Sistemi operativi
per il corso di diploma universitario in Ingegneria Elettronica
anno accademico 1999/00

Agenda delle lezioni dell'A.A. 1999-00

Posizione originaria: <URL:http://fly.cnuce.cnr.it/didattica/agenda99.html>.

Abbreviazioni usate nell'agenda:

:2L
: 2 ore di lezione (sviluppo di nuovi argomenti)

:1E
: 1 ora di esercitazioni (esercizi in classe)

:1S
: 1 ora di laboratorio (pratica in classe o in laboratorio)

T: 9.1, 9.5…; S:7.6; P: 2.1-2.12
: Argomenti trattati nel testo di riferimento Tanenmbaum, capitoli 9.1 e tutto il 9.5, in Silberschatz, capitolo 7.6, e nel manuale PSOS, da pagina 2-1 a pagina 2-12.

Cos'è un sistema operativo

T: 1, S: 1-3

- [1:1L] Struttura a strati della macchina, processore, microprogramma, linguaggio macchina, sistema operativo, librerie e compilatori.
- S.O. come macchina virtuale estesa, che presenta un'interfaccia uniforme e semplice a diversi dispositivi (p.es. accesso al disco).
- S.O. come gestore delle risorse, che ne consente l'uso coordinato a più programmi, anche contemporaneamente (p.es. uso della stampante).
- [2:2L] Sistema monoprogrammato, multiprogrammato, condivisione di tempo, real-time hard e soft, prelazione.
- Protezione hardware, spazio utente e supervisore, trap, interrupt, system call, eccezioni.
- [3:2L] Processi: creazione, esecuzione, morte, figli, attesa, segnali, mascheramento, proprietà di proprietario e di gruppo.
- [4:1L] Memoria: gerarchie di memorizzazione, prezzo, velocità, volatilità.
- File: directory, working directory, nomi assouti e relativi, apertura, chiusura, accessi, diritti, lock, file speciali (device), pipe.
- [5:2L] Kernel monolitici (Unix), a strati (Multics), macchine virtuali (VM), microkernel (Mach). Macchine virtuali vecchio stile e nuova maniera.
- Sistema monoprocessore, multiprocessore, sistemi ridondanti, sistemi distribuiti.
- Shell, sistema a finestre. Programmi che accompagnano il sistema operativo. Interfaccia utente, interfaccia di programmazione. Solo meccanismi (DOS), solo politica (Macintosh).

Boot, compilazione, link, caricamento, esecuzione

T: 3.1.3, 3.7, 8.3.2; S: 3.8, 8.1…, 13.3.2

- [6:2L] Processo di boot da ROM e da disco, boot a più stadi. Settore di boot, tabella delle partizioni, master boot record.
- Compilazione di un sorgente, oggetti rilocabili e non rilocabili, tabella di rilocazione.
- Risoluzione dei riferimenti esterni, tabella dei simboli, librerie, link.
- Caricamento, caricatori rilocanti e non, esecuzione.
- [7:1L] Codice indipendente dalla posizione. Struttura di un file oggetto, di una libreria, di un eseguibile, di un'immagine per ROM. Core dump.
- Overlay. Librerie dinamiche.

I processi

T: 2.1…; S: 4.1…, 4.3…

- [8:2L] Programma, (istanze di un) processo.
- Creazione di un processo in DOS: interprete dei comandi, TSR. load_and_exec, load_overlay, end_prog, keep_prog. Le routine del DOS non sono rientranti, come si fa una routine rientrante.
- Creazione di un processo in Unix: fork, exec, exit.
- Creazione di un processo in PSOS: t_create, t_start.
- Stato di un processo: nuovo, pronto, in esecuzione, bloccato, zombi. Transizioni possibili fra gli stati, blocco, sblocco, prelazione, scheduling.
- Process Control Block: stato della CPU, diritti, allocazioni di memoria, priorità, segnali, file aperti, accounting, blocco su una coda di eventi.

