

SCHEDULING

Danilo Croce

Novembre 2024



LO SCHEDULING DEI PROCESSI/THREAD

- In un computer multiprogrammato, molteplici processi/thread possono competere per la CPU contemporaneamente.
- Lo scheduler decide quale processo/thread eseguire successivamente seguendo un algoritmo di scheduling.
 - Molti problemi di scheduling per processi valgono anche per i thread.
- Lo scheduling al livello del kernel avviene per thread, indipendentemente dal processo di appartenenza.
- Sfide specifiche emergono nello scheduling dei thread.



UN PO' DI STORIA

- Nei sistemi (storici) **batch**, lo **scheduling** era **lineare**
 - Si eseguiva il lavoro successivo sul nastro.
- Con la **multiprogrammazione**, lo **scheduling** è diventato **complesso** a causa della concorrenza tra utenti.
- Gli algoritmi di scheduling erano cruciali per la prestazione e la soddisfazione dell'utente nei mainframe.
- Nei personal computer:
 - Spesso un solo processo è attivo.
 - La CPU raramente è una risorsa scarsa: la maggior parte dei programmi è limitata dalla velocità dell'input dell'utente.



QUANTO COSTA IN TERMINI DI TEMPO?

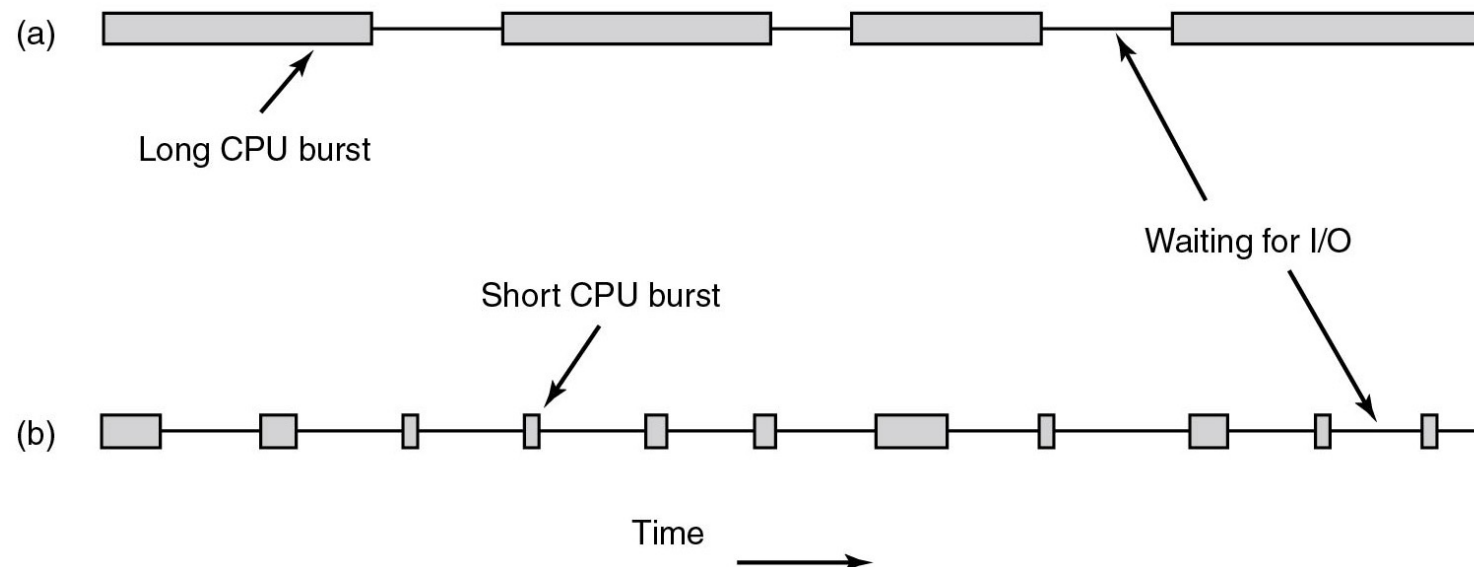
- Nei server in rete, la CPU spesso è contesa: lo scheduling torna ad essere vitale.
- Per i dispositivi IoT, gli smartphone e i nodi di sensori, la durata della batteria è un vincolo cruciale.
 - Lo scheduling può cercare di ottimizzare il consumo energetico.
- Lo scambio di processi (o "**context switch**") è oneroso:
 - Cambio da modalità utente a modalità kernel
 - Salvataggio dello stato del processo
 - Esecuzione dell'algoritmo di scheduling
 - Cambio della mappa della memoria
 - Invalidazione potenziale della memoria cache (prossime lezioni...)
- Troppe commutazioni possono consumare tempo di CPU:
 - la prudenza è essenziale.



INTRODUZIONE AL PROBLEMA DI SCHEDULING DEI PROCESSI

I processi alternano fasi di elaborazione CPU-intense con richieste di I/O.

- a) **Compute-bound (CPU-bound):** Burst di CPU lunghi, attese di I/O infrequenti.
- b) **I/O-bound:** Burst di CPU brevi, attese di I/O frequenti. Sono tali a causa della bassa necessità di calcoli, non della durata delle richieste di I/O.

