frame: libera, equa, a priorità. Free list con buffering. Prepaging. Effetti della dimensione della pagina. Influenza della programmazione sul page fault rate.

25 Gestione della Memoria in Unix (17/2)

Il caso UNIX. Quando risulta necessario liberare memoria: dopo una FORK, una BRK, uno stack overflow, un copy-on-write, o uno swap in.

Prima del 3BSD, e di SVR4, l'unico modo per liberare memoria era lo *swapping*. Dal 3BSD: la paginazione. Struttura della memoria fisica in 4BSD: kernel, mappa centrale (core map), frames.

La paginazione on demand in Unix: chi alloca le pagine (il kernel), e chi le libera: il processo 2, *pagedaemon*. Quando viene attivato, cosa fa. Algoritmo di paginazione two-handed clock. Variante con isteresi. Il processo 0 di Unix: *swapper*, o *scheduler*. Quando si attiva. Descrizione del funzionamento.

Come monitorare il funzionamento della memoria virtuale: lettura del comando *ps(1)*. , *vmstat(1M)*, *top(1)*; il tool *perfmeter*.

26 Gestione della Memoria in Windows (18/2)

Il caso Windows 2000. Modello della memoria. Stati di una pagina. Algoritmo di allocazione. Algoritmo di paginazione. Thread del kernel che si occupano della paginazione.

Esercitazioni.

27 I/O 0: Riassunto di Hardware (24/2)

Tipi di dispositivi I/O. Meccanismi di comunicazione tra CPU e controller: mappatura in memoria, con spazio separato. Modi di I/O: programmed I/O, interruzioni (vettore di interrupt), DMA, DVMA.

Gestione degli interrupt. Interruzioni precise, non precise. Effetti sulle architetture pipeline e superscalari. Diverse soluzioni (in hardware, o in software, nel S.O.).

28 I/O I (25/2)

Le componenti hardware/software dell'I/O: controller, driver, sw device/independent, sw a livello utente. Evoluzione di un sistema di I/O.

Interfaccia di I/O per le applicazioni: uniformità, incapsulamento. Parametri e caratteristiche dei dispositivi. Classi di dispositivi: I/O a blocchi, a carattere, mappato in memoria, socket di rete. Casi che esulano da queste classi (timer, orologio).

I/O bloccante, non bloccante, asincrono.

Sottosistema di I/O del kernel: funzionalità richieste (scheduling, buffering, caching, spooling, accesso esclusivo, gestione degli errori).

29 I/O II (3/3)

Descrizioni di ogni strato del sottosistema di I/O. I driver degli interrupt. I driver dei controller. Lo strato device-independent (interfaccia uniforme, denominazione uniforme, bufferizzazione, gestione errori, allocazione/rilascio dei dispositivi ad accesso esclusivo). Software a livello utente: librerie (esempio: `printf`), processi di sistema (demoni).

Traduzione delle richieste di I/O in operazioni hardware: comunicazione tra gli strati all'atto di un I/O.

30 I/O III (4/3)

Performance: costi nella gestione di I/O. Esempio: il telnet. Ottimizzazioni. Livello di implementazione delle funzionalità.

Gestione dei dischi. Algoritmi di scheduling dei dischi: FCFS, SSTF, SCAN, C-SCAN, LOOK, C-LOOK. Considerazioni.

Gestione dello spazio di swap: soluzioni 4.3 BSD, Solaris, Windows.

Affidabilità dei dischi: i sistemi RAID. Livello 0, 1, 4, 5.

31 I/O Unix 1 (10/3)

Input/output in Unix. Obiettivi. Principali chiamate di sistema: *open(2)*, *close(2)*, *write(2)*, *read(2)*, *ioctl(2)*, *lseek(2)*, *mmap(2)*, *fsync(2)*...