Scheduling

T: 2.4, 7.5.1; S: 4.2…, 21.5…, 22.4, 22.4.1…, 22.5.2;; P: 2.1-2.12

- [9:2L] Meccanismo: contesto di un processo, cambio di contesto, dispatcher, dispatch latency.
- Rescheduling scatenato da interrupt: vettore degli interrupt, cambiamento di stato, gestore dell'interrupt, salvataggio del processo in esecuzione, sblocco di un altro processo, rescheduling.
- Politica: algoritmi di scheduling. Time slicing. Round robin, priorità, round robin con priorità. Posix: priorità statiche e dinamiche, politiche standard, Fifo e Rr.
- Criteri di valutazione: utilizzo della CPU (efficienza), throughput (es.: numero di transazioni per secondo), tempo di completamento di un task, tempo di risposta.
- Il kernel esegue una chiamata di sistema per conto del processo. Chiamate interrompibili: Solaris2.
- [10:1L] Una generica politica di scheduling: multilevel feedback queue.
- La politica classica di scheduling in Unix, priorità dinamica, nice level, rescheduling. Implementazione in Linux. Politiche Posix per le applicazioni real-time: Fifo e Round robin, priorità statiche, sched_yield.
- [11:2L] Scheduling in PSOS: occasioni di rescheduling, prelazione e condivisione di tempo, priorità.

[11:2E] Esercitazione: struttura di un file oggetto, dettagli sulle tabelle di rilocazione e dei simboli.

[12:2E] Esercitazione: struttura di un file eseguibile, esempio di file ELF.

[13:1E] Prova in itinere: rispondere a tre domande scritte sul programma svolto sinora.

Sincronizzazione: concetti, mutua esclusione

T: 2.2.1, 2.2.2, 2.2.3, 2.2.4; S: 6.1, 6.2, 6.3

- [14:2L] Spooler di stampa che usa un buffer di nomi di file di lunghezza infinita e due variabili condivise che indicano la base e la cima dello stack dei nomi: race condition quando c'è più di uno scrivente. Concetti di mutua esclusione e sezione critica.
- Condizioni per il funzionamento di processi sincronizzati con mutua esclusione, supponendo velocità relative dei processi imprevedibili:
  - Non più di un processo dentro la sezione critica.
  - Un processo fuori della sezione critica non ne può bloccare altri.
  - Il tempo di attesa per entrare nella sezione critica è finito.
- Mascherare gli interrupt è un'operazione privilegiata, ed è poco furbo darne la possibilità ad un processo utente. Inoltre, su sistemi multiprocessore non è generalmente sufficiente. Tuttavia è una tecnica usata dentro il kernel per implementare sezioni critiche dentro gli interrupt handler.
- Due processi che usano una risorsa alternativamente possono usare una variabile di turno (alternanza). Questo metodo è utilizzabile anche fra più dispositivi in DMA, per esempio è il metodo usato dal chip LANCE per sincronizzare l'accesso ad un anello di buffer. Tuttavia non è generale, perchè un processo non può entrare due volte di seguito nella sezione critica.
- Per togliere il vincolo dell'alternanza, si può usare un array di variabili booleane (una per processo) per registrare le prenotazioni. Ma non funziona, perché anche con due soli processi può causare deadlock.
- [15:2L] Soluzione particolare: produttore-consumatore con un anello di lunghezza finita e due variabili `in` e `out` che sono indici nell'array. Si spreca un posto, ma funziona per un singolo produttore ed un singolo consumatore.
- Esiste una soluzione generale per la mutua esclusione fra due processi, inizialmente proposto da Dekker e semplificata da Peterson. Consiste nel combinare la variabile di turno con le variabili booleane di prenotazione. È una soluzione che soddisfa tutte le tre condizioni dette. Ne esiste una versione per un numero dato di processi, più complicata.

```
// Ci sono 2 processi: un processo si chiama 0 e l'altro si chiama 1.
// Quindi se uno si chiama p, l'altro si chiama 1-p.
int turn = 0;
boolean interested[2] = { false, false };

void enter_critical_section (int p)
{
  interested[p] = true;
  turn = p;
  while (interested[1-p] && turn == p)
    continue;
}

void exit_critical_section (int p)
{
  interested[p] = false;
}
```

