

## CAP 3: Descrizione e controllo dei processi

### Stati del processo:

Un SO deve soddisfare principalmente le seguenti richieste:

- Interrallacciare l'esecuzione di diversi processi per massimizzare l'uso del processore, fornendo un tempo di risposta ragionevole
- Allocare risorse ai processi, usando una specifica politica ed evitando lo stallo
- Supportare la comunicazione tra processi e la loro creazione a livello utente

Un processore deve eseguire sequenzialmente istruzioni macchina, che risiedono nella memoria principale come programmi.

Per PROCESSO (o task) si intende un insieme di istanze di un programma in esecuzione completate in modo sequenziale. Esso è costituito dalle istruzioni e dai dati elaborati. Le prime possono essere elencate in una TRACCIA che caratterizza il comportamento di un singolo task.

Quando si hanno più processi rispetto ai processori, il SO consente l'esecuzione solo per un tot di tempo permettendo l'uso della CPU a tutti quanti. Così servendosi di un allocatore alterna più processi da completare, saltando quelli in attesa di un evento.

Per gestire al meglio il collegamento tra processi viene introdotto il concetto di STATO.

Esistono vari modelli, il più semplice è IL MODELLO A DUE STATI.

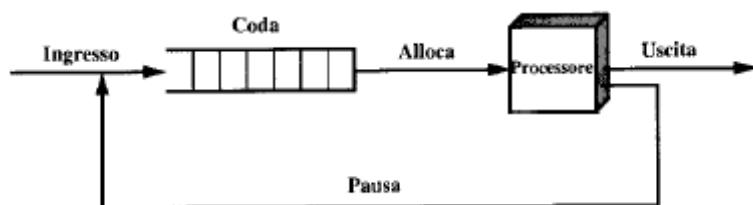
Un processo può essere eseguito dal processore (RUNNING) o no (NOT-RUNNING).



Un processo viene creato e inserito nel gruppo dei processi not-running. Poi viene preso ed eseguito, lo stato passa a running. Esso potrebbe terminare o essere messo in pausa perché o è finito il suo quanto di tempo o è in attesa di un evento I/O.

Già da questo facile schema si può capire che il SO deve conservare traccia del processo, incluso lo stato corrente e la locazione di memoria.

I processi non eseguiti sono memorizzati in qualche coda tramite puntatori oppure blocchi di dati. Essa può funzionare sia come FIFO (first in first out) sia come coda di priorità (il processo con priorità più alta viene selezionato).



La creazione di un processo avviene tramite due azioni fatte dal SO:

1. Costruisce le strutture dati
2. Alloca lo spazio di indirizzamento

Le cause sono principalmente 4:

- In ambiente non interattivo (batch) viene ricevuto un nuovo job batch (programmi dove non è richiesto l'intervento dell'utente)
- Un nuovo utente si collega al sistema
- Per svolgere una funzione per conto di un programma utente affinché non ci sia attesa (es. stampa)
- A seguito di una richiesta di un processo (generazione di processo) per sfruttare il parallelismo

La terminazione dei processi può avvenire per molti più motivi. Essi sono:

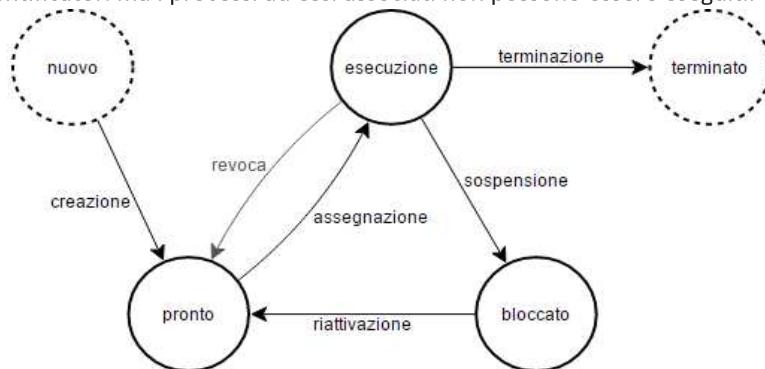
- Terminazione normale
- Superamento del tempo massimo specificato
- Memoria non disponibile
- Violazione dei limiti di memoria

