Sistemi Operativi

AAF - Secondo anno - 3CFU

A.A. 2021/2022

Corso di Laurea in Matematica

I Processi

Annalisa Massini

Dipartimento di Informatica
Sapienza Università di Roma

Argomenti trattati

Che cos'è un processo

Come i processi sono rappresentati e controllati dal SO

Requisiti di un SO

- Compito fondamentale dei SO è la **gestione dei processi**
  - ovvero la gestione delle diverse computazioni che si vogliono eseguire in un sistema computerizzato
- Il sistema operativo deve:
  - permettere l'esecuzione alternata (*interleaving*) di processi multipli
  - assegnare le risorse ai processi e proteggere dagli altri processi le risorse assegnate ad un processo
  - permettere ai processi di scambiarsi informazioni
  - permettere la sincronizzazione tra processi

Concetti preliminari

Riassumiamo quello che abbiamo già visto:

- I computer sono composti da un insieme di risorse hardware
- Le applicazioni per computer sono sviluppate per compiere un qualche compito (ricevono un input dall'esterno, compiono un'elaborazione, producono un output)
- Non è efficiente scrivere direttamente applicazioni per una particolare architettura hardware
- Il SO fornisce un'interfaccia comune per le applicazioni (è lo strato tra le applicazioni utente e l'hardware)
- Il SO fornisce una rappresentazione delle risorse che può essere consultata su richiesta dalle applicazioni

Concetti preliminari

La gestione delle applicazioni da parte del SO si basa sulle seguenti osservazioni:

- Le risorse sono rese disponibili a più applicazioni contemporaneamente
- L'uso del processore viene concesso alternativamente a diverse applicazioni, ora ad una, ora ad un'altra
- È necessario un uso efficiente del processore e dei dispositivi di input/output

Che cos'è un *processo*?

Abbiamo già visto le possibili definizioni di **processo**

- Un programma in esecuzione
- Un'istanza di un programma in esecuzione su un computer
- L'entità che può essere assegnata ad un processore per l'esecuzione
- Un'unità di attività caratterizzata dall'esecuzione di una sequenza di istruzioni, da uno stato corrente e da un insieme associato di risorse

Processi, esecuzione e programmi

- Un processo è composto da:
  - codice (anche condiviso): le istruzioni da eseguire
  - un insieme di dati
  - un numero di attributi che descrivono lo stato del processo
- Per adesso, *processo in esecuzione* vuol dire che *un utente ha richiesto l'esecuzione di un programma, che ancora non è terminato*
- Vedremo che questo non significa necessariamente che il processo sia in esecuzione su un processore
  - o meglio, non è detto che, fissato un istante tra la richiesta della sua esecuzione e la sua terminazione, esso sia in esecuzione su un processore
  - decidere se mandare in esecuzione un processo su un processore è uno dei compiti fondamentali di un sistema operativo

Processi, Esecuzioni e Programmi

- Dietro ogni processo c'è un *programma*
  - nei sistemi operativi moderni, è solitamente memorizzato su una memoria di massa, ad esempio un disco rigido
  - possono far eccezione i processi creati dal sistema operativo stesso
  - solo eseguendo un programma si può creare un processo
  - eseguendo un programma si crea *almeno* un processo

Processi, esecuzioni e programmi

- Un processo ha 3 macrofasi: **creazione**, **esecuzione** e **terminazione**
- La terminazione può essere:
  - *prevista*
    - es1: un programma deve leggere numeri, ordinarli e scrivere l'output riordinato; alla fine dell'ultimo passo, il processo è terminato
    - es2: word processor (basato su eventi); se l'utente clicca sulla X della finestra, il processo è terminato, **ma** se non lo chiude esplicitamente, potrebbe rimanere in esecuzione per sempre
  - *non prevista*
    - ad esempio, il processo esegue un'operazione non consentita: viene attivato automaticamente un interrupt che può portare alla chiusura forzata del processo da parte del SO
    - ad esempio, il programma che ordina i numeri cerca di leggere della memoria non assegnata a lui, cioè dichiara un array da 100 numeri e cerca di leggere il 101-esimo

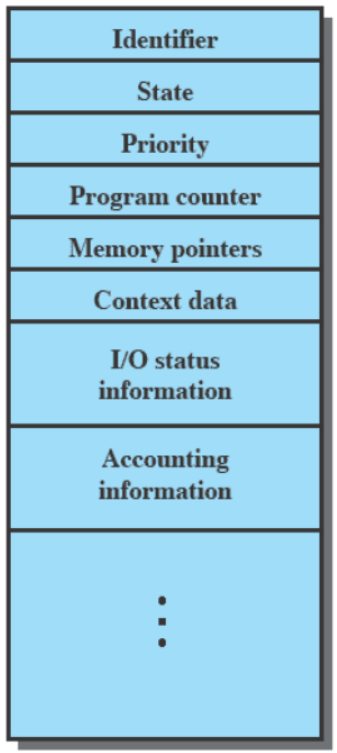
Elementi di un processo

- Finché il processo è in esecuzione, ad esso sono associati un certo insieme di informazioni, tra le quali:
  - identificatore
  - stato (*running*, ma non solo...)
  - priorità
  - *hardware context*: valore corrente dei registri della CPU
    - include il program counter e il contenuto dei registri
  - puntatori alla memoria (che definiscono l'*immagine* del processo)
  - informazioni sullo stato dell'input/output
  - informazioni di accounting (quale utente lo esegue)