Visione astratta dei device: i file in `/dev`. Come leggere `ls -l /dev`: device a blocchi/a carattere, major e minor number. Ruolo dei major e minor number: tabelle dei device, parametro dei driver. Esempi. Il comando `mknod`. Protezione dei device.

Stuttura del kernel I/O secondo 4.3BSD. Compiti di ogni strato. Funzionalità dello strato device-independent. Funzionalità implementate da un device driver in Linux: la struttura `file_operations`. Registrazione di un driver. Come la chiamata di libreria risale alla corrispondente routine del driver.

32 I/O Unix 2 e Windows (11/3)

Accesso *cooked* e *raw* (*canonical* e *noncanonical*, secondo POSIX). Per i dispositivi a blocchi: cache, ottimizzazione dei tempi di accesso ai dischi. Il controllo di un dispositivo a caratteri: il comando *stty(1)*.

STREAM di System V: idea di base, funzionamento, esempi.

Monitoraggio I/O: il comando *iostat(1M)* di Solaris.

Il sottosistema di I/O di Windows 2000. Caratteristiche generali. I/O Asincrono. Il *Microsoft Driver Model*. Gli I/O request packets (IRP). Il PnP manager: riconoscimento dei dispositivi, caricamento dei driver, inizializzazione dei device e strutture correlate. Stack di driver in Windows.

33 File system I (14/4)

I file sono delle astrazioni di dati persistenti. Struttura, nomi, modi di accesso, protezione, e implementazione dei file sono peculiari del S.O. Aspetti generali. Attributi dei file: i metadati.

34 File system II (15/4)

Operazioni sui file. Esempio di uso dei file. File mappati in memoria. Directory. Strutture tipiche di directory: piatte, ad albero, DAG, a grafo. Caratteristiche, problematiche.

35 File system III (28/4)

Protezione: matrice di accesso, sua decomposizione in *access control list* o *capability list*. La protezione in UNIX: bit di modo, dominio del processo. Come restringere l'accesso ad un file ad un gruppo di utenti.

Struttura del file system. Tabella dei file aperti. Mounting e unmounting dei file system.

36 File system IV (29/4)

Allocazione dei blocchi: contigua, concatenata, indicizzata. Vantaggi e svantaggi di ogni soluzione. Esempi: FAT16, FAT32. Inodes di Unix: Contenuto e utilizzo di un inode: blocchi diretti, primi, secondi, terzi indiretti. Limiti massimi di un file: imposti dal file system, imposti dall'offset. I *largefile(5)*.

37 File system V (5/5)

Gestione dello spazio libero: bitmap, linked list.

Implementazione delle directory: a dimensioni fisse, a heap; lineari, ad hash, ad albero bilanciato.

Problemi di efficienza e performance. Influenza della dimesione del blocco. Cache. Recovery da failure: utility di ripristino della consistenza (scandisk, fsck). File system journalled, esempi.

38 Esempi di file system (6/5)

La FAT di Windows. Le directory in DOS e Windows.

Il file system di UNIX. File system supportati, file system nativi (UFS, EXT2, EFS, SysV). Dimensioni dei blocchi in EXT2, SysV. Blocchi e fragment in UFS.

Cosa succede nel kernel quando si apre un file: caricamento dell'i-node in memoria. Lista degli i-node in core, la file structure, la tabella dei file aperti per processo. Dove risiedono queste strutture, e perché ci sono (importanza della file structure). Il comando *sync(1m)*.

Le directory: struttura di BSD (nomi fino a 255 caratteri, link simbolici, ...). Hard links, le entry per `.` `...` Traduzione pathname (relativo e assoluto) nel relativo i-node.

39 Esempi di file system; Sistemi a più processori (12/5)

Esempio di file system fisico: il file system EXT2 di Linux: partizioni, bootblock, superblock, gruppi, tabella degli i-nodes, blocchi.