- Errore di protezione
- Tempo scaduto
- Fallimento di un'operazione di I/O
- Istruzione non valida
- Istruzioni privilegiate
- Intervento dell'operatore o del SO
- Terminazione del genitore
- Richiesta del genitore

Il modello appena riportato sarebbe perfetto se i processi not-running fossero sempre disponibili, ma capita che alcuni siano bloccati in quanto in attesa di qualcosa. Perciò potrebbe capitare che prima di trovare un task adatto, l'allocatore debba percorrere tutta la coda. Per questo motivo viene introdotto il MODELLO A CINQUE STATI.

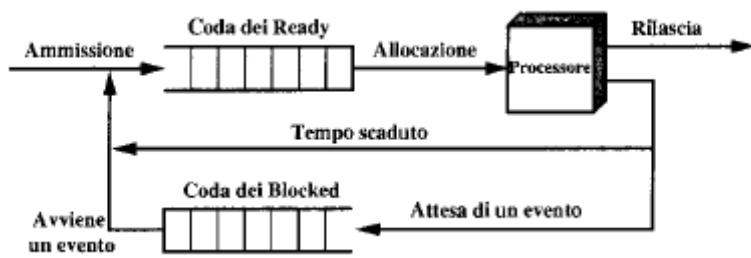
Essi sono: RUNNING (stato di esecuzione), READY (pronto per l'esecuzione), BLOCKED (in attesa di qualche evento), NEW (processo appena creato ma ancora non ammesso dal SO), EXIT (processo rilasciato perché fallito o terminato).

Gli stati new ed exit sono stati aggiunti per non far affaticare la memoria. SO ha costruito o conserva le tabelle e gli identificatori ma i processi ad essi associati non possono essere eseguiti.



Un processo può terminare da qualsiasi stato, non solo quando è running. Ad esempio un processo padre può uccidere il figlio in qualsiasi momento.

In questa nuova struttura avremo due code:

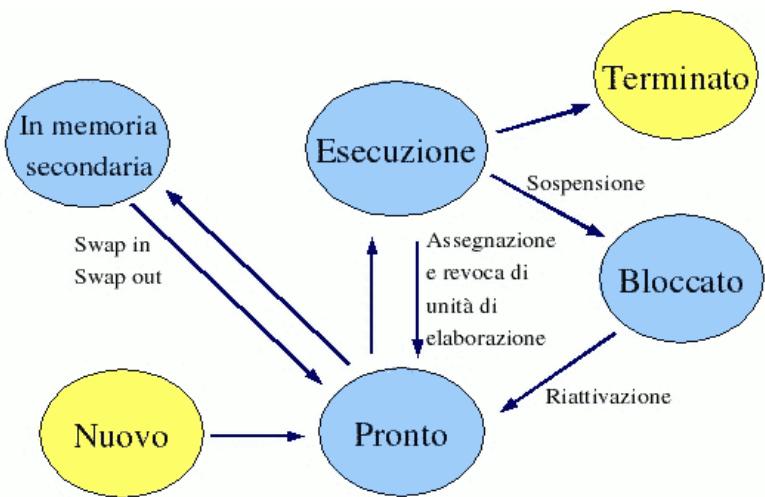


Per migliorare le prestazioni potrebbero essere introdotte più code blocked, una per ogni tipo di evento, così da riportare tutta la lista in ready appena si verifica ciò che si stava attendendo. Oppure, se si usa una politica prioritaria, si potrebbero creare tante code ready quante sono i livelli di priorità. Ciò permetterebbe una scelta del processo molto più veloce.