Process Control Block

Tali informazioni sono raccolte in una struttura dati, il **Process Control Block**

- È creata e gestita dal sistema operativo
- Contiene gli elementi del processo
- Permette al SO di gestire più processi contemporaneamente
- Contiene sufficienti informazioni per bloccare un programma in esecuzione e farlo riprendere successivamente dallo stesso punto in cui si trovava



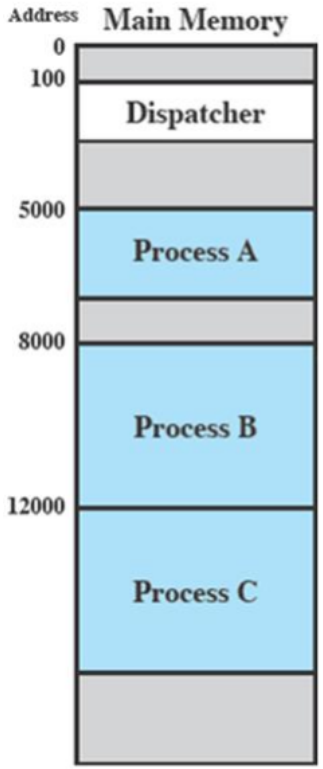
Traccia di un processo

- Quando un programma deve essere eseguito, per esso viene creato un processo
- Il comportamento di un particolare processo è caratterizzato dalla sequenza delle istruzioni che vengono eseguite
- Questa sequenza è detta **Trace** (**traccia**)
- Il **dispatcher** è un piccolo programma che sospende un processo per farne andare in esecuzione un altro assegnandogli l'uso del processore

Esecuzione di un processo

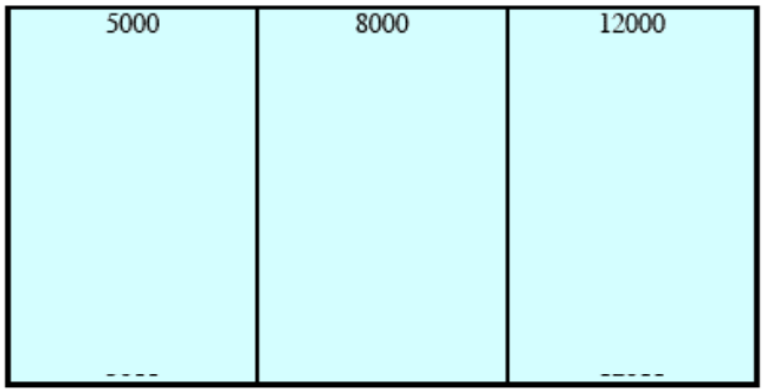
Esempio

- Consideriamo 3 processi in esecuzione
- Sono tutti in memoria (più il dispatcher)
- Ignoriamo per ora la memoria virtuale



La traccia dal punto di vista del processo

Ogni processo viene eseguito senza interruzioni fino al termine



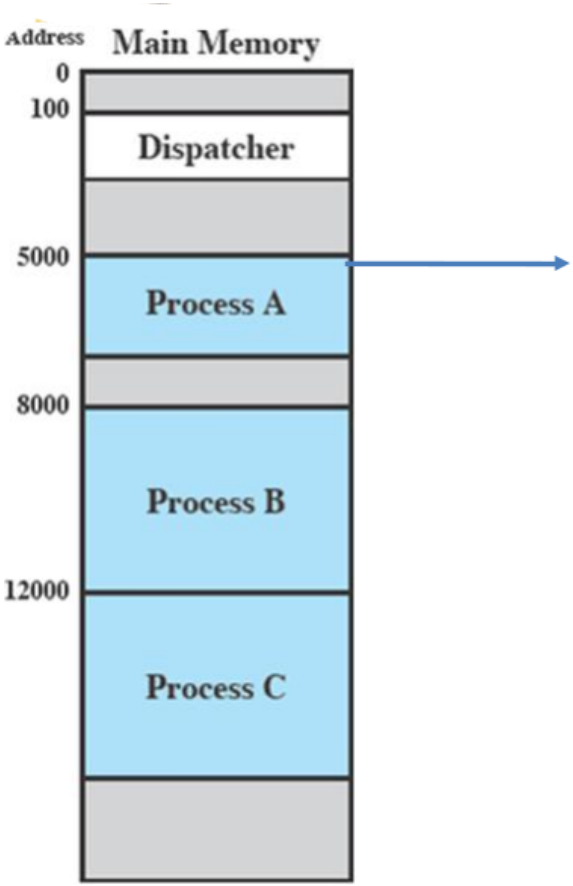
(a) Trace of Process A (b) Trace of Process B (c) Trace of Process C