- Proviamo a semplificare usando una singola variabile di lock:

```
int lock = 1;
…
while (lock == 0) continue; lock = 0; // enter critical section
…                                      // sezione critica
lock = 1;                              // exit critical section
```

Non funziona, perché ci riporta esattamente allo stesso problema che si tenta di risolvere. Se però si rende atomica l'operazione sul lock, allora funziona (su monoprocessore), anche se il tempo di attesa non è limitato:

```
int lock = 1;
…
while ((lock -= 1) < 0) lock += 1;    // enter critical section
…                                      // sezione critica
lock += 1;                             // exit critical section
```

- In alternativa, su una macchina monoprocessore, basta disabilitare gli interrupt per rendere atomiche le operazioni sulla variabile di lock. Su multiprocessore non è pratico, e quindi è necessario supporto hardware: test and set lock (TSL) per risolvere la concorrenza sullo stesso processore e su diversi processori.

Sincronizzazione: semafori e monitor

T: 2.2.5, 2.2.7; S: 6.4…, 6.5.1, 6.5.2, 6.7(l'inizio); P: 2.49-2.50

- [16:1L] Semafori.
- Tutte le precedenti soluzioni usano attese attive. Sleep e wakeup.
- [17:2E] Esercizi: un produttore, un consumatore, buffer finito. Più produttori e consumatori con mutex unico sul buffer. Più produttori e consumatori con due mutex sul buffer.
- Esercizio: lettori e scrittori, precedenza ai lettori.
- Due processi che devono bloccare due sezioni critiche, che devono essere accessibili anche da altri: possibile stallo. Si evita evitando di invertire l'ordine dei lock, ma in situazioni complesse può essere impossibile.
- [18:2L] Implementazione dei semafori in IPC (Unix), con trucco per semplificare problemi di stallo (più operazioni atomiche). Non c'è garanzia sull'ordine in cui i processi vengono rilasciati. Operazione bloccante o no, a scelta. Incremento o decremento di quantità arbitrarie, uso come variabile di lock.
- Implementazione in PSOS: semplice incremento o decremento. Code FIFO o a priorità. Operazione bloccante, non bloccante, bloccante con timeout.
- Definizione di monitor: mutua esclusione su una sezione critica e operazioni di wait e signal sulle variabili di monitor. Una signal su una variabile su cui nessun processo è fermo non ha alcun effetto. Chi fa una signal è sospeso, o è sospeso l'altro. In Concurrent Pascal chi fa la signal esce.

Sincronizzazione e comunicazione: messaggi

T: 2.2.8, 2.2.9; S: 4.6.1, 4.6.2, 4.6.3, 4.6.4; P: 2.44;2.46

- Messaggi come primitive di sincronizzazione: semplice ordinamento parziale fra due processi, mutua esclusione con mailbox condivisa, accesso multiplo ad una sezione critica con mailbox condivisa. Si dimostra (noi no) l'equivalenza coi semafori ed i monitor.
- Esercizio: implementare il caso un produttore - un consumatore con buffer finito e messaggi vuoti. Mettendo dentro i messaggi l'indice del posto nel buffer si può avere concorrenza totale fra più produttori e più consumatori.