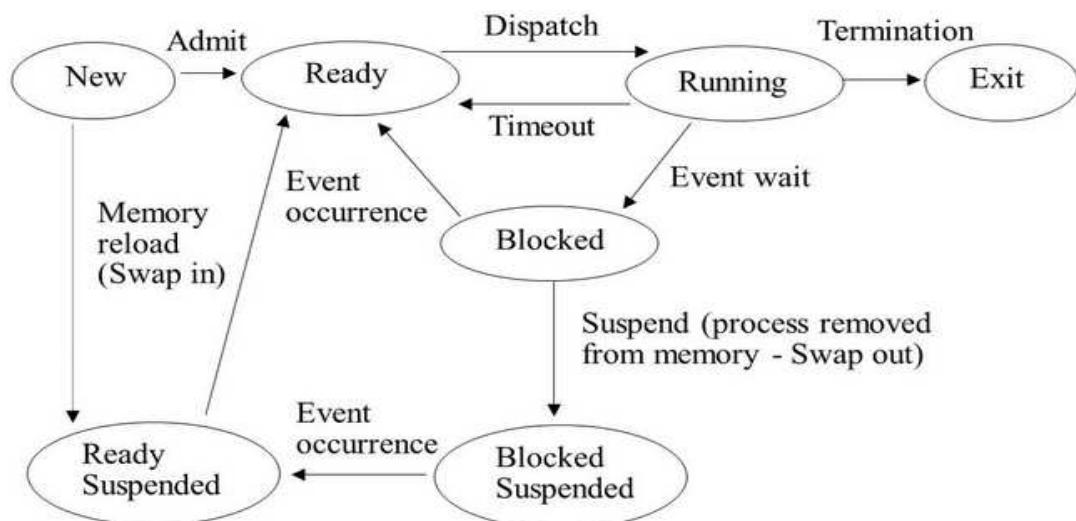
Come sappiamo un processo per essere eseguito, deve essere caricato nella memoria principale. I modelli visti fino ad ora seguono questa linea. Il problema principale di queste strutture è la lentezza degli eventi I/O che potrebbero portare all'inattività del processore, poiché quest'ultimo è ampiamente più veloce. Per evitare ciò si potrebbe aumentare la memoria principale, ma questo comporterebbe un aumento dei costi, in più il peso dei programmi sta crescendo in maniera notevole.

Altra soluzione è lo SWAPPING, cioè spostare un processo, o una sua parte, sul disco, nella coda dei SUSPEND. Ciò permette di liberare memoria e quindi il SO può creare un nuovo processo o riabilitare un programma sospeso.

L'operazione introdotta è sempre di I/O ma essendo del disco è generalmente più rapida e quindi le prestazioni miglioreranno.



Come già detto il SO scarica su disco un processo (SWAP OUT) e dopo sceglie se sostituirlo con uno nuovo o uno sospeso. In quest'ultimo caso bisogna fare attenzione a sceglierne uno che non è più bloccato. Quindi è bene dividere lo stato suspend in due: BLOCKED-SUSPEND e READY-SUSPEND.



Solitamente SO preferisce sospendere un processo bloccato, ma può capitare di dover sospendere uno pronto poiché solo così si riesce a liberare un blocco sufficientemente grande di memoria principale. Inoltre SO può decidere di sospendere un processo pronto a bassa priorità rispetto a uno bloccato con una priorità maggiore, se ritiene che quest'ultimo ritorni disponibile a breve.

Sempre per una preferenza di priorità, il SO può decidere di riabilitare un processo blocked-suspend, che quindi entra nella coda blocked. Naturalmente sempre se ritiene che l'evento atteso sia prossimo ad accadere.

Esistono altre cause che portano alla sospensione dei processi:

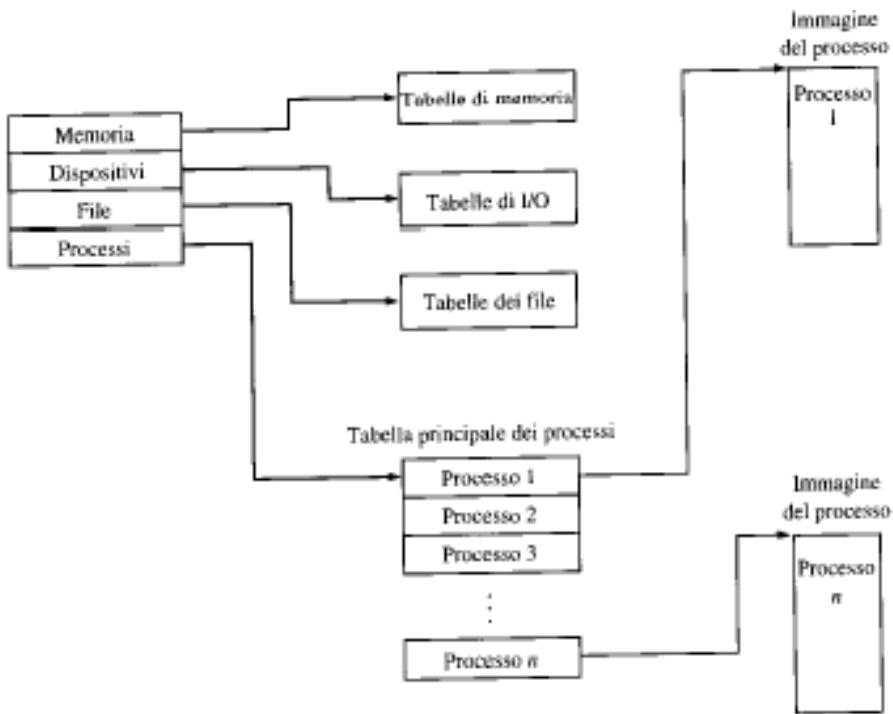
- Sono processi in background o sospettati di causare problemi
- A seguito di una richiesta da parte di un utente, ad esempio per effettuare un debug
- Se è un processo periodico e si trova nel tempo di inattività
- A seguito della richiesta di un processo genitore

#### Descrizione del processo:

Il SO si occupa di controllare gli eventi che avvengono all'interno del sistema di elaborazione, cioè schedula e seleziona i processi per l'esecuzione, alloca le relative risorse e risponde alle richieste dei programmi utente. Quindi è l'entità che gestisce l'uso delle risorse di sistema da parte del processo.

È importante chiedersi "dove si trovano le informazioni di cui necessita il SO per controllare i processi e gestire per conto loro le risorse? Quali sono?"

Le informazioni di cui necessita sono legate allo stato corrente e per averle sempre disponibili costruisce e mantiene delle TABELLE.



La figura riporta una struttura generale delle tabelle di controllo di un SO.

Esse sono 4:

1. Le tabelle di memoria: conservano traccia sia della memoria principale che virtuale. La prima in parte è dedicata al SO, il restante ai processi. Esse conservano:
  - allocazione della memoria principale ai processi
  - allocazione della memoria secondaria (virtuale) ai processi
  - attributi di protezione dei segmenti della memoria principale o virtuale
  - informazioni per gestire la memoria virtuale
2. Le tabelle I/O: servono per gestire i dispositivi I/O e i canali del sistema di elaborazione. Se un'operazione I/O è in esecuzione, il SO deve conoscerne lo stato e la locazione di memoria principale usata come sorgente o destinazione
3. Le tabelle dei file: forniscono informazioni sull'esistenza dei file la loro locazione in memoria secondaria, il loro stato corrente ed altri attributi.
4. Le tabelle dei processi: contengono, direttamente o indirettamente, riferimenti alle risorse considerate sopra (memoria, I/O e file), che devono essere gestiti per conto dei processi stessi.

In figura le tabelle sono riportate come quattro blocchi separati ma esse, in qualche modo, sono collegate.

E' importante notare che esse sono costruite dal SO, quindi ci si potrebbe chiedere in che modo vengono create. Per far ciò è fondamentale che conosca determinate informazioni sul sistema, ad esempio: la grandezza della memoria esistente, il numero di dispositivi I/O, i loro identificatori... Questa conoscenza avviene tramite la configurazione, cioè i dati necessari vengono determinati al di fuori del SO e inseriti con l'intervento di un operatore umano.