La traccia dal punto di vista del processo

Ogni processo viene eseguito senza interruzioni fino al termine

(a) Trace of Process A	(b) Trace of Process B	(c) Trace of Process C
5000	8000	12000
5001	8001	12001
5002	8002	12002
5003	8003	12003
5004		12004
5005		12005
5006		12006
5007		12007
5008		12008
5009		12009
5010		12010
5011		12011

La traccia dal punto di vista del processore



1	5000
2	5001
3	5002
4	5003
5	5004
6	5005
---------- Timeout	
7	100
8	101
9	102
10	103
11	104
12	105
13	8000
14	8001
15	8002
16	8003
---------- I/O Request	
17	100
18	101
19	102
20	103
21	104
22	105
23	12000
24	12001
25	12002
26	12003
27	12004
28	12005
---------- Timeout	
29	100
30	101
31	102
32	103
33	104
34	105
35	5006
36	5007
37	5008
38	5009
39	5010
40	5011
---------- Timeout	
41	100
42	101
43	102
44	103
45	104
46	105
47	12006
48	12007
49	12008
50	12009
51	12010
52	12011
---------- Timeout	

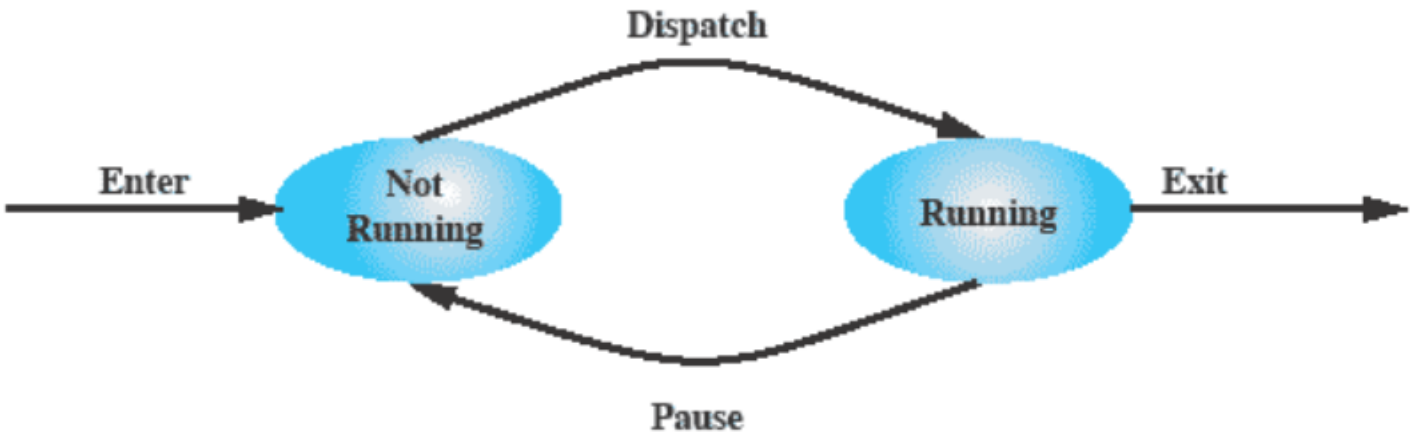
100 = Starting address of dispatcher program

Stati di un processo

Processi a due stati

Modello dei processi a 2 Stati

- Un processo potrebbe essere in uno di due stati
  - In esecuzione - **running** (*sta girando*)
  - Non in esecuzione - ma è comunque un processo **attivo**



- È il modello più semplice possibile
- Come vedremo *troppo semplice*

Diagramma a Coda

Il lavoro del dispatcher può essere descritto in termini di coda

I processi *not running* vengono tenuti in una coda in attesa di diventare *running*