- [19:1L] Caratteristiche di un sistema a scambio di messaggi:
  - Connessione globale con mailbox da inizializzare, o per processo, automatica: in questo caso, receive con indirizzo del mittente o senza
  - Nomi dei corrispondenti e della casella postale
  - Quanti processi possono usare la connessione
  - Proprietà e diritti sulla mailbox
  - Copia del messaggio o mappatura in memoria del ricevente
  - Niente buffer (rendezvous), dimensione finita o infinita senza wait: nessuna garanzia sulla ricezione, si può usare un ack
  - Dimensione dei messaggi fissa o variabile, piccola o grande
  - Connessione unidirezionale o bidirezionale
  - Connessione affidabile o no; chi verifica la perdita
  - Condizioni di errore: mailbox distrutta, interruzione della chiamata, mancanza di buffer di sistema
- [20:2L] Implementazione dei messaggi in IPC (Unix): messaggi di lunghezza arbitraria, code di lunghezza limitata, messaggi con tipo, ricezione su qualunque messaggio, solo tipo dato, tutti eccetto tipo dato, tipo pari ad almeno il dato. Operazioni bloccanti o no, a scelta. A scelta, il messaggio da ricevere troppo lungo dà errore o tronca il messaggio.
- Implementazione dei messaggi in PSOS: code a lunghezza limitata per coda o per sistema, messaggi senza tipo lunghi 16 byte, code dei task gestite con politica FIFO o secondo la priorità dei task. La send non è mai bloccante, la receive può essere bloccante, non bloccante, bloccante con timeout. Il messaggio può esser messo in testa o in coda alla coda di messaggi. Si può mandare un broadcast di un messaggio a tutti i task in attesa su una coda: se non ce ne sono, il messaggio è perduto, altrimenti tutti i task ricevono una copia del messaggio, e la coda rimane vuota e senza task in attesa.

Condizioni di stallo (deadlock)

T: 6.1, 6.2, 6.3, 6.4.3, 6.6, 6.7.1; S: 7.1, 7.2, 7.3, 7.4, 7.7

- Processi e risorse, ciclo richiesta → utilizzo → rilascio, risorse con più istanze. Esempi di risorsa: spazio temporaneo su disco, un particolare file con lock in scrittura, memoria da allocare, CPU.
- Condizioni necessarie per uno stallo:
  1. Presenza di risorse accedute in mutua esclusione
  2. Presenza di processi che impegnano delle risorse e sono in attesa su altre.
  3. Impossibilità di prelazione sulle risorse impegnate.
  4. Attesa circolare nel grafico di allocazione delle risorse.
- Grafico di allocazione delle risorse, gli anelli sono condizione necessaria, non sufficiente.
- Metodi per gestire gli stalli: ignorare, prevenire, evitare, curare. *Ignorare* è il più diffuso, perché dal punto di vista ingegneristico è quasi sempre conveniente. In Unix ci sono potenzialità di deadlock per accesso alla tabella dei processi, o dei file aperti, o in generale di qualunque tabella di lunghezza fissa. Ma anche per lo spazio di spool.
- *Prevenire* significa evitare almeno una delle quattro condizioni di stallo.
  1. La prima si può evitare il più possibile, cercando di ridurre la mutua esclusione solo ai casi necessari.
  2. La seconda può essere evitata in casi particolari allocando a priori tutte le risorse necessarie. Un esempio ne è il two phase lock protocol, utilizzato per esempio per l'accesso a database.
  3. La terza si può evitare il più possibile, cercando di attivare la prelazione su più risorse possibili.
  4. La quarta si può evitare dando un ordinamento totale alle risorse, e consentendo i loro impegno solo in ordine crescente.