Le informazioni che servono sono:

Locazione dei processi un processo contiene uno o più programmi che devono essere eseguiti. A tali programmi viene associata l'**IMMAGINE DEL PROCESSO** composta da:

1. un insieme di locazioni (dove si trovano i dati, le variabili e le costanti)
2. il programma utente (quello da eseguire)
3. una stack (dove viene conservata la traccia delle chiamate di procedura e i parametri passati tra di esse)
4. gli attributi per il controllo (questa collezione è chiamata **PROCESS CONTROL BLOCK**).

La locazione dell'immagine del processo dipende dallo schema della gestione della memoria. Nel caso più semplice essa sarà mantenuta in un blocco contiguo in memoria secondaria (disco). Però, per la gestione, una piccola porzione, comunque, deve essere mantenuta in quella principale, invece per l'esecuzione, deve essere caricata tutta. Quindi entriamo nel caso più complesso; quando un processo è scaricato sul disco, parte dell'immagine viene mantenuta nella memoria principale, perciò il SO deve ricordare quale porzione sta lì e quale manca. I SO moderni gestiscono l'immagine come un insieme di blocchi non necessariamente contigui. Se questi hanno dimensione variabile, si chiamano **SEGMENTI**, invece se fissa, sono definite **PAGINE**. Esiste anche una variante mista. In questo modo è più facile la partizione tra le memorie, ma le tabelle devono riportare le locazioni di ciascun segmento e/o pagina di ciascuna immagine di processo.

Nella figura di sopra, si può notare la presenza della tabella principale dei processi che racchiude i puntatori alle immagini di ogni processo. Qui viene anche precisato se l'immagine che punta è contenuta in blocchi multipli.

Attributi dei processi (Process Control Block) esse sono divise in tre categorie generali:

1. IDENTIFICAZIONE DEL PROCESSO: comprende l'identificatore numerico univoco del processo. Esso può essere l'indice della tabella primaria a cui corrisponde, oppure, tramite un'opportuna mappatura, localizza le tabelle appropriate.  
Gli identificatori sono necessari durante la comunicazione tra processi. Essi possono indicare anche il genitore che li ha creati e suoi, eventuali, discendenti. Può essere aggiunto anche l'identificatore dell'utente responsabile di quel job.
2. INFORMAZIONI SULLO STATO DEL PROCESSORE: comprendono i contenuti dei registri del processore quando il processo corrispondente non è in esecuzione. Sono di questa categoria sia quelli visibili all'utente tramite linguaggio macchina, sia quelli del controllo e di stato, sia i puntatori allo stack.
3. INFORMAZIONI DI CONTROLLO DEL PROCESSO: raggruppa delle nozioni supplementari e sono:
  - Schedulazione e informazioni di stato: servono al SO per schedulare i processi (Stato, priorità, informazioni legate alla schedulazione, evento)
  - Strutturazione dei dati: puntatori ad altri processi, con cui condividono, ad esempio, la coda di priorità o una relazione generazionale
  - Comunicazione tra processi: possono essere presenti flag, segnali e messaggi legati al dialogo tra processi
  - Privilegi: istruzioni o zone di memoria a cui può accedere in maniera privilegiata
  - Gestione della memoria: puntatori a segmenti e/o pagine che descrivono la memoria virtuale di quel processo
  - Proprietà ed utilizzo delle risorse: memorizza le risorse controllate dai processi (es. file aperti)

Il Process Control Block è la struttura più importante del SO, infatti, l'insieme di tutti i blocchi rappresenta lo stato del SO stesso. I PCBs, però, possono essere letti e/o modificati da tutte le routine che entrano in gioco (es. la schedulazione, l'allocazione delle risorse...). L'accesso diretto a queste tabelle non è difficile, poiché si sfrutta l'identificatore del processo che diventa indice della tabella dei puntatori dei PCBs. Il vero punto è critico la protezione di queste strutture. Infatti potrebbero nascere due problemi:

- o Un baco in una routine potrebbe danneggiare i PCBs distruggendo la capacità di gestione del SO
- o Un cambiamento della semantica o struttura dei blocchi potrebbe riguardare diversi moduli del sistema

Questi potrebbero essere risolti facendo passare le varie routine attraverso una routine di gestione. Essa ha il compito di proteggere, assecondare e negare la lettura o la modifica dei PCBs. Ciò, però, richiede un compromesso tra prestazioni globali e grado di affidabilità e correttezza del SW di sistema.