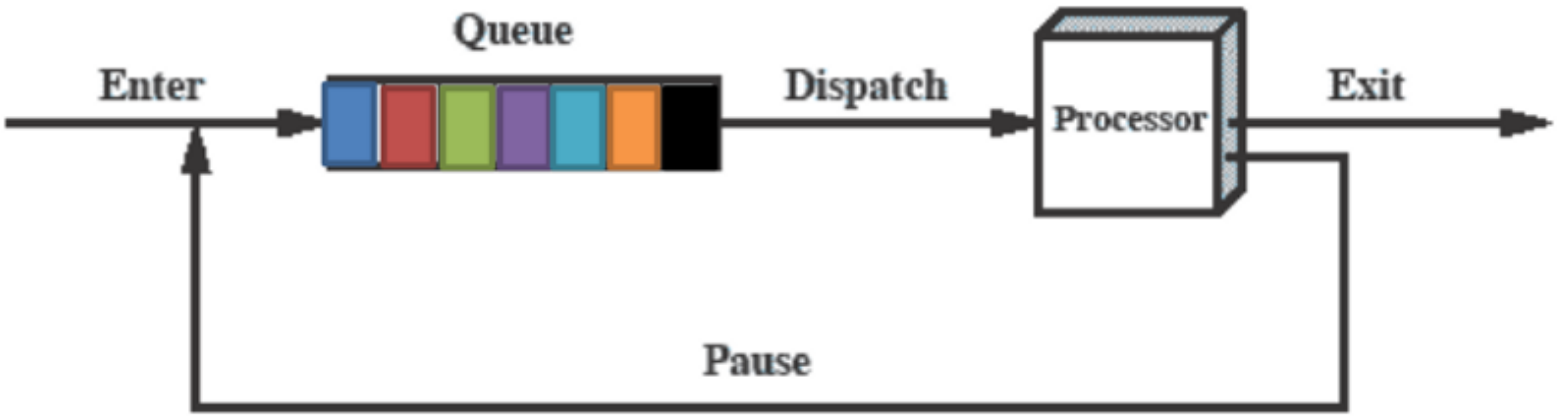
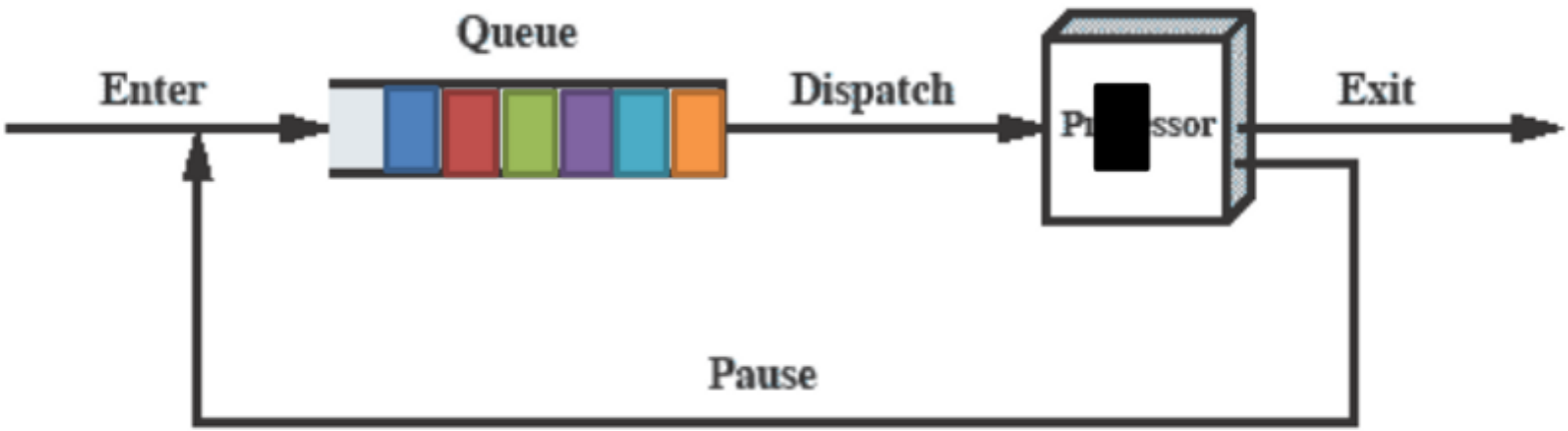


Diagramma a Coda

I processi vengono mossi dal dispatcher del SO dalla CPU alla coda e viceversa, finché un processo non viene completato



Stati di un processo

Creazione e terminazione di processi

Creazione di processi

La vita di un processo è delimitata dalla sua creazione e dalla sua terminazione

- Per creare un processo il SO
  - crea le strutture dati usate per la gestione del processo
  - alloca uno spazio indirizzi nella memoria principale per il processo
- Il processo che viene generato viene aggiunto ai processi già esistenti e si passa da $n \geq 1$ processi ad $n + 1$
- Come detto, un processo viene creato dal SO per fornire un servizio

Creazione di processi

- Eventi che portano alla creazione di processi:
  - ambiente batch (non interattivo)
    - il processo è creato quando viene sottomesso un job
  - ambiente interattivo
    - il processo è creato quando un utente esegue logon
  - il SO può creare un processo da parte di un'applicazione
    - per esempio, se un utente vuole stampare un file, il SO crea un processo per la gestione della stampa
  - un processo può essere generato da un altro processo - **process spawning**
    - per esempio, un processo genera un processo che riceve e organizza i dati generati
    - il nuovo processo - *processo figlio* - gira in parallelo con il processo originale - *processo padre*