- *Evitare* significa effettuare a tempo di esecuzione dei controlli atti ad evidenziare le situazioni di possibile stallo. Si usa il concetto di *stato sicuro* del sistema ed un algoritmo detto *del banchiere*. Tuttavia l'algoritmo del banchiere richiede che il processo dichiari a priori di che risorse ha bisogno. Come se non bastasse, lo stato sicuro è condizione sufficiente per evitare uno stallo, ma non necessaria, per cui riduce l'efficienza del sistema. Calcolare se lo stato è sicuro è un'operazione O(risorse·processi²), e richiede la gestione di apposite strutture dati.
- *Curare* implica due passi: individuazione dello stato di stallo e rottura dello stallo. L'individuazione dello stato di stallo è pur essa un'operazione di complessità O(risorse·processi²) e richiede la gestione di apposite strutture dati. Deve essere eseguita ad ogni possibile operazione pericolosa, o periodicamente, o per induzione quando l'uso della CPU è basso. Una volta individuata la condizione di stallo, si può effettuare una prelazione manuale su alcuni tipi di risorsa, o tramite riavvolgimento (rollback) dello stato di un processo al più recente checkpoint, o riavvolgimento all'inizio, uccidendo un processo alla volta finchè il ciclo si rompe. La scelta del processo da uccidere non è banale e, se deterministica, può riportare alla stessa situazione di stallo, determinando attesa infinita del processo (starvation).

Panoramica sulla gestione della memoria

T: 3.1, 3.1.1, 3.1.3, 3.2, 3.2.1, 3.2.3, 3.2.5, 3.3, 3.3.1, 3.3.2 (prima parte), 3.4.7, 3.6.1; S: 8.2, 8.3, 8.4…, 8.5.1, 8.5.2…, 9.1, 9.4, 9.5.3, 9.7, 9.7.1, 9.8.1

- [21:2L] Uso della memoria in DOS, lista dei processi in memoria e delle aree di memoria libere e occupate (PSP). Partizioni fisse e variabili, compattazione, first fit, next fit, best fit, worst fit. Frammentazione interna ed esterna.
- Swap dell'immagine di un processo su disco. Paginazione: scompare la frammentazione esterna, si può fare swap di parte di un processo. Esempi di memoria virtuale più grande o più piccola della memoria fisica, tabella delle pagine come lista di coppie fisico-virtuale, bit di validità, bit di presenza. Pagina mancante e sua gestione, possibile numero di page fault generati da una singola istruzione.
- Località dei riferimenti, working set, thrashing. Algoritmo LRU, bit di riferimento, invecchiamento come approssimazione di LRU. Prepaginazione, paginazione su domanda, bit di modifica. Il codice di un programma può non essere mai messo in swap, ma restare sul file system.

Memoria virtuale: paginazione

T: 3.6.4; S: 8.5.5, 9.2, 9.3, 9.8.5, 9.8.6

- [22:1L] Meccanismo: le voci della tabella delle pagine non contengono l'indirizzo virtuale da tradurre, che invece è usato come indice. Dimensione eccessiva, accenno alle pagine a più livelli, alla cache della MMU e alla possibilità di swap delle tabelle stesse. Bit di cache abilitata, pagina valida, riferita, modificata, presente, diritti (o protezione).
- Tabella delle pagine per processo ed inverted page table per sistema (tabella dei frame) dove fra l'altro c'è scritto per ogni frame se è libero oppure a quale processo, dispositivo o altro è attualmente allocato.
- Memoria virtuale per il real-time: costo della traduzione fisico↔virtuale effettuata dall MMU, della necessità di traduzioni dentro il kernel per parlare coi task e coi dispositivi di i/o, maggior costo del cambio contesto. Protezione preferibile in ambienti complessi o eterogenei.
- Un *bus error* è generato per accesso a memoria fisica o virtuale inesistente. Protezione fra task: spazi separati; dentro il task: indirizzo 0 non valido e sola lettura per il codice. *Segmentation violation* ver violazione dei diritti.