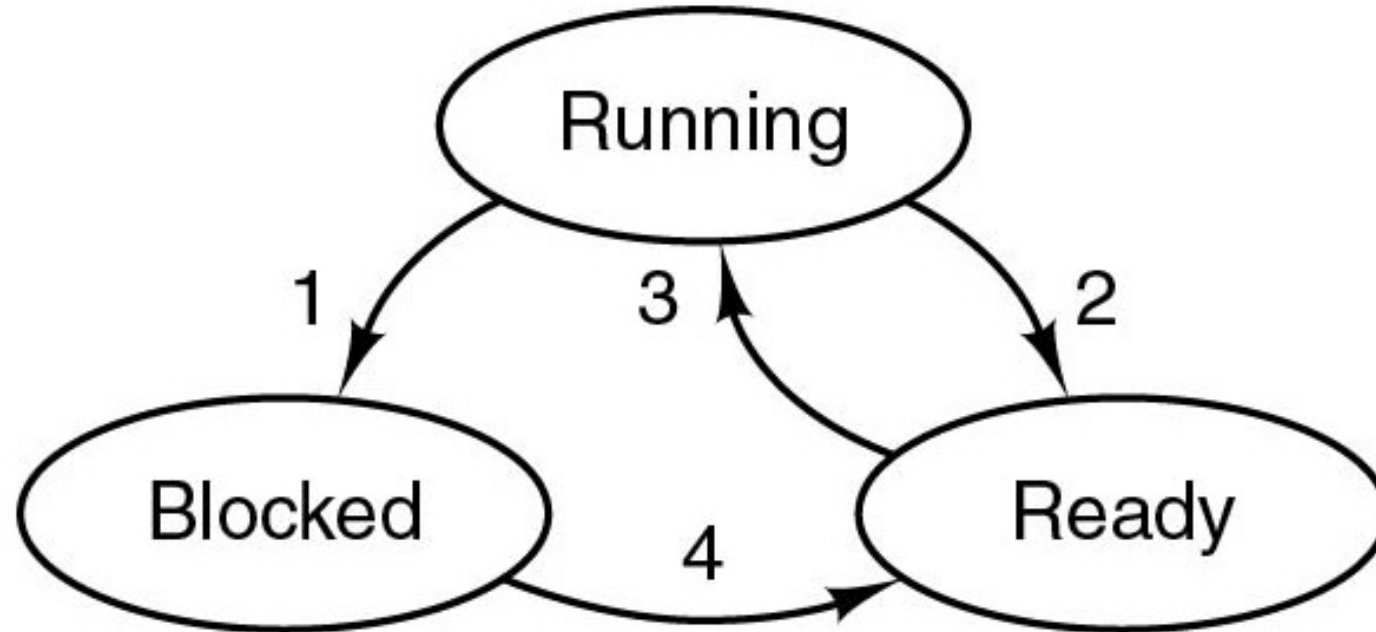


INTRODUZIONE AL PROBLEMA DI SCHEDULING DEI PROCESSI (2)

- Con **CPU più veloci**, i processi tendono a essere più **I/O-bound**.
 - CPU e dischi magnetici non stanno avanzando rapidamente in velocità.
- **SSD sostituiscono gli hard disk nei PC**, ma i data center utilizzano ancora HDD per il costo per bit.
- Lo scheduling **varia in base al contesto**.
 - Ciò che funziona per un dispositivo potrebbe non essere efficace per un altro.
 - Il panorama tecnologico è in continua evoluzione.



PROCESS STATE REVISITED



- Se ci sono più processi pronti che CPU disponibili:
 - Lo scheduler decide quale processo eseguire successivamente.
 - L'algoritmo utilizzato dallo scheduler è chiamato **algoritmo di scheduling**.



SITUAZIONI IN CUI È NECESSARIO LO SCHEDULING

- **Creazione Nuovo Processo**

- Decisione tra l'esecuzione del processo genitore o figlio.
- Entrambi pronti: può essere scelto chiunque.

- **Uscita di un Processo**

- Se un processo esce, occorre scegliere un altro dai processi pronti.
- Se nessuno è pronto, occorre eseguire un processo inattivo del sistema.

- **Blocco del Processo**

- Se un processo si blocca (I/O, semaforo, etc.), occorre selezionarne un altro.
- A volte la causa del blocco può influire sulla decisione.

- **Interrupt di I/O**

- Alla conclusione di un I/O, un processo potrebbe diventare pronto.
- Decidere se eseguire il processo appena pronto, il precedente o un altro.



TIPOLOGIE DI SCHEDULING E PRELAZIONE

- **Non Preemptive (Senza Prelazione):**

- Seleziona un processo e lo **lascia eseguire fino al blocco o alla rilascio volontario**.
- **Nessuna decisione** durante gli interrupt del clock.
- Ripristina il processo precedente dopo l'interrupt, a meno che non ci sia una priorità superiore.

- **Preemptive (Con Prelazione):**

- Sceglie un processo e lo **lascia eseguire per un tempo massimo** definito.
- Se ancora in esecuzione **dopo il tempo, è sospeso** e viene scelto un altro.
- Richiede un interrupt del clock per restituire controllo allo scheduler.

- **Importanza della Prelazione:**

- Rilevante per le applicazioni e i kernel dei sistemi operativi.
- Necessaria per **prevenire** che un driver o una chiamata di sistema lenti **blocchino** la CPU.
- In un kernel con prelazione, lo scheduler può forzare un cambio di contesto.



DIVERSITÀ NEGLI AMBIENTI DI SCHEDULING

- **Batch:**

- Ideale per attività aziendali periodiche.
- Accetta algoritmi senza prelazione.
- Priorità a prestazioni efficienti.

- **Interattivo:**

- **Prelazione fondamentale.**
- Previene monopolizzazione della CPU.
- Adatto per server e utenti multipli.

- **Sistemi Real-time:**

- Processi spesso si bloccano velocemente sapendo di non poter eseguire a lungo.
- **Prelazione non sempre necessaria.**
- Eseguono programmi per specifiche applicazioni, a differenza dei sistemi interattivi che possono eseguire programmi arbitrari.



OBIETTIVI GENERALI DEGLI ALGORITMI DI SCHEDULING

- **Sistemi Batch:**

- **Throughput:** Numero di job completati in un tempo fissato.
- **Tempo di Turnaround:** Minimizzare il tempo dallo start all'end di un job.
- **Utilizzo della CPU:** Mantenere la CPU costantemente attiva.