Terminazione di processi

- Il sistema deve poter indicare quando un processo deve terminare
- Dopo una terminazione si passa da $n \geq 2$ processi ad $n - 1$
- Resta sempre un processo *master* che non può essere terminato, a meno di non spegnere il computer
- Il SO può gestire direttamente la terminazione o essere avvisato (tramite interruzione) che il processo è terminato

Terminazione di processi

- Normale Completamento
  - il processo esegue una chiamata al servizio del SO per la terminazione
    - ad esempio un'istruzione macchina HALT genera un'interruzione per il SO
- Uccisioni:
  - da parte del SO, per errori come:
    - memoria non disponibile
    - errore di protezione
    - errore fatale a livello di istruzione (divisione per zero...)
    - operazione di I/O fallita
  - da parte dell'utente (es.: X sulla finestra)
  - da parte del processo creatore

Stati di un processo

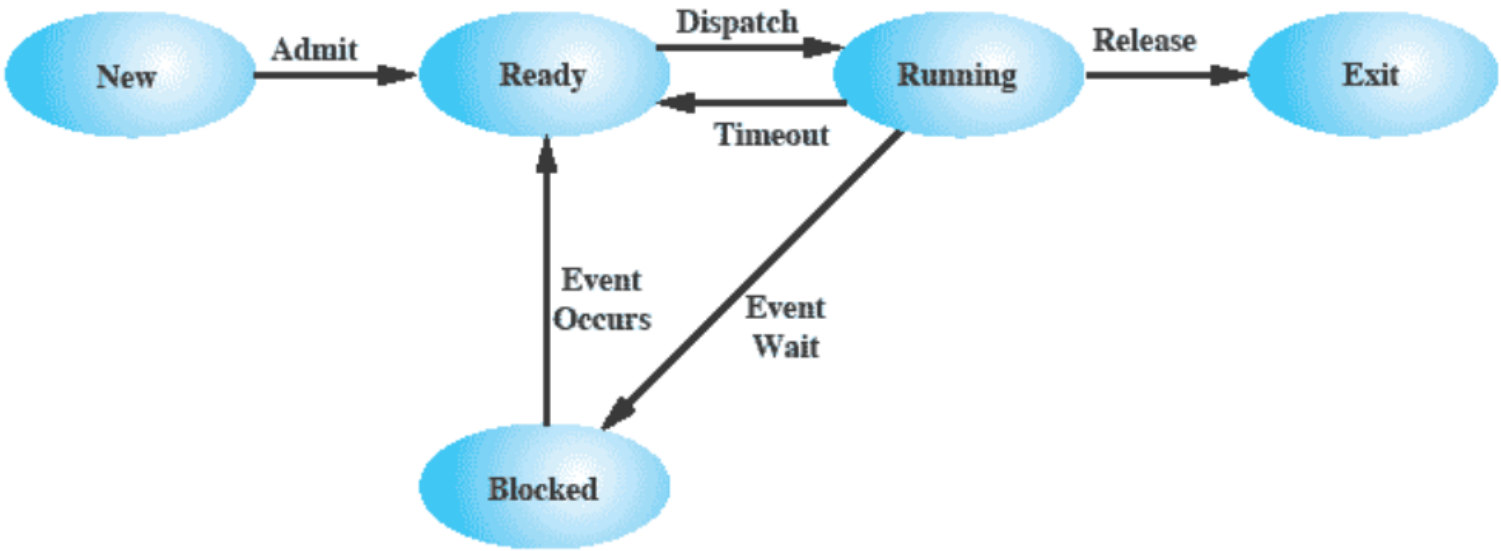
Processi a cinque stati

Modello dei processi a 5 Stati