- [23:1L1E] Cambio di contesto implica svuotamento della cache di memoria. Con memoria virtuale implica anche svuotamento della cache di MMU. Con swapping può anche causare swap-in delle pagine di memoria del nuovo processo ed anche delle sue tabelle delle pagine.
- Swap device o backing store. Swap su più partizioni. Memoria virtuale coincidente con lo spazio di swap (BSD) o memoria fisica più swap (Linux). I dati non variabili (codice) possono essere lasciati sul disco al loro posto o subire swap-out una volta in memoria per velocizzarne l'accesso (BSD). Blocco delle pagine in memoria su chiamata di sistema, buona per real-time e per DMA su aree di utente. Mai usare swap per real-time.
- Copia su scrittura ottimizza l'uso della memoria e minimizza le copie inutili, massimizzando la condivisione delle pagine. Su fork, il nuovo processo ha tutte le pagine in sola lettura. Ad ogni segmentation violation, si alloca una nuova pagina che è una copia della vecchia, altrimenti la pagina resta condivisa. Se c'è copia su scrittura, è importante separare i dati in sola lettura dagli altri, e scrivere codice rientrante (non modificante) per le librerie.

Esercizio: in quali casi è generata un'eccezione di pagina mancante?

Esercizio: mentre l'accesso ad una pagina non valida genera un'eccezione di errore si bus (*bus error*), l'accesso ad una pagina non presente genera un'eccezione di pagina mancante (*page fault*), che si usa per gestire la paginazione su domanda. Descrivere più edttagliatamente possibile la sequenza di azioni che porta alla sostituzione di una pagina dallo swap.

1. Accesso a pagina *valida* ma non *presente*, con *diritti* corretti.
2. La MMU genera un *page fault*.
3. La CPU gestisce l'eccezione e chiama il gestore.
4. Il gestore verifica che ci sia un frame libero.
   a) Se non c'è un frame libero, si decide quale pagina va liberata.
      - Se la pagina da liberare è sporca (modificata), si inizia il suo trasferimento sullo swap.
      - Cambio di contesto, la CPU va ad un altro processo. Il primo processo è bloccato in attesa non interrompibile.
      - Arriva un interrupt che segnala la fine del trasferimento della pagina sporca sul disco.
   b) Che sia avvenuto lo swap o no, la pagina da liberare ora è pulita (non modificata): è marcata come non valida e il frame corrispondente è marcato come libero.
5. La pagina che ha generato il page fault poteva essere di memoria non inizializzata mai acceduta prima. In questo caso, non si fa niente. Oppure deve essere letta dal file system (codice o dati) o dallo swap, comunque dal disco.
   a) Se è sul disco, si inizia il trasferimento da disco.
   b) Cambio di contesto, la CPU va ad un altro processo. Il primo processo è bloccato in attesa non interrompibile.
   c) Arriva un interrupt che segnala la fine del trasferimento da disco.
6. La pagina viene marcata come valida e non modificata, e associata al frame che era stato liberato, e che ora è marcato non libero.
7. Il processo che aveva generato il page fault è sbloccato e messo nella coda dei pronti.

[24:2E] Esercizio: in quali casi è generata un'eccezione di errore di bus?

Esercizio: in quali casi è generata un'eccezione di violazione di segmentazione?

Esercizio: come sono implementate le operazioni shmget, shmat e shmdt nel kernel di Unix?

Esercizio: come si implementano le librerie condivise (solo dal punto di vista dell'uso della memoria virtuale)?

Esercizio: come è implementato l'algoritmo di invecchiamento delle pagine che si usa per scegliere la pagina da sostituire?

[25:1E] Prova in itinere: rispondere a tre domande scritte sul programma svolto sinora.

[26:2E] Esercizi: commenti alle domande fatte nella prova in itinere.

Chiamate di sistema e di libreria per gestire la memoria

T: 3.7; S: 8.6…, 8.7, 9.5.4…, 9.6, 9.6.1

- Le chiamate per la creazione, aggancio e sgancio di memoria condivisa in IPC.
- Allocazione di memoria in Unix: la chiamata di sistema brk. Le funzioni Posix per l'allocazione della memoria: malloc, free, calloc, realloc.
- [27:2L] Allocazione di memoria in PSOS. Regioni contigue di memoria fisica, segmenti allocabili di dimensione arbitraria con attesa FIFO o priorità. Region create, delete, getseg, retseg. Partizioni contigue di memoria fisica, buffer allocabili di dimensione fissa senza attesa.