- **Sistemi Interattivi:**

- **Tempo di risposta:** Risposta rapida alle richieste degli utenti.
- **Adeguatezza:** Soddisfare le aspettative dell'utente in termini di tempi di risposta.

- **Sistemi Real-time:**

- **Rispetto delle scadenze:** Assicurarsi che i dati vengano elaborati nei tempi previsti.
- **Prevedibilità:** Assicurarsi che il funzionamento sia costante, specialmente in sistemi multimediali per evitare degradi della qualità.



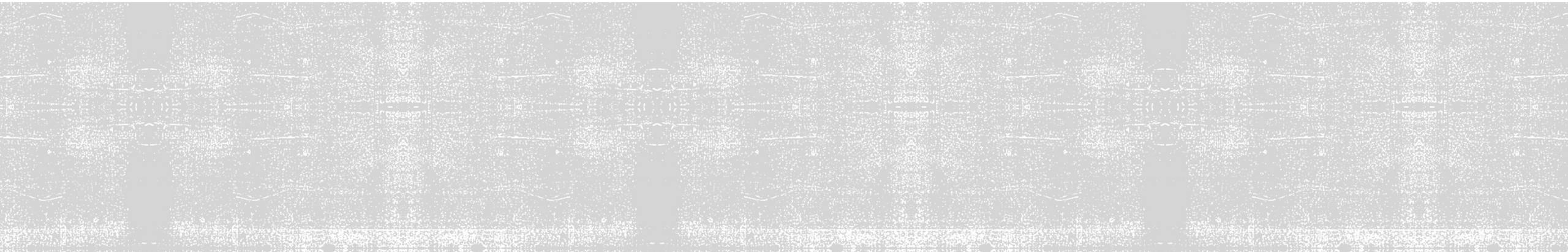
OBIETTIVI GENERALI DEGLI ALGORITMI DI SCHEDULING (CONT')

- **Tutti i sistemi:**
 - **Equità:** Garantire un'equa condivisione della CPU a tutti i processi.
 - **Imposizione della policy:** Garantire l'attuazione delle policy dichiarate.
 - **Bilanciamento:** Mantenere tutti i componenti del sistema attivi.
- L'equità è fondamentale in ogni scenario.
- In un sistema batch, è ideale combinare processi CPU-bound e I/O-bound.
- Nei sistemi real-time è cruciale rispettare le scadenze e garantire la prevedibilità





SCHEDULING IN SISTEMI BATCH



SCHEDULING IN BATCH SYSTEMS

- First-Come First-Served
- Shortest Job First
- Shortest Remaining Time Next



FIRST-COME, FIRST-SERVED (FCFS)

- **Descrizione:**

- Algoritmo di scheduling **senza prelazione**.
- Processi **assegnati alla CPU nell'ordine in cui arrivano**.
- Una singola coda di processi in stato pronto. Il primo job esegue immediatamente senza interruzioni.
- Processi bloccati ritornano in fondo alla coda.

- **Vantaggi:**

- Facile da capire e programmare.
 - Gestione semplice con una *linked list*.
- Equo in base all'ordine di arrivo.

- **Svantaggi:**

- Prestazioni non ottimali in scenari misti (es. processi CPU-bound e I/O-bound).
- Può risultare in tempi di attesa molto lunghi per processi I/O-bound in presenza di un processo CPU-bound.



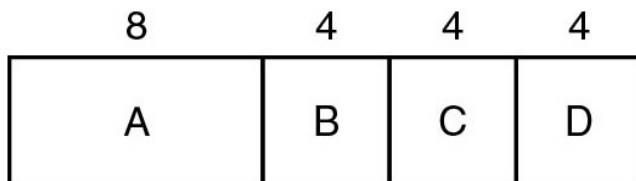
SHORTEST JOB FIRST (SJF)

- **Descrizione:**

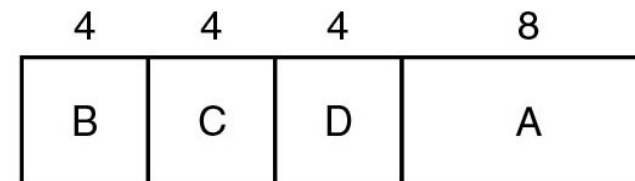
- Algoritmo batch **senza prelazione**.
- Richiede che i tempi di esecuzione siano noti in anticipo.
- Il job più breve viene eseguito per primo.

- **Esempio**

- 4 job (A, B, C, D) con tempi di 8, 4, 4 e 4 min.
- Esecuzione in ordine:
 - 8 min per A, 12 min per B, 16 min per C, 20 min per D (media 14 min).
- Esecuzione con SJF:
 - 4 min, 8 min, 12 min e 20 min (media 11 min).



(a)



(b)



OTTIMALITÀ DI SJF E CONSIDERAZIONI

- **Ottimalità:**

- Shortest Job First è **ottimale nel minimizzare il tempo di turnaround medio** (il tempo dallo start all'end di un job) quando tutti i job sono disponibili contemporaneamente.

- **Limitazione:**

- Se i job arrivano in momenti diversi, SJF potrebbe non essere ottimale.
- Esempio:
 - Job A-E con tempi 2, 4, 1, 1, 1 e arrivi a 0, 0, 3, 3, 3.
 - Due sequenze diverse producono medie di 4,6 e 4,4.



SHORTEST REMAINING TIME NEXT (SRTN)

- **Descrizione:**

- Versione con **prelazione** di SJF.
- Seleziona sempre il processo con il tempo rimanente più breve per completare.
- Il tempo di esecuzione deve essere noto in anticipo.

