

Soluzione alle corse critiche	Descrizione	Contro
Interruzioni disabilitate (hardware)	Ogni volta che un processo entra in sezione critica le interruzioni sono disabilitate temporaneamente a livello hardware, per poi essere riabilite quando il processo esce dalla sezione critica.	<ul style="list-style-type: none"> • Potenziale blocco definitivo delle interruzioni • Non adatto a sistemi multiprocessore (agisce su una sola CPU)
Variabili di lock (software)	Una variabile condivisa indica la presenza di processi in sezione critica. I processi modificano questa variabile ogni volta che entrano o escono dalla sezione critica dopo averne effettuato un controllo.	<ul style="list-style-type: none"> • Potenziale arresto definitivo dei processi (nel caso in cui lo scheduler fermi il processo fra il controllo e la modifica della variabile di lock)
Alternanza stretta (turni)	I processi si passano alternativamente e in tempi brevi la possibilità di entrare in sezione critica, tramite una variabile di turno condivisa. Simile alle variabili di lock.	<ul style="list-style-type: none"> • Attesa attiva (controllo continuo della variabile <i>turn</i>) • Processi non in sezione critica possono bloccare altri processi
Peterson (turni + lock)	Combina le due soluzioni precedenti. Il turno è condizionato all'effettiva richiesta da parte dei processi. Un processo che necessita di entrare in sezione critica si appropria del turno e rimane in attesa finché nessun altro processo è in sezione critica. Nel caso di accesso contemporaneo da parte di due processi il primo arrivato entra direttamente in sezione critica, lasciando l'altro in attesa (algoritmo d'esempio valido solo per due processi).	<ul style="list-style-type: none"> • Attesa attiva (ciclo while) • Inversione di priorità
TSL – Test and Set Lock (hardware)	Corrispettivo hardware della soluzione delle variabili di lock. A differenza di questa, con la chiamata di sistema TSL è garantita l'atomicità dell'operazione di controllo e modifica del registro di lock.	<ul style="list-style-type: none"> • Attesa attiva (jump not equal) • Inversione di priorità
Sleep – Wakeup	Elimina il problema dell'attesa attiva bloccando i processi in attesa tramite due chiamate di sistema. La <i>sleep</i> blocca il chiamante (il processo anziché entrare in loop si 'addormenta'), mentre la <i>wakeup</i> 'risveglia' il processo specificato da un parametro. E' possibile, in alternativa, associare le chiamate tramite un parametro che specifichi un comune indirizzo di memoria.	<ul style="list-style-type: none"> • Potenziale arresto definitivo dei processi: un processo può svegliarne un altro che sta per addormentarsi, perdendo il segnale di <i>wakeup</i> (risolvibile con un <i>bit di attesa di wakeup</i> nel caso di due soli processi: se il bit è a 1 il processo che ha 'sonno' rimane sveglio)
Semafori	Estensione della soluzione precedente. A <i>sleep</i> e <i>wakeup</i> vengono sostituite le chiamate <i>down</i> e <i>up</i> , che prima di addormentare o svegliare i processi decrementano o incrementano i semafori corrispondenti (analoghi al <i>bit di attesa di wakeup</i>). I semafori funzionano come dei contatori dei <i>wakeup</i> messi da parte. E' necessaria l'atomicità	<ul style="list-style-type: none"> • Potenziale arresto definitivo dei processi (se la <i>down</i> e la <i>up</i> non sono implementate correttamente si rischia un <i>deadlock</i>)

	dell'operazione di controllo e modifica dei semafori ed eventuale sospensione o ripresa di un processo: questa è in genere garantita da una breve disabilitazione delle interruzioni operata dal sistema operativo all'atto delle chiamate di <i>down</i> e <i>up</i> .	
Monitor	Sincronizzazione e mutua esclusione vengono interamente affidate al compilatore. All'interno del codice è specificato un costrutto <i>monitor</i> , all'interno del quale può operare un solo processo per volta. Per evitare <i>deadlock</i> vengono introdotte delle variabili di condizione, oltre a due operazioni che operano su di esse, <i>wait</i> e <i>signal</i> (analoghe a <i>sleep</i> e <i>wakeup</i>)	<ul style="list-style-type: none"> • Il compilatore utilizzato deve riconoscere i monitor e garantire in qualche modo la mutua esclusione e sincronizzazione
Messaggi	<p>I processi comunicano direttamente tra di loro tramite due chiamate di sistema: <i>send</i> e <i>receive</i>. Queste chiamate fanno solo da vettore tra un processo ed un altro trasportando dei messaggi. I messaggi servono ai processi per comunicarsi lo stato delle cose e decidere così il da farsi.</p> <p>Mittente e destinatario sono identificati da un codice univoco in ogni transazione. La correttezza delle comunicazioni è garantita da un messaggio di feedback chiamato <i>acknowledgement</i>: se questo non viene ricevuto dal mittente entro un certo tempo, il mittente rinvia il messaggio. Un ulteriore perfezionamento di questa soluzione è costituito dall'introduzione di accorgimenti quali:</p> <ul style="list-style-type: none"> ○ Autenticazione (garantisce che ogni messaggio ricevuto venga preso in considerazione solo se il mittente è autorizzato all'invio di quel particolare messaggio) ○ Mailboxes (sono dei buffers - costituiti da un numero costante di messaggi eventualmente vuoti - associati ognuno ad un processo: ricevono i messaggi in entrata consentendo al processo di elaborarli secondo il tempo a disposizione, ottimizzando così le prestazioni) 	