Device

T: 5.2…; S: 12.3…

- Dispositivi a blocco, da cui ci si aspetta read, write, seek; a carattere, da ci ci si aspetta get, put. Dispositivi mappati in memoria, ad esempio in Unix come mmap, munmap. I socket sono usati soprattutto per le comunicazioni in rete.
- In Posix e in PSOS, un driver è implementato mediante le interfacce di open, close, read, write e ioctl, più una routine di gestione dell'interrupt. Le interfacce dei driver sono quindi uniformi, essenziale per implementare *device independence*. In Unix è implementato anche il concetto di *uniform naming*.
- Operazioni su device: bloccanti; nonbloccanti sincrone, usate per il polling; asincrone, con meccanismo di callback implementato con mecanismi tipo segnali (SIGIO in BSD Unix). La select consente il blocco contemporaneo su più device, anche con timeout.
- Per scrivere un driver in Posix e PSOS basta scrivere le cinque routine di open, close, read, write, ioctl e l'interrupt handler. Il sistema operativo si occupa dell'interfaccia. Open e close, se non servono, possono essere vuote. Per semplici dispositivi implementano l'accesso serializzato alla risorsa, o effettuano controlli di diritti prima di garantire l'accesso.

I file system

T: 4.1, 4.1.4, 4.3.1 (senza I-nodes), 4.3.2 (solo MS-DOS), 4.5.2, 5.3.2 (ultimo capoverso); S: 10.1.1, 10.1.2, 11.2.2, 11.2.3, 10.4.2, 10.4.4, 13.5

- [28:1L] Le funzioni di filesystem vedono sotto di loro il driver, che presenta il disco come una sequenza lineare di settori, e presentano sopra un'interfaccia con open, close, read, write, seek.
- Implementazione dei file system: tabella di allocazione dei file. Tenere una lista dei blocchi sul disco con puntatore nel blocco è poco furbo. La FAT in DOS: 12, 16, 24, 32 bit. Voci da 24 bit con cluster da 1 KB $\Rightarrow$ partizione da 16GB al più.
- In DOS si alloca il file con algoritmo first fit. In ext2fs si usa worst fit.
- Le directory in DOS, accenno alle directory in Unix.
- Accesso sequenziale e casuale. Tabella dei file aperti di sistema, con le caratteristiche del file.

Tabella dei file per processo, con puntatore per accesso sequenziale. Accenno ai problemi per l'accesso concorrente.

- [29:2L] Diritti: lettura, scrittura, esecuzione, modifica, cancellazione, lista, cambiamento di nome. Diritti di uso dei file in Unix e in NT (liste di accesso). Diritti sulle directory in Unix. Il bit suid.
- Backup, restore. RAID 0 (striped), 1 (mirror), 4 (striped with centralised checksum), 5 (striped with distributed checksum).

Esercizio: qual è la sequenza di eventi che ha luogo quando si alloca memoria con una chiamata di realloc?

Laboratorio: progetto di un device driver per il chip LANCE

- [30:1S] Cosa serve per implementare un driver per il LANCE: open, close, interrupt handler, read e write sincrone e asincrone.
- [31:2S] Operazioni di i/o sincrone e asincrone. Code e semafori in PSOS, operazione t_mode usata per disabilitare la prelazione.
- [32:2S] Eventi, segnali asincroni

Esercizio: qual è la sequenza di eventi che ha luogo quando si apre un file in un sistema operativo di tipo Unix? E quando si fa una read?

Esercizio: apro un file in lettura e scrittura e ottengo un errore perché non ho diritti di scrittura. Cosa è successo esattamente?

[33:1E] Prova in itinere: rispondere a tre domande scritte sul programma svolto sinora.