#### Controllo del Processo:

Prima di tutto è importante ricordare i due modi di esecuzione:

1. MODALITA' UTENTE
2. MODALITA' KERNEL (o DI SISTEMA) funzioni tipiche:
  - Gestione dei processi (creazione e terminazione, schedulazione ed allocazione, cambio, sincronizzazione e supporto per la comunicazione, gestione dei process control block)
  - Gestione della memoria (allocazione di uno spazio di indirizzi ai processi, swap, gestione della paginazione e segmentazione)
  - Gestione dell'I/O (gestione dei buffer, allocazione dei canali di I/O e dei dispositivi per i processi)
  - Funzioni di supporto (gestione delle interruzioni, contabilità, supporto)

Esistono questi due tipi di esecuzione per proteggere il SO e le sue tabelle dalle interferenze dei programmi utente.

Il processore per capire quale è il modo corrente va a leggere un bit che si trova nel PROGRAM STATUS WORD (PSW). Esso è diverso per ogni architettura e contiene informazioni sullo stato dei programmi in esecuzione. Il bit viene cambiato in seguito a determinati eventi, ad esempio a seguito di una chiamata ad un servizio del SO in cui bisogna entrare in modalità Kernel.

Adesso vediamo in dettaglio la creazione di un processo. Il SO compie 5 passi:

1. Assegnazione dell'identificatore unico ed aggiunta di un entry nella tabella dei processi
2. Allocazione dello spazio necessario, che comprende tutti gli elementi dell'immagine. I valori dello spazio necessario per gli indirizzi privati e lo stack sono, solitamente, determinati per difetto a meno di richieste da parte dell'utente o del genitore. Qui vengono creati dei collegamenti opportuni nel caso debba entrare in uno spazio condiviso, già creato, di memoria.

3. Inizializzazione del process control block:
  - IDENTIFICAZIONE: contiene ID del processo, più altri, ad esempio quello del genitore
  - INFORMAZIONI SULLO STATO: eccetto il program counter e lo stack di sistema, il resto dei campi sono inizializzati a 0.
  - INFORMAZIONI DI CONTROLLO: sono settati su valori standard per difetto e sugli attributi richiesti per il processo. La priorità è impostata al valore più basso, a meno di richieste specifiche. In molti casi il nuovo task non ha risorse, salvo che non le abbia ereditate dal genitore.
4. Creazione di collegamenti appropriati, ad esempio con altri processi nello stesso stato o con la stessa priorità
5. Estensione o creazione di eventuali nuove strutture dati

Per quanto riguarda il cambio dei processi (switch) si possono osservare delle problematiche progettuali:

- a. Riconoscere gli eventi che provocano uno switch
  - b. Definire la differenza tra cambio di processo e di modalità di esecuzione
  - c. Determinare le operazioni da compiere sulle strutture dati per effettuare lo switch
- a. Un cambio di processo può verificarsi quando il SO ottiene il controllo del processo in esecuzione. I meccanismi che possono provocare ciò sono:
- Interruzione: le cause sono eventi esterni al processo. Il controllo prima passa ad un gestore dell'interruzione, così da poter effettuare delle operazioni di manutenzione, poi salta alla routine del SO competente.
  - Trap: è legato ad eccezioni ed errori all'interno del processo corrente. Quando si verificano il SO deve stabilire se sono fatali o no. Nel primo caso il processo viene portato nello stato exit e avviene il cambio. Nell'altro caso, l'azione dipende dalla scelta di progettazione: il SO potrebbe tentare una procedura di recupero, o semplicemente notificare l'errore all'utente, effettuare un cambio o riprendere l'esecuzione di quello corrente
  - Chiamata al supervisore: il controllo viene trasferito al sistema sotto richiesta del processo utente che entra in stato blocked.
- b. La differenza tra cambio di modo e di processo è che il primo caso può avvenire senza che ci sia un cambio di stato e ciò comporta un minor salvataggio delle informazioni