Collegamento del file system al logical file system: il comando *mount(1m)*. Overlay dei file system. Al boot: / è montato dal kernel, gli altri vengono montati dalla tabella `/etc/fstab`. Come vengono tradotti i pathname nel caso incontriamo un i-node che sia un mount point: la tabella `/etc/mnttab`. Altri file system montati con `mount`.

Usi di fsck: verifiche di integrità del file system prima del mounting, riparazione dei block link counters, recupero del superblock.

Il File System NTFS: principi di progetto, caratteristiche di base. Il Master File Table (MFT): struttura, contenuti. Esempi di record. Attributi residenti, non residenti. File: allocazione, immediati, non residenti, run. Directory: corte (liste), lunghe (B+tree).

Introduzione ai sistemi a più processori. Processori tightly-coupled, loosely-coupled. Sistemi multiprocessore, multicomputer, distribuiti: differenze.

Sistemi multiprocessore: SMP UMA, NUMA.

40 Sistemi multiprocessore e multicomputer (13/5)

Sistemi operativi per multiprocessore: locale, AMP, SMP.

Multicomputer: motivazioni, differenze con i multiprocessori. Topologie di connessione per multicomputer. Interfacce di rete. Programmazione nei multicomputer con `send` e `receive`. `send` bloccante versus non bloccante. Programmazione con RPC. Difficoltà delle RPC.

41 Sistemi di rete e distribuiti (19/5)

Il modello client/server. Motivazioni all'uso dei sistemi distribuiti: condivisione delle risorse, accelerazione della computazione, aumento dell'affidabilità, comunicazione. Confronto con gli altri modelli di calcolo parallelo.

Hardware dei sistemi distribuiti: reti LAN, WAN.

Programmazione nei sistemi distribuiti: necessità di un modello di calcolo comune. Il *middleware*.

Servizi di rete e servizi distribuiti. Esempi.

Servizi di rete: orientati alla connessione, senza connessione. Affidabilità del servizio. Esempi di diversi protocolli.

42 Servizi e Sistemi di rete (20/5)

Protocolli. Modello ISO/OSI. Stack TCP/IP. Principali protocolli.

Communication endpoint per servizi di rete: le socket. Sono strumenti di IPC tra macchine remote. Tipi di socket, famiglie di indirizzamento.

Strutture dati per le socket. Chiamate di sistema *socket(3n)*, *bind(3n)*, *listen(3n)*, *accept(3n)*.

Monitoraggio delle socket con il comando *netstat(8)*.

I *daemon* in UNIX. I runlevel: significato, funzionamento degli script, sequenza di startup. I servizi di UNIX (protocolli a livello applicazione): la tabella */etc/services*, le porte di Unix; porte riservate e loro motivazione. Il super server `inetd`.

43 Servizi e Sistemi distribuiti (26/5)

Obiettivi dei sistemi distribuiti. Problemi di progetto: trasparenza e località, mobilità dell'utente, tolleranza ai guasti, scalabilità, sistemi su larga scala, struttura dei processi server.

Robustezza di un sistema distribuito: rilevamento dei guasti, riconfigurazione, reintegro delle parti dopo la riparazione.

Modelli di sistemi distribuiti basati su migrazione dei dati: WWW, Lotus Notes, AFS, Coda, NFS, SMB.

44 RPC, NFS (27/5)

Modelli di sistemi distribuiti basati sulla migrazione delle computazioni (RPC, RMI, CORBA); sulla migrazione dei processi/thread (MOSIX, DCE); sulla coordinazione distribuita (Linda, Jini).

Esempi di servizi distribuiti. Schema di chiamata remota (RPC). Porta, servizio, associazione statica e dinamica (portmapper).

Il Network File System: concetti base, schema di funzionamento via RPC. Caratteristiche e architettura del sistema. Condivisione di un file system via rete: esportazione e montaggio NFS.

Riferimenti bibliografici

[1] Andrew S. Tanenbaum. *I moderni sistemi operativi*, II edizione. Jackson, 2002.