LEZIONE 6

Scheduling della CPU

**CPU:** La CPU è l’unità più importante del nostro sistema per diversi motivi; in primo luogo è l’unità sulla quale realmente avvengono i calcoli e poi perché essendo l’unità più rapida, occorre trovare per essa una gestione molto intelligente e rapida.

Concetti fondamentali

* **L’obiettivo della multiprogrammazione è avere sempre processi in esecuzione al fine di massimizzare l’utilizzo della CPU (CPU sempre sottopressione).** Bisogna però evitare **l’effetto convoglio;** cioè se si hanno più CPU si devono equilibrare in modo uniforme le code che si vengono a formare, in quanto più corte saranno le code prima finisce di lavorare.
* L’esecuzione del processo consiste in un ciclo d’elaborazione (svolta dalla CPU) e d’attesa del completamento delle operazioni di I/O. **I processi si alternano tra questi due stati;** ciò vuol dire che la CPU può trattare un solo processo alla volta
* L’esecuzione del processo comincia con una sequenza (una “raffica”) di operazioni d’elaborazione svolte dalla CPU (**CPU burst)** seguita da una sequenza di operazioni di I/O (I/O burst), quindi un’altra sequenza di operazioni della CPU, di nuovo una sequenza di operazioni di I/O e così viaImmagine che contiene screenshot

  Descrizione generata automaticamente.

**Diagramma delle sequenze di operazioni della CPU**

**Immagine che contiene schermo, edificio, orologio

Descrizione generata automaticamente**Appena un processo viene mandato in esecuzione, la CPU ha un picco iniziale di elaborazione che poi nel tempo si assesta.

Scheduler della CPU

* **Ogni qualvolta la CPU passa nello stato d’inattività, il SO sceglie per l’esecuzione uno dei processi presenti nella coda dei processi pronti**
* Le decisioni riguardanti lo scheduling della CPU si possono prendere nelle seguenti circostanze:

1. **Un processo passa dallo stato di esecuzione a quello di attesa**
2. Un processo passa dallo stato di esecuzione allo stato pronto
3. Un processo passa dallo stato di attesa allo stato pronto
4. **Un processo termina**

* **I punti 1 e 4 hanno in comune il fatto che a decidere di fermarsi sono loro e non il SO;** quindi si dice che lo schema di scheduling è senza diritto di prelazione **(non preemptive)**
* **I punti 2 e 3 possono essere stoppati in caso arrivi un processo con priorità maggiore di quello attualmente in esecuzione;** quindi sidice che lo schema di scheduling è con diritto di prelazione **(preemptive)**

Dispatcher

* Il dispatcher è il modulo che passa effettivamente il controllo della CPU ai processi scelti dallo scheduler a breve termine. Questa funzione riguarda:
* Il cambio di contesto
* Il passaggio al modo d’utente
* Il salto alla giusta posizione del programma d’utente per riavviarne l’esecuzione
* **Latenza di dispatch:** il tempo richiesto dal dispatcher per fermare un processo e avviare l’esecuzione di un altro.

Criteri di scheduling Ci sono, a seconda di ciò che bisogna fare, dei criteri di scheduling da considerare, tra cui:

* **MAX utilizzo della CPU:** la CPU deve essere più attiva possibile
* **MAX produttività (throughput):** numero di processi completati nell’unità di tempo
* **MIN tempo di completamento:** tempo necessario per eseguire il processo stesso
* **MIN tempo di attesa:** somma degli intervalli d’attesa passati nella coda dei processi pronti
* **MIN tempo di risposta:** tempo che intercorre tra la sottomissione di una richiesta e la prima risposta prodotta (è dato dal tempo necessario per iniziare una risposta, non dal suo tempo d’emissione)

Algoritmi di scheduling

1. SCHEDULING FIRST-COME, FIRST-SERVED (FCFS)

Immagine che contiene screenshot

Descrizione generata automaticamenteSe si cambiano un pò i parametri, dal punto di vista della CPU non cambia niente in quanto code della proprietà commutativa cioè saranno sempre 30 unità di tempo sia se facciano p1 p2 e p3 oppure p3 p1 p2