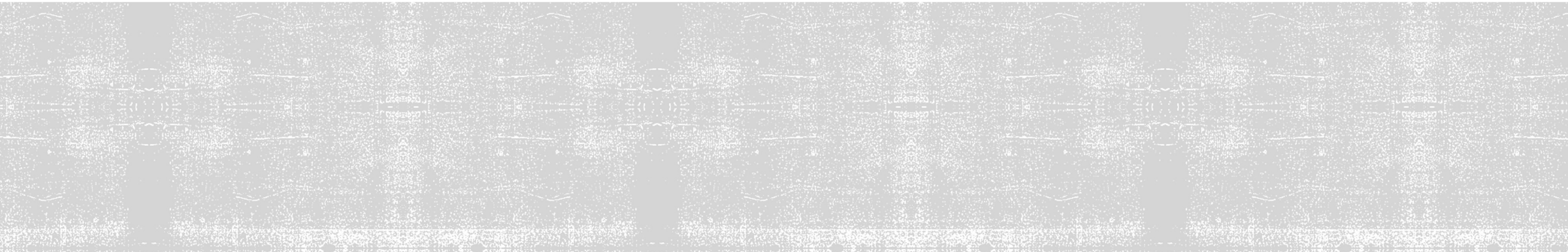
- **Funzionamento:**

- Confronta il tempo totale del nuovo job con il tempo rimanente dei processi in esecuzione.
- Se il nuovo job è più breve del processo corrente, sospende il processo corrente ed esegue il nuovo job.
- Assicura che i nuovi job brevi ricevano un servizio rapido.





SCHEDULING IN SISTEMI INTERATTIVI



SISTEMI INTERATTIVI

- **Tempo di risposta è fondamentale** rispondendo rapidamente alle richieste
- **Proporzionalità:** occorre soddisfare le aspettative degli utenti
- **Metodi:**
 - Round-Robin Scheduling
 - Priority Scheduling
 - Shortest Process Next
 - Guaranteed Scheduling
 - Lottery Scheduling
 - Fair-Share Scheduling



McClatchy-Tribune/Tribune Content Agency LLC/Alamy Stock Photo

Apple MacIntosh (1984)



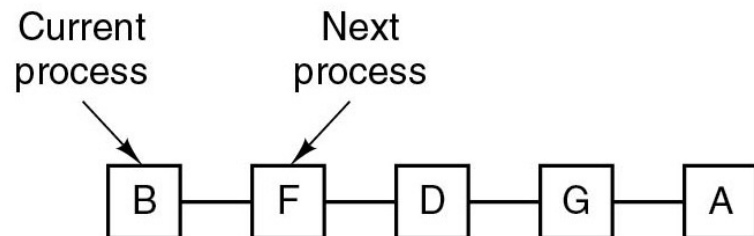
ROUND-ROBIN SCHEDULING

- **Concetto:**

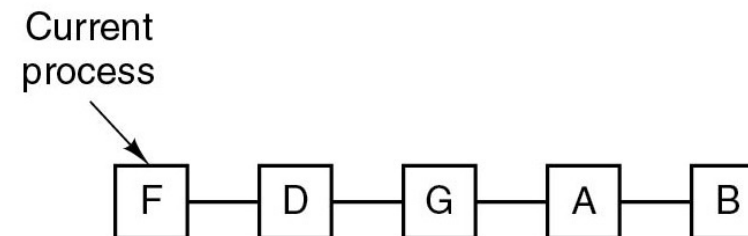
- Uno degli algoritmi di scheduling più **vecchi, semplici, equi e ampiamente utilizzati**.
- **Ogni processo riceve** un intervallo di tempo o "**quanto**" per l'esecuzione.
- Se il processo **non ha terminato** al termine del quanto, **la CPU è oggetto di prelazione** per un altro processo.
- Se un processo termina o si blocca prima del quanto, il passaggio avviene automaticamente

- **Implementazione:**

- Mantenere una lista dei processi eseguibili.
- Una volta esaurito il quanto, il processo viene spostato alla fine della lista.



(a)



(b)



SFIDE DEL ROUND-ROBIN E DURATA DEL QUANTO

- **Durata del Quanto:**

- La scelta del quanto influisce sull'efficienza.
- Supponendo 1 ms per il cambio di contesto e 4 ms per il quanto: il 20% del tempo CPU sprecato in overhead.

- **Trade-off:**

- **Quanto lungo:** riduce l'overhead, ma peggiora la reattività (es. 5 secondi di attesa per un breve comando in un server affollato).
- **Quanto breve:** maggiore overhead e riduzione dell'efficienza della CPU.

- **Ottimizzazione:**

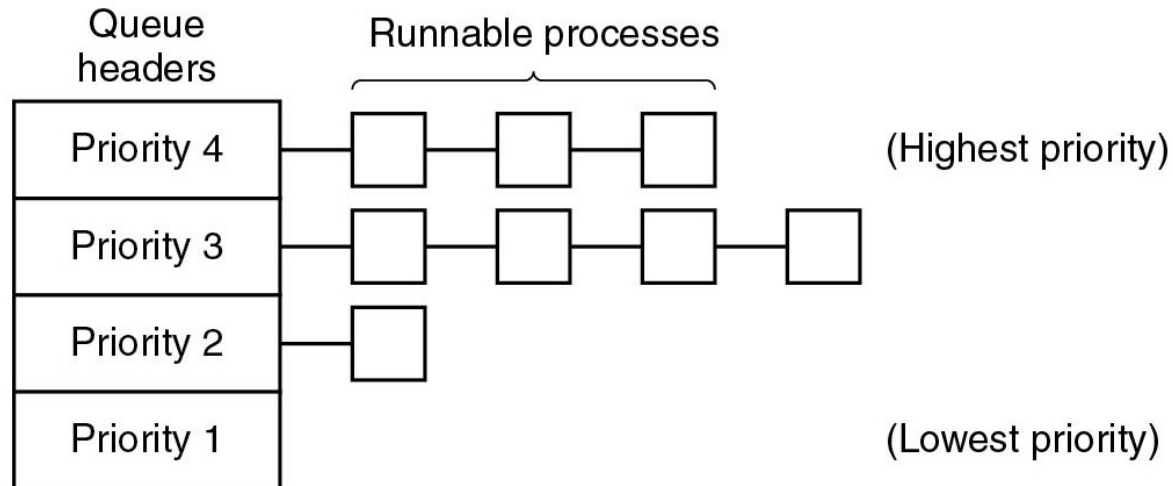
- Se il quanto è maggiore del tempo medio di burst di CPU, la prelazione potrebbe non avvenire spesso. Molti processi potrebbero bloccarsi prima.
- **Compromesso:** un quanto tra 20 e 50 ms è spesso ragionevole per bilanciare efficienza e reattività.