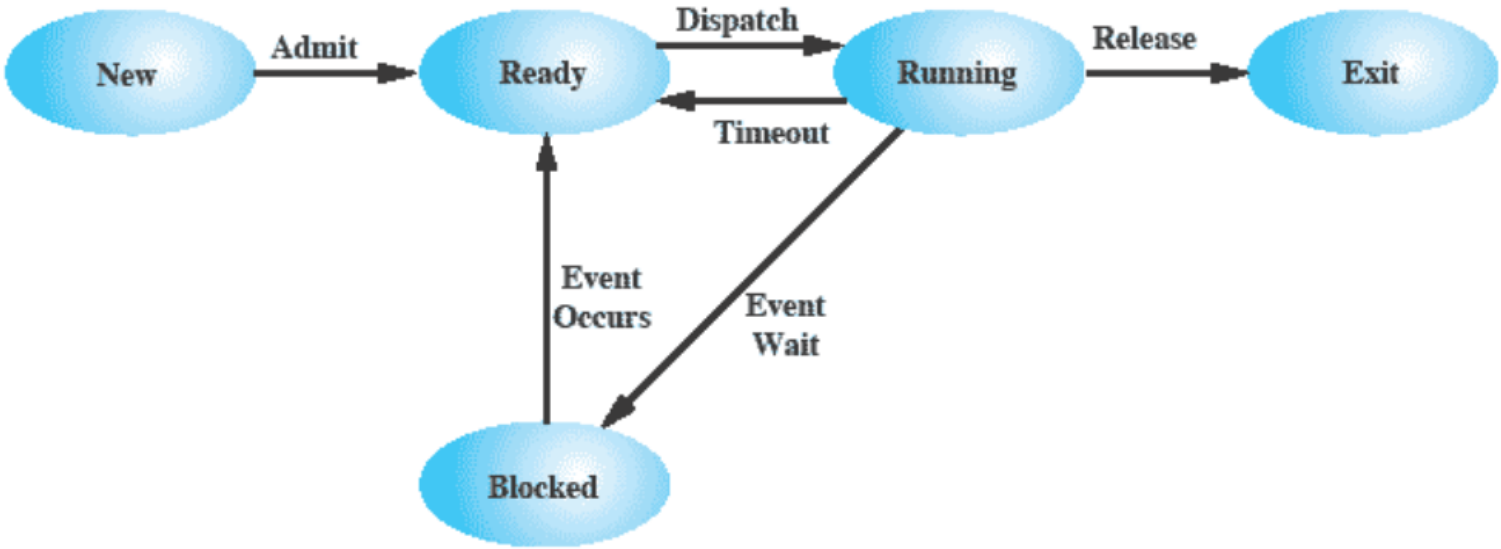
- Torniamo al modello con la gestione a coda
- La coda è del tipo *first-in-first-out*
- Il processore gestisce i processi in coda in *round-robin*, cioè ogni processo in coda riceve una certa quantità di tempo, poi torna in coda
- Il problema è che tra i processi Not Running ci sono:
  - processi *ready to execute*
  - processi *blocked* (in attesa di operazioni di I/O)
- Il dispatcher deve quindi cercare il processo non bloccato che si trova da più tempo in coda

Modello dei processi a 5 Stati

- Usando una singola coda, il *dispatcher* non riesce efficientemente a individuare il processo più vecchio in coda
- Un modo semplice per risolvere il problema è suddividere lo stato *Not Running* in **Ready** e **Blocked**

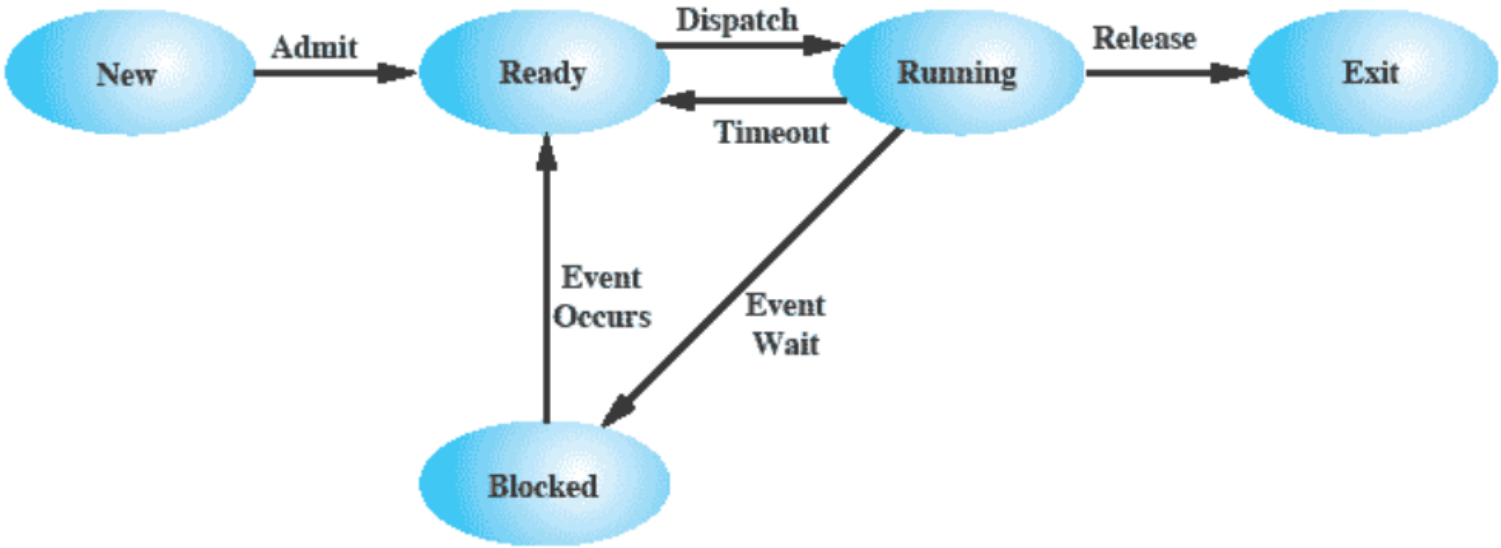


Modello dei Processi a 5 Stati



- *Waiting* è un termine spesso usato in alternativa a *blocked*
- Si può andare anche da ready o blocked ad exit (un processo ne *killa* un altro)