**Esempio con parametri invertiti**

**Immagine che contiene screenshot

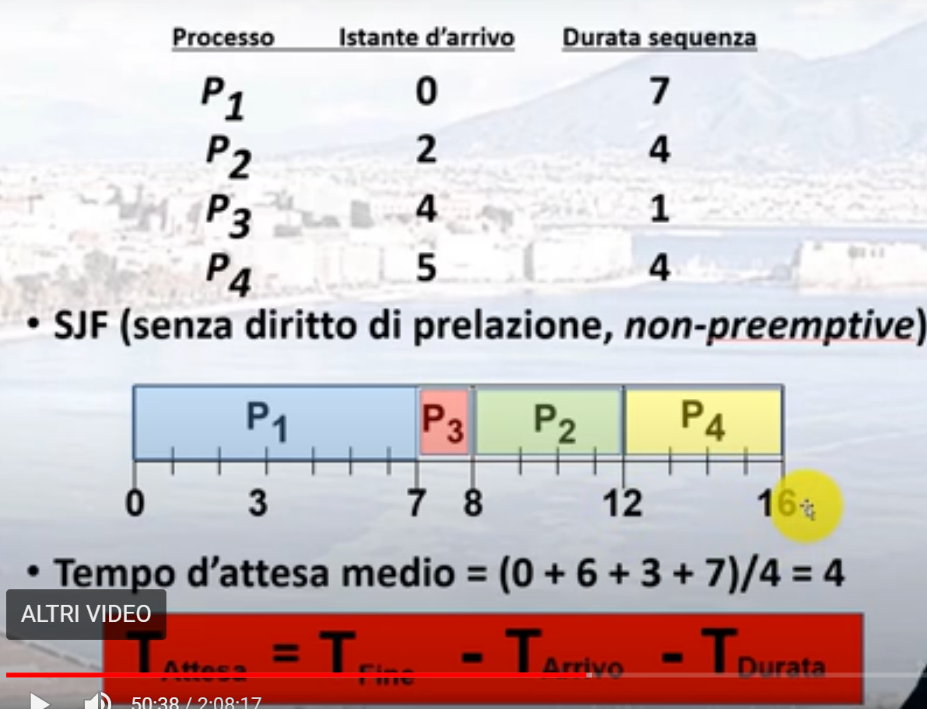
Descrizione generata automaticamente**

1. SCHEDULING SHORTEST-JOB-FIRST (SJF)

* **Associa a ogni processo la lunghezza della successiva sequenza di operazioni della CPU. Quando è disponibile, si assegna la CPU al processo che ha la più breve lunghezza della successiva sequenza di operazioni della CPU.**
* Due schemi:

1. **Senza prelazione:** permette al processo correntemente in esecuzione di portare a termine la propria sequenza di operazioni della CPU.
2. **Con prelazione:** se arriva un nuovo processo con sequenza di CPU inferiore a quella del tempo restante per eseguire il processo in corso, lo sostituisce al processo attualmente in esecuzione (talvolta chiamato shortest-remaining-time-first, SRJF)

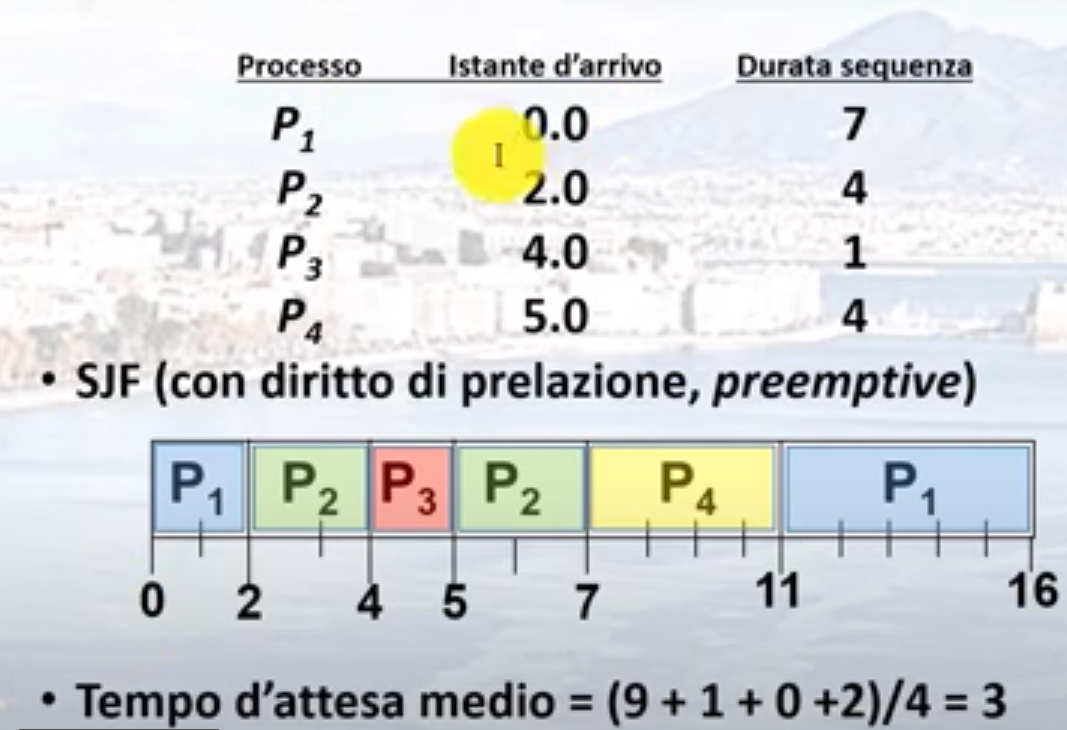
* **L’algoritmo di scheduling SJF è ottimale nel senso che rende minimo il tempo d’attesa medio per un dato insieme di processi.**
* **Lo svantaggio è che non riesce a fare una predizione esatta sui tempi di esecuzione dei processi.**