INTRODUZIONE ALLO SCHEDULING A PRIORITÀ

- **Premessa:**

- Round-robin considera tutti i processi ugualmente importanti.
- Alcuni contesti richiedono una gerarchia (es. università con differenti ruoli).



- **Scheduling a Priorità:**

- Ogni processo ha una priorità assegnata.
- La CPU esegue il processo con la priorità più alta tra quelli pronti.
- Applicabile anche su singoli PC:
 - ad es. un *daemon* di posta elettronica avrebbe meno priorità di un video in tempo reale.



FUNZIONAMENTO DELLO SCHEDULING A PRIORITÀ

- **Gestione delle Priorità:**

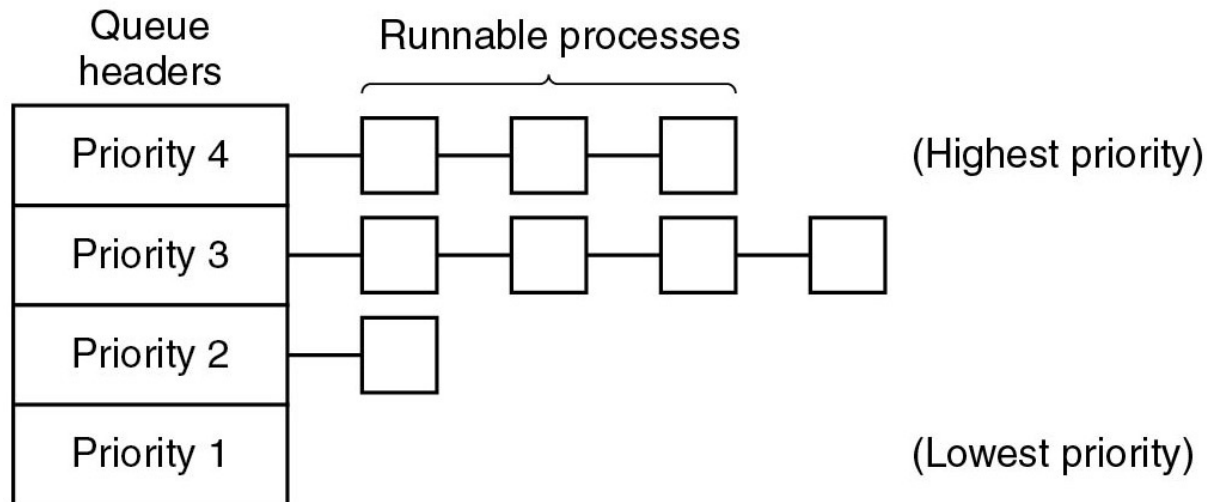
- **Priorità** del processo attualmente in esecuzione **può diminuire con il tempo**.
- **Se scende** sotto quella del processo successivo, **avviene uno cambio**.
- Possibilità di assegnare un quanto di tempo: al suo esaurirsi, si passa al processo con priorità appena inferiore.
- Evitare che processi rimangano inibiti indefinitamente, altrimenti potrebbero finire a priorità 0.

- **Priorità Statica vs Dinamica:**

- **Statica:** es. gerarchie militari o basate sui costi nel data center.
- **Dinamica:** es. basata sull'utilizzo della CPU o sul comportamento I/O bound.



STRATEGIE COMBinate E CLASSI DI PRIORITÀ



- **Raggruppamento in Classi:**

- Processi divisi in classi di priorità.
- Scheduling a priorità tra le classi, ma round-robin all'interno della stessa classe.

- **Esempio:**

- Sistema con 4 classi di priorità.
- Fintanto ci sono processi in priorità 4, si usano in round-robin. Se vuota, si passa alla 3, poi alla 2, e così via.
- Importante rivedere periodicamente le priorità per evitare che processi a bassa priorità non vengano mai eseguiti
 - Morendo di inedia ☹️



SHORTEST PROCESS NEXT CON AGING

- "Shortest Job First" ottimizza il tempo medio di risposta nei sistemi batch. L'obiettivo è applicarlo ai sistemi interattivi.
 - Ricordare che processi lunghi potrebbero aspettare e morire di inedia (*starvation*)
- **Sfida:** Identificare quale tra i processi eseguibili sia effettivamente il più breve.
- **Soluzione - Aging:** Fare stime basate sul comportamento passato.
 - Stima del tempo per un comando: T_0 .
 - Stima aggiornata dopo nuova esecuzione T_1 diventa $aT_0 + (1 - a)T_1$.
 - Scelta di ' a ' determina il peso delle esecuzioni precedenti nella nuova stima.
 - Esempio con $a = 1/2$:
 - T_0 , $T_0/2 + T_1/2$, $T_0/4 + T_1/4 + T_2/2$, ...
 - Dopo 3 esecuzioni, peso di T_0 nella stima è $1/8$.
 - E «passa avanti»
- **Utilizzo dell'Aging:** Prevista situazioni dove si basa la previsione su valori passati.