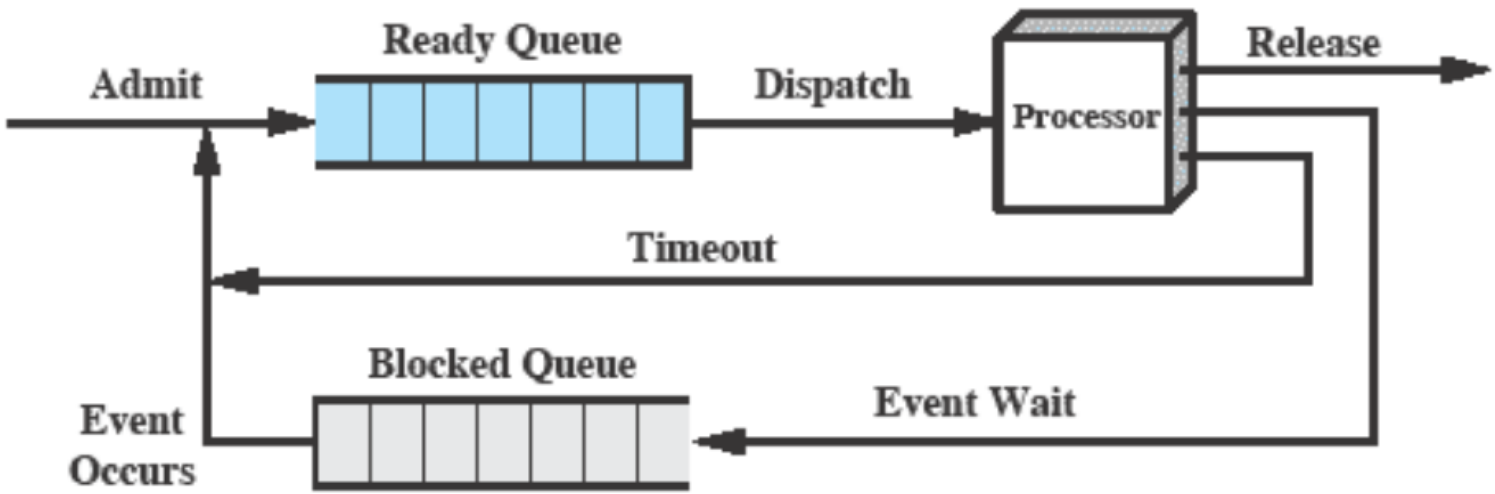
Modello dei Processi a 5 Stati



- Un processo appena creato, viene messo nella coda **Ready**
- Il SO sceglie il processo da far girare dalla coda *Ready*