**Esempio di SJF senza diritto di prelazione**

**Spiegazione**

All’istante 0 ci sta solo P1 che può partire, quindi viene eseguito; finisce all’istante 7, ma nel frattempo sono arrivati i processi P2, P3 e P4. SJF sceglie quello con tempo di esecuzione minore; in wquesto caso quindi partirà dopo P1, P3, poi P2 e in fine P4.

**Esempio con diritto di prelazione**

**Spiegazione**

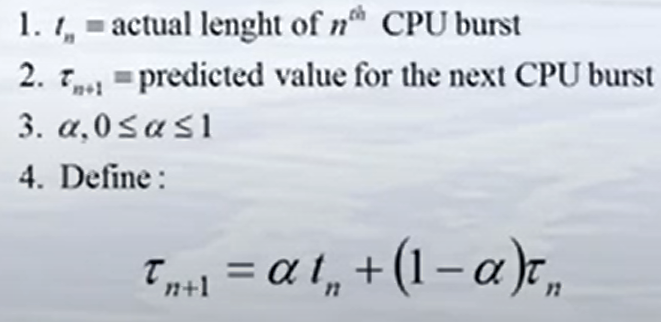
Funziona allo stesso modo del primo esempio solo che ci sono più cambi di contesto in quanto i processi ogni volta che ne arriva uno nuovo vengono prelazionati

* Un altro svantaggio è proprio il fatto che i cambi contesto hanno un costo. Quindi se cominciano a diventare troppi l’algoritmo non risulta essere più efficiente.

**Determinare la lunghezza della successiva sequenza di operazioni della CPU**

* **Cercare di “predire” il suo valore: è probabile, infatti che sia simile ai precedenti**
* Calcolando un valore approssimativo della lunghezza, si può scegliere il processo con la più breve fra tali lunghezze previste.

**Principio di località:** un evento si comporta nell’immediato futuro esattamente come si è comportato nell’immediato passato

****

**Predizione della lunghezza della successiva sequenza di operazioni della CPU**

**Immagine che contiene testo

Descrizione generata automaticamente**

**Esempi di media esponenziale**

**Immagine che contiene uccello

Descrizione generata automaticamente**

Scheduling per priorità

* Si assegna a ciascun processo un numero (intero) di priorità.
* Si assegna la CPU al processo con priorità più elevata (intero più piccolo =priorità più elevata)
* Con diritto di prelazione
* Senza diritto di prelazione
* **L’algoritmo SJF è un caso particolare del più generale algoritmo di scheduling per priorità,** dove la priorità è l’inverso della lunghezza (prevista) della successiva sequenza di operazioni della CPU (a una maggiore lunghezza corrisponde una minore priorità, e viceversa).
* **Problema = Attesa indefinita (starvation):** processi con bassa priorità possono rimanere nell’attesa indefinita della CPU.

**Soluzione = Invecchiamento (again):** aumento graduale della priorità dei processi che attendono nel sistema da parecchio tempo.

1. SCHEDULING CIRCOLARE (ROUND ROBIN, RR)

E’ un meccanismo circolare. Si stabilisce un quanto di tempo e lo si assegna a tutti i processi in maniera circolare

* **Ciascun processo riceve una piccola quantità fissata del tempo della CPU, chiamata QUANTO DI TEMPO (time-quantum), che varia generalmente da 10 a 100 millisecondi, e la coda dei processi pronti è trattata come una coda circolare.**
* Se nella coda dei processi pronti esistono n processi e il quanto di tempo è pari a q, ciascun processo ottiene 1/n-esimo del tempo di elaborazione della CPU in frazioni di, al più, q unità di tempo, allora **ogni processo non deve attendere per più di (n-1)q unità di tempo.**
* **Prestazioni:**
* Se si sceglie un q troppo grande, allora RR tende a FCFS.
* Se si sceglie q troppo breve, si stressa troppo la CPU per i troppi cambi di contesto.