Le operazioni che vengono effettuate in un cambio di modo di esecuzione:

- salvataggio del contesto del programma in esecuzione, che comprende le informazioni sullo stato, contenute nel process control block, che potrebbero essere alterate e che permettono il ripristino del processo
  - impostazione del program counter all'indirizzo di partenza del programma che si occupa della gestione dell'interruzione
  - passaggio da modalità utente a kernel, così da eseguire le istruzioni privilegiate della gestione di interruzione
- Solitamente, a seguito di un'interruzione, il SO mantiene il processo in esecuzione quindi non sono necessarie altre operazioni. Quindi, dopo che il gestore ha finito, è sufficiente ripristinare tutto.

c. Nel caso, invece, che ci sia un cambio di processo, i passi da seguire sono più numerosi:

- salvataggio del contesto del processore, tra cui i registri e il program counter
- aggiornamento del process control block del processo in stato running -> cambio di stato e inserimento del motivo
- spostamento del process control block nella coda appropriata (ready, ready-suspend, Blocked)
- selezione di un nuovo processo
- aggiornamento del process control block del task prescelto
- aggiornamento delle strutture dati per la gestione della memoria
- ripristino del contesto del processore

Per ultimo poniamo l'attenzione sul SISTEMA OPERATIVO. Esso è uno speciale programma che viene eseguito sul processore. Cede frequentemente il controllo e, per riprenderlo, bisogna aspettare il processore.

In fin dei conti il SO è una collezione di programmi eseguiti dal processore, quindi si può definire processo?

I progettisti hanno fornito varie risposte, di seguito sono riportati i tre diversi approcci più famosi:

1. KERNEL SEPARATO DAI PROCESSI -> l'esecuzione del kernel avviene al di fuori di qualsiasi processo. Quando vi è un cambio di modalità, il SO ha la sua regione di memoria e il suo stack per controllare le chiamate ed i ritorni da procedura. Alla fine può ripristinare il processo chiamante o terminare le operazioni di salvataggio schedulandone uno nuovo. È ben delineata la soglia tra programma utente e codice del SO eseguito in modalità privilegiata.
2. FUNZIONI DEL SO ESEGUITE ALL'INTERNO DEI PROCESSI UTENTE -> questo approccio, solitamente, è usato in macchine piccole. Consiste nell'eseguire il SW del SO nel contesto di un processo utente. Il SO è visto come una collezione di routine che l'utente chiama per effettuare diverse funzioni eseguite nell'ambiente utente. Il kernel ha, comunque, uno stack separato necessario per gestire le chiamate e i ritorni. Però i dati e il codice del SO si trovano nello spazio degli indirizzi condivisi con tutti i processi. Quando si verifica un evento che provoca un cambio di modalità, avviene il salvataggio del contesto e l'intervento di una routine del SO che

viene eseguito dentro il processo corrente. Per ritornare alla modalità utente, non è necessario un cambio di processo (poiché già siamo in quello giusto) ma un semplice ripristino! Quando, invece, il processo deve entrare in non-esecuzione, e deve essere schedulato uno nuovo, entra in gioco la funzione dello switch, che gira al di fuori di tutti i processi. Questo approccio sottolinea la relazione tra processo e programma, infatti non è 1 a 1, ma entro un processo possono essere eseguiti sia programmi utente che di SO.

3. SO BASATI SUI PROCESSI -> il SO viene implementato come un insieme di processi di sistema. Il sw, che è parte del kernel, è eseguito in modalità kernel. Le sue funzioni sono organizzate come processi separati. I vantaggi di questo approccio sono:
  - impone una progettazione dei programmi basta sull'uso di un SO modulare
  - alcune funzioni non critiche sono implementate come processi separati. Quindi possono essere interallacciate ad altri processi, ed avere un livello di priorità
  - è utile in un ambiente multi-core, in cui alcuni servizi del SO possono essere affidati ad un processore dedicato, migliorando le prestazioni.