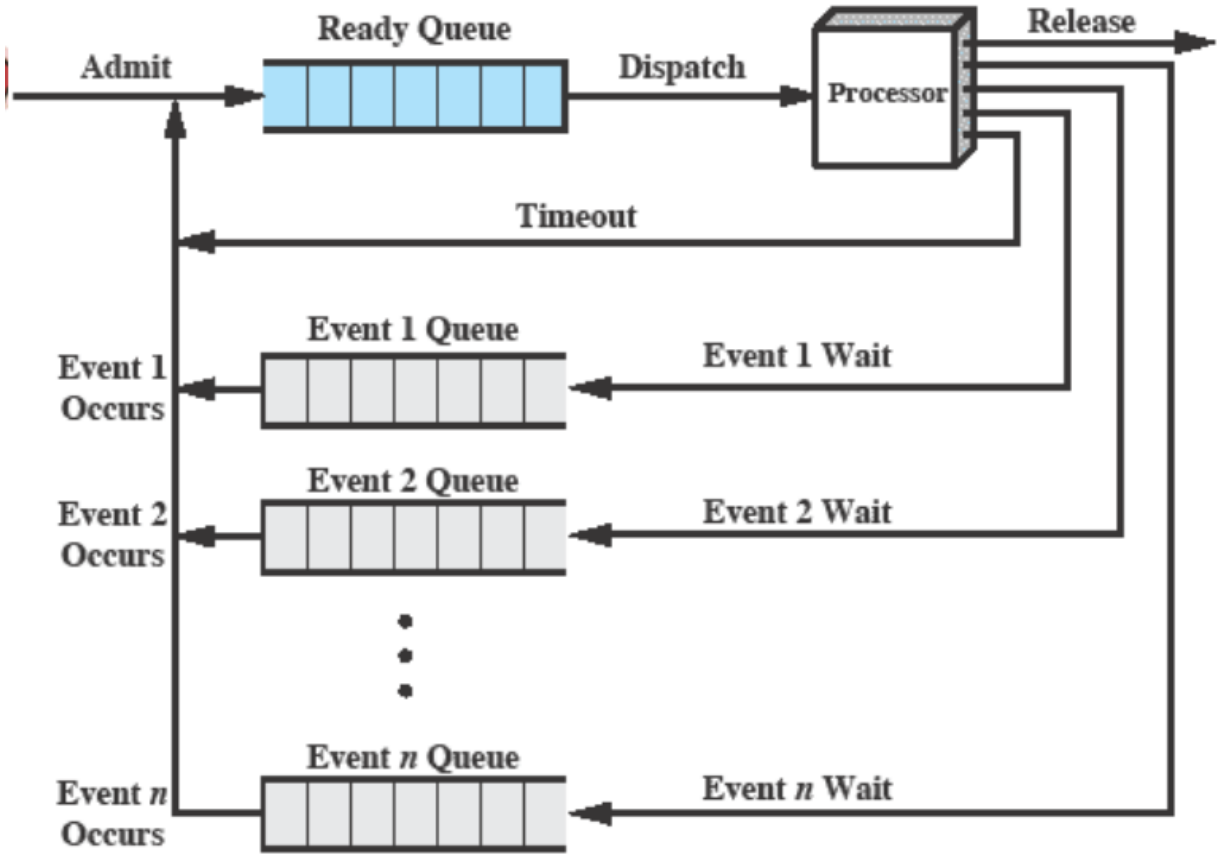
Usando Due Code

- Quando il processo *running* è tolto dall'esecuzione
  - può terminare, *oppure*
  - essere posto in una delle code *Ready* o *Blocked*



Molteplici Code Bloccanti

- Per una gestione più efficiente delle centinaia/migliaia di processi si usano multiple code *Blocked* basate su tipi di eventi



Stati di un processo

Processi sospesi

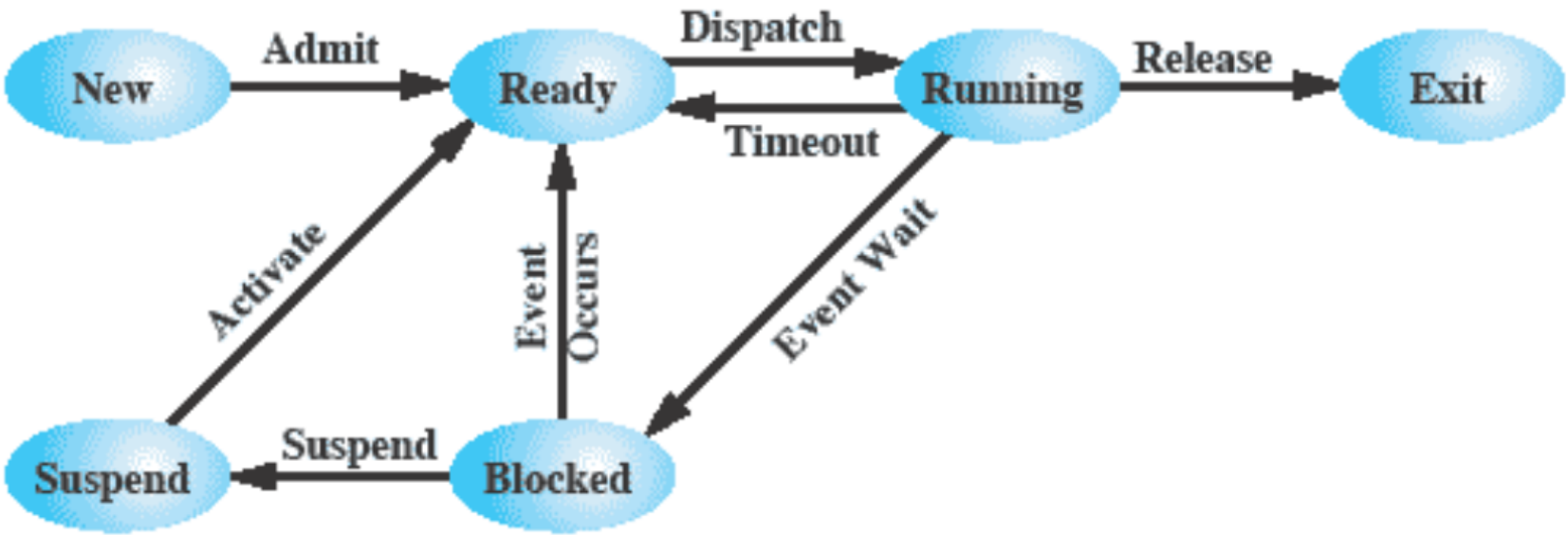
Processi Sospesi

- Oltre agli stati *Ready*