**Esempio di RR con q = 20**

****

* Il RR funziona bene soprattutto con un elevata elaborazione (numero elevato di processi)
* Non troppo ottimale in quanto ci sono moltissimi cambi contesto

Scheduling a code multiple

* Una distinzione è quella che si fa tra i processi che si eseguono:
* **In primo piano** (foreground), o **interattivi**
* **In sottofondo** (nbackground), o **a lotti (batch)**
* Questi 2 tipi di processi possono avere diverse necessità di scheduling:
* in primo piano: RR
* in sottofondo: FCFS
* E’ necessario avere uno scheduling tra le code.
* Scheduling con priorità fissa e prelazione (es. la coda dei processi in primo piano può avere la priorità assoluta sulla coda dei processi in sottofondo). Possibilità di starvation.
* Possibilità di impostare i quanti di tempo (per ogni coda si stabilisce una parte del tempo d’elaborazione della CPU, che si può a sua volta suddividere tra i processi che la costituiscono: 80% alla coda dei processi in primo piano, in RR, 20% a quelli in sottofondo, in FCFS.

Immagine che contiene screenshot, parcheggiato

Descrizione generata automaticamente

Scheduling a code multiple con retroazione

* Lo scheduling a code multiple con retroazione permette ai processi di spostarsi fra le code. In questo modo si attua una forma d’invecchiamento che impedisce il verificarsi di un’attesa indefinita.
* E’ caratterizzato dai seguenti parametri:
* Numero di code
* Algoritmo di scheduling per ciascuna coda
* Metodo usato per determinare quando spostare un processo in una coda con priorità maggiore.
* Metodo usato per determinare quando spostare un processo in una coda con priorità minore.
* Metodo usato per determinare in quale coda si deve spostare un processo quando richiede un servizio.

**Immagine che contiene screenshot, luce, pensile, sedendo

Descrizione generata automaticamenteEsempio di code multiple con retroazione**

**Tre code:**

-Q0 = quanto di tempo di 8 millisecondi

-Q1 = quanto di tempo di 16 millisecondi

-Q2 = FCFS

**Scheduling**

-Un nuovo processo viene assegnato alla coda Q0 secondo il criterio FCFS e ottiene un quanto di 8 millisecondi. Se non termina in quel quanto di tempo, viene spostato alla fine della coda Q1.

-Si assegna un quanto di 16 millisecondi al processo alla testa della coda Q1 ma se questo non riesce a completare la propria esecuzione, viene sottoposto a prelazione e messo nella coda Q2.

Scheduling per sistemi con più unità d’elaborazione

* Lo scheduling della CPU diventa più complesso nei sistemi con più unità d’elaborazione
* **Sistemi omogenei:** sistemi nei quali le unità d’elaborazione sono, in relazione alle loro funzioni, identiche.
* **Condivisione del carico (load sharing)**
* **Multielaborazione asimmetrica** (asymmetric multiprocessing): **tutte le attività del sistema sono gestite da un’unica unità centrale** (master server); le altre unità d’elaborazione eseguono soltanto il codice d’utente.

Scheduling per sistemi d’elaborazione in tempo reale

* **Sistemi in tempo reale stretto:** system real-time hard
* **Sistemi in tempo reale debole:** system real-time soft

**Differenze:** il sistema hard deve necessariamente garantire il completamento delle funzioni critiche entro un tempo definito (oltre sarebbe inutile). In quello soft invece i processi critici hanno una priorità maggiore dei processi ordinari; quindi il SO che ci sta un processo con alta priorità ma non è garantito quando questo interverrà per eseguirli.

**VALUTAZIONE DEGLI ALGORITMI**

* **Modello deterministico:** tipo di valutazione analitica che considera un carico di lavoro predeterminato e definisce le prestazioni di ciascun algoritmo per quel carico di lavoro.
* **Reti di code:** se sono note le distribuzioni degli arrivi e dei servizi si possono calcolare l’utilizzo, la lunghezza media delle code, il tempo medio di attesa.
* **Simulazioni e realizzazione:** si parametrizza un sistema e lo si fa girare; dopo di che si registra tutto (tape trace), dopo si cerca di modificare qualche parametro per poi rimandarlo in esecuzione (con i nuovi parametri). Quindi la valutazione di algoritmi di scheduling serve proprio a cercare valutazioni successive partendo dai parametri inizialmente posti per via statistica. In seguito elaborando sempre di più, si hanno dei feedback che aiutano ad avere un tuning migliore per la simulazione. Ma effettuare un tuning del genere è molto dispendioso, anche se diventa in questo modo sempre più preciso (dato che si rilancia in esecuzione di continuo con parametri man mano sempre più preci)