GUARANTEED SCHEDULING

- **Concetto Principale:**

- Fare promesse concrete sugli standard di prestazione e rispettarle.

- **Promessa Base:**

- Se ci sono n utenti o processi, ciascuno ottiene $\sim 1/n$ della potenza della CPU.

- **Come Funziona:**

- Il sistema tiene traccia di quanta CPU ha ricevuto ogni processo dal momento della sua creazione (esempio 100 secondi)
- Calcola quanto tempo CPU ogni processo dovrebbe avere
 - tempo da creazione $\div n$ (ad esempio 100 sec / 10 processi, un processo dovrebbe avere 10 secondi).
- Valuta il rapporto tra il tempo CPU consumato e quello dovuto.
 - Rapporto di 0,5: ha avuto metà di quanto dovuto.
 - Rapporto di 2,0: ha avuto doppio di quanto dovuto.
- Esegue il processo con il rapporto più basso finché non supera il suo concorrente più vicino.



SCHEDULING A LOTTERIA (LOTTERY SCHEDULING)

- **Concetto di base:**

- Assegnazione di biglietti della lotteria ai processi per le risorse del sistema, come il tempo della CPU.
- Estrazione casuale di un biglietto per decidere quale processo ottiene la risorsa.
- Esempio: Estrazione 50 volte al secondo => vincitore riceve 20ms di tempo della CPU.

- **Distribuzione delle probabilità:**

- Biglietti extra per processi più importanti => maggiori probabilità di vincere.
- Esempio: Se un processo ha il 20% dei biglietti, guadagnerà a lungo termine il 20% della CPU.



PROPRIETÀ E APPLICAZIONI DELLO SCHEDULING A LOTTERIA

- **Reattività:**
 - Risponde velocemente ai nuovi processi grazie alla distribuzione dei biglietti.
- **Cooperazione tra processi:**
 - Possibilità di scambiarsi biglietti tra processi cooperanti.
 - Esempio: Un processo client dona i suoi biglietti a un processo server per farlo eseguire più rapidamente.
- **Soluzione a problemi complessi:**
 - Adatto a situazioni dove altri metodi falliscono.
 - Esempio: Server video con diverse necessità di frequenze di fotogrammi.
 - Assegnazione di biglietti in base alla velocità necessaria => divisione automatica della CPU nelle porzioni corrette.



SCHEDULING FAIR-SHARE

- **Premessa:**

- Tradizionalmente, ogni processo è oggetto di scheduling individualmente.
- Es. Se l'utente 1 ha 9 processi e l'utente 2 ne ha 1, con round-robin o priorità uguali, l'utente 1 avrà il 90% della CPU, l'utente 2 solo il 10%.

- **Approccio Fair-Share:**

- Considera la proprietà di ogni processo prima di considerarlo.
- Ogni utente riceve una frazione predefinita di CPU.
- Lo scheduler si assicura che ogni utente riceva la sua frazione, indipendentemente dal numero di processi posseduti.



SCHEDULING FAIR-SHARE (2)

- **Esempio:**

- Due utenti, ciascuno con il 50% della CPU.
 - Utente 1 ha processi **A, B, C, D**;
 - Utente 2 ha processo **E**.
- Sequenza con round-robin:
 - **A E B E C E D E A E ...**
- Se l'utente 1 ha il doppio del tempo di CPU rispetto all'utente 2:
 - **A B E C D E A B E ...**

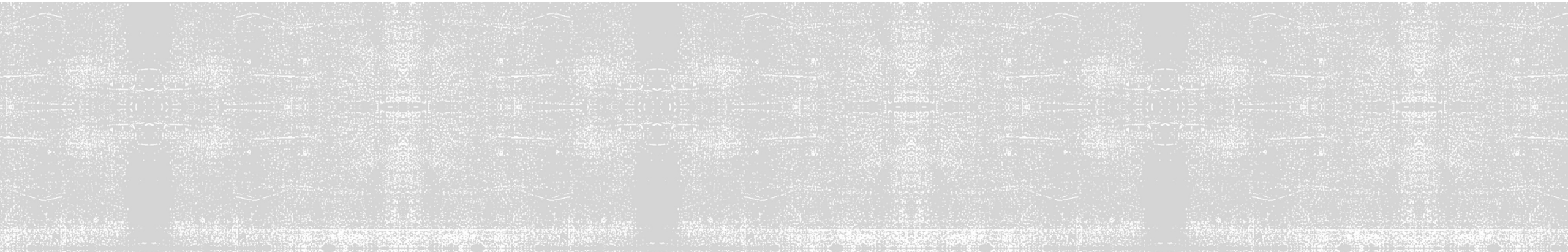
- **Versatilità:**

- Molti modi di implementare, basati sulla definizione di "equità".





SCHEDULING IN SISTEMI REALTIME



SCHEDULING NEI SISTEMI REAL-TIME

- Usato nei sistemi operativi in applicazioni in cui il tempo di risposta è fondamentale.
 - Lettori di compact disc, Monitoraggio in terapia intensiva, Piloti automatici, Controllo robotico in fabbriche
- Ritardi o mancati tempi di risposta possono avere gravi implicazioni.



Yauhen_D/Shutterstock



NASA



TIPI DI SISTEMI E CARATTERISTICHE REAL-TIME

- **Categorie:**
 - **Hard Real-Time:** Scadenze assolute da rispettare.
 - **Soft Real-Time:** Qualche scadenza mancata è tollerabile.
- **Comportamento:** Processi prevedibili, brevi e noti in anticipo.
- **Tipi di eventi:**
 - **Periodici:** Avvengono a intervalli regolari.
 - **Non Periodici:** Avvengono in modo imprevedibile.
- **Condizione di «Schedulabilità»:** La CPU deve essere in grado di gestire la somma totale del tempo richiesto dai processi.
- Per esempio, se ci sono m eventi periodici, l'evento i avviene con un periodo P_i e richiede C_i secondi di tempo della CPU per gestire ogni evento, allora il carico può essere gestito solo se

$$\sum_{i=1}^m \frac{C_i}{P_i} \leq 1$$



ESEMPI E TIPOLOGIE DI ALGORITMI REAL-TIME

- **Esempio:**

- Eventi periodici: 100ms, 200ms, 500ms.
- Tempi richiesti: 50ms, 30ms, 100ms.
- Condizione: $0,5 + 0,15 + 0,2 < 1$.

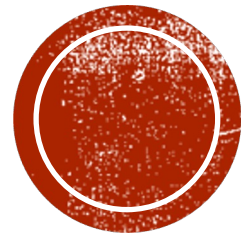
- **Algoritmi di Scheduling:**

- **Statici:** Decisioni prese prima dell'esecuzione.
- **Dinamici:** Decisioni prese durante l'esecuzione.

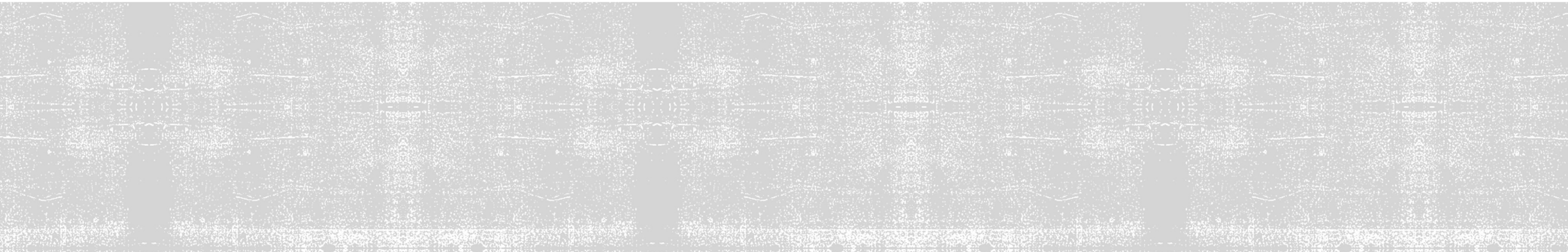
- **Limitazioni:** Lo scheduling statico richiede una perfetta conoscenza delle esigenze e delle scadenze.

	Periodo	Tempi Richiesti
P1	100 ms	50 ms
P2	200 ms	30 ms
P3	500 ms	100 ms





PROCESSI E SCHEDULING



PROCESSI E SCHEDULING

- **Premessa:** Abbiamo sempre considerato i processi come appartenenti a utenti differenti in competizione per la CPU.
- **Scenario reale:**
 - Un processo può avere molti processi figli sotto il suo controllo.
 - Esempio: Un sistema di gestione di una base di dati con molti figli, ognuno con funzioni specifiche.
- **Problematica:** Gli scheduler tradizionali non accettano input dai processi utente, spesso portando a decisioni sub-ottimali.



SEPARAZIONE TRA MECCANISMO E POLITICA DI SCHEDULING

- **Principio:** Separare il meccanismo di scheduling dalla politica di scheduling (Levin et al., 1975).
- **Vantaggio:** L'algoritmo di scheduling può essere parametrizzato, ma i parametri sono forniti dai processi utente.
- **Esempio pratico:**
 - Kernel con algoritmo di scheduling a priorità.
 - Chiamata di sistema permette a un processo di impostare le priorità dei suoi figli.
 - Il genitore può influenzare lo scheduling dei suoi figli senza controllarlo direttamente.
- **Conclusione:** Il meccanismo sta nel kernel, la policy è determinata dal processo utente - un concetto fondamentale.



PARALLELISMO: PROCESSI E THREAD

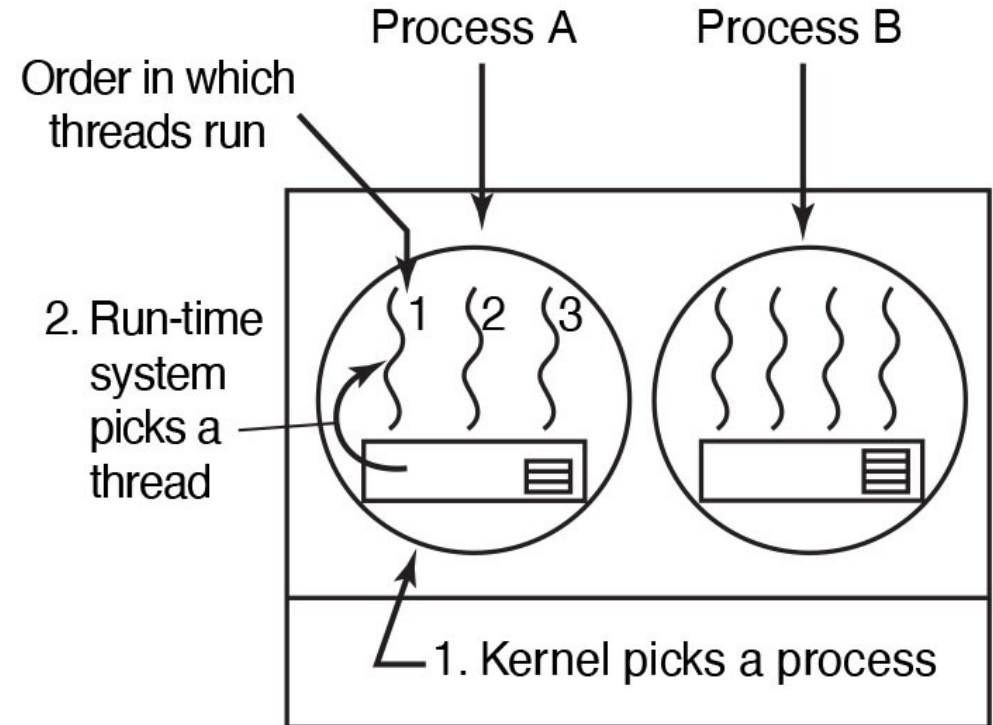
- Due livelli di parallelismo: **processi** e **thread**.
- Lo scheduling differisce in base al tipo di thread:
 - **livello utente** vs. **livello kernel**
- **Thread a livello utente:**
 - Il kernel ignora l'esistenza dei thread; sceglie un processo per il suo quanto.
 - Il thread interno decide quale thread eseguire senza interruzione del clock.
 - **Risultato:** Un thread può consumare l'intero quanto del processo, influenzando solo il processo interno e non gli altri.



SCHEDULING DEI THREAD A LIVELLO UTENTE

- Possibile ordine di esecuzione:
A1, A2, A3, A1, A2...
- Scheduling del sistema run-time può variare: spesso **round-robin** o **a priorità**.
- Cooperazione tra thread; nessuna interruzione forzata.

Possibile scheduling di thread a livello utente con un quanto per un processo di 50 msec e thread che eseguono 5 msec per burst della CPU.



Possible: A1, A2, A3, A1, A2, A3
Not possible: A1, B1, A2, B2, A3, B3

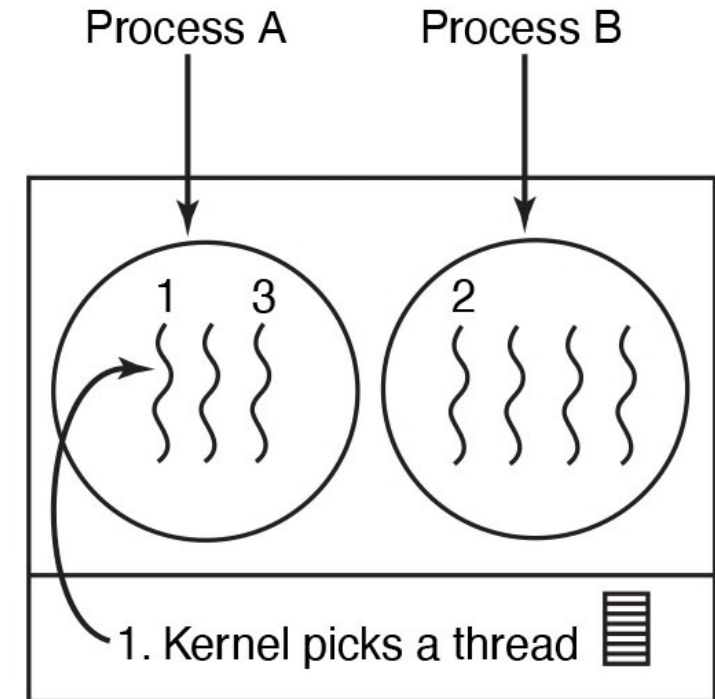
(a)



SCHEDULING DEI THREAD (2)

- **Thread del kernel:** Il kernel seleziona un thread specifico per l'esecuzione.
 - Ordine potenziale: A1, B1, A2, B2... (come in Figura (b)).
 - Se un thread eccede il quanto, viene sospeso.

Possibile schedulazione di thread a livello kernel con le stesse caratteristiche della diapositiva precedente



Possible: A1, A2, A3, A1, A2, A3

Also possible: A1, B1, A2, B2, A3, B3

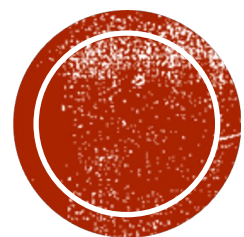
(b)



DIFFERENZE PRESTAZIONALI E CONSIDERAZIONI

- **Thread a livello utente vs. Thread del kernel:**
 - Scambio thread utente: poche istruzioni.
 - Scambio thread kernel: scambio completo di contesto => più lento.
 - Blocco su I/O:
 - con thread utente, intero processo sospeso;
 - con thread kernel, solo il thread specifico.
- **Decisioni del kernel:**
 - Considera i costi per passare da un thread a un altro.
 - Può dare preferenza ai thread dello stesso processo.
- **Scheduling specifico dell'applicazione:**
 - Permette maggiore controllo e ottimizzazione dell'applicazione rispetto allo scheduling del kernel.





BREVE RECAP

MODELLI DI ESECUZIONE E GESTIONE DEI PROCESSI

- **Concetti Base:** Processi sequenziali eseguiti in parallelo per nascondere gli effetti degli interrupt.
- **Caratteristiche dei Processi:**
 - Creati e terminati dinamicamente.
 - Spazio di indirizzi unico per ogni processo.
- **Thread:**
 - Multipli thread di controllo in un singolo processo.
 - «Schedulati» indipendentemente con stack propri.
 - Condivisione dello stesso spazio di indirizzi.
 - Possono essere implementati nello spazio utente o nel kernel.



SINCRONIZZAZIONE, STATI E SCHEDULING

- **Sincronizzazione e Comunicazione:**

- Uso di semafori, monitor, messaggi.
- Prevengono l'accesso simultaneo a regioni critiche per evitare caos.
- Processi possono essere: in esecuzione, eseguibili o bloccati.
- Cambiamenti di stato causati da primitive di comunicazione.

- **Algoritmi di Scheduling:**

- Varietà studiata: shortest-job-first, round robin, scheduling a priorità, code multilivello, e molti altri.
- Distinzione in alcuni sistemi tra meccanismo di scheduling e policy, permettendo agli utenti di influenzare l'algoritmo di scheduling.

