Nei sistemi multitasking, la schedulazione ha il compito di gestire la turnazione dei processi sul processore in base a delle politiche. Lo scheduler della CPU ordina la lista dei processi pronti, ponendo in prima posizione quello che il dispatcher manderà in esecuzione. È importante che sia efficiente, ma soprattutto veloce poichè viene chiamato molto di frequente e si vuole minimizzare il sovraccarico di gestione. Esistono altri 2 livelli di schedulazione (a lungo e a medio termine), qui non trattati.
L'attivazione dello schedulatore può avvenire:

- Senza preemption: la cpu viene assegnata ad un processi finchè questi non la rilascia esplicitamente o si mette in attesa di un evento/periferica; quindi il controllo torna allo schedulatore. In questo caso l'attivazione è sincrona con l'evoluzione della computazione del processo
- Con preemption: il sistema può anche forzare il rilascio della cpu assegnata a un processo. Se ciò accade, l'attivazione è asincrona con la computazione del processo e se questi condivide risorse con altri, deve usare tecniche di sincronizzazione per evitare inconsistenze.

La scelta di un algoritmo di schedulazione va fatta in base ai criteri che si vogliono ottimizzare. Questi criteri sono:

- Sfruttamento del processore
- Throughput: ossia quanti processi vengono completati in un'unità di tempo.
- Tempo di turnaround: ossia il tempo che passa dal momento in cui un processo entra nel sistema e il momento in cui termina la sua esecuzione. Conta anche il tempo speso nelle varie code di attesa.
- Tempo di attesa: il tempo che un processo passa nella coda dei processi pronti.
- Tempo di risposta: il tempo che passa dal momento in cui viene formulata una richiesta e il momento in cui viene presentato il primo risultato.

Si desidera massimizzare i primi 2 e minimizzare gli ultimi 3. Inoltre, si desidera minimizzarne la varianza, così da rendere il sistema più predicibile.
Per valutare la bontà di un algoritmo di schedulazione nel contesto di applicazione si può procedere in vari modi.
Valutazione analitica:
Un tipo di valutazione analitica è la modellazione deterministica, che definisce in modo matematico le prestazioni di un algoritmo in base a un carico di lavoro prestabilito.
E' un metodo semplice, veloce e preciso, ma i risultati non sono generalizzabili, e nella realtà il carico di lavoro non è costante.
Valutazione statistica:
La valutazione statistica permette di avere dei risultati con un grado di incertezza associato.
Un esempio è la modellazione a reti di code: fa uso di distribuzioni di picchi di CPU e IO, e vede il computer come una rete di servizi, ognuno con una coda associata. Studia quindi i parametri di queste code (lunghezza media, tempi di attesa, …). Questo metodo è piuttosto veloce, ma richiede alcune semplificazioni che potrebbero discostarsi troppo dalla realtà.
Un altro tipo di valutazione statistica è la simulazione: si realizza un modello software del sistema, il più completo possibile, e lo si utilizza per testare il comportamento dell'algoritmo. I dati in input alla simulazione sono picchi di CPU e IO generati casualmente o la traccia di un sistema reale. Sono piuttosto accurate, ma onerose da realizzare.
Implementazione:
Se si vuole maggiore accuratezza, si può procedere con l'implementazione nel sistema reale, utilizzando gli strumenti di rilevazione del sistema per valutare l'efficienza dell'algoritmo.
E' il metodo più accurato, ma richiede tempo per la realizzazione e la cooperazione da parte degli utenti. Inoltre, per ottenere le prestazioni migliori nei vari momenti può essere necessario implementare più algoritmi di scheulazione, e si deve decidere se analizzare i casi singolarmente o studiare i valori medi.
Politiche di schedulazione:
FCFS:
Gestisce la coda dei processi pronti con una politica FIFO: i processi ottengono la CPU nell'ordine con cui

entrano nella coda. E' di tipo non preemptive. È una politica facile da implementare e molto veloce da eseguire, tuttavia il tempo di attesa medio dipende fortemente dall'ordine di arrivo dei processi (effetto convoglio).
Priorità:
Permette di associare ad ogni processo un indice di priorità definito internamente (in base a parametri misurabili), oppure esternamente (dall'utente o dall'amministratore del sistema), quindi ordina la coda in base a questi indici. Può essere realizzata con o senza preemption: nel primo caso quando un processo diventa pronto, richiede schedulazione, e se ha priorità maggiore rispetto a quello in esecuzione, lo sostituisce; nel secondo caso invece la schedulazione viene eseguita quando la cpu viene rilasciata.
Un problema di queste politiche è la possibilità di starvation, ossia il blocco indefinito di processi a priorità bassa poiché ci sono sempre processi a priorità maggiore ad ottenere la CPU. Si può risolvere introducendo l'aging, ossia aumentando la priorità di un processo mentre attende. In questo modo, anche i processi a priorità bassa saranno sicuramente eseguiti, dopodiché la priorità tornerà al valore iniziale.
Un caso particolare è la schedulazione SJF (shortest job first), in cui la priorità è l'inverso della durata del successivo picco di cpu di un processo. Minimizza il tempo di attesa medio, ma è necessario stimare la durata del picco in base alla storia passata poiché non è nota, e la stima potrebbe non essere precisa.
Round robin:
È la politica tipica dei sistemi time sharing e permette di distribuire uniformemente il tempo della cpu tra n processi in modo che ognuno ne ottenga 1/n. E' di tipo preemptive.
Si implementa impostando un timer per generare un interrupt allo scadere di un quanto di tempo prestabilito ogni volta che si assegna la cpu a un processo. Se questi non la rilascia la entro la fine del quanto, subisce preemption, torna in coda e lo schedulatore sceglie un nuovo processo da eseguire. Se la turnazione è abbastanza veloce, permette di creare l'illusione di esecuzione parallela. La velocità di esecuzione dei processi dipende dal numero di processi in coda; il tempo di turnaround dipende dalla durata del quanto di tempo. Il problema è deciderne la durata: più lungo riduce il sovraccarico di gestione ma anche l'illusione di parallelismo fino a degenerare in FCFS; più breve aumenta l'illusione di parallelismo, ma anche il sovraccarico di gestione. Idealmente, si vuole che circa l'80% dei picchi di CPU si esaurisca entro un quanto di tempo.
Coda a più livelli (C+L):
Se è possibile raggruppare i processi per tipologie omogenee, si può spezzare la coda dei processi pronti in più code, una per tipologia, così da poter scegliere l'algoritmo di schedulazione più adatto per ogni tipo di processo. Le code sono poi schedulate con un algoritmo dedicato (ad esempio condividendo il tempo tra le code).
I processi sono assegnati permanentemente ad una coda, tuttavia il loro "comportamento" può variare nel tempo, e quindi anche la coda più adatta. La variante con retroazione (C+LR), permette ai processi di migrare da una coda a un'altra, definendo delle politiche di promozione, di degradazione e di allocazione dei processi. In questo modo si può far si che ogni processo si trovi sempre nella coda più adatta al suo attuale compito.
È l'algoritmo più efficiente ma anche il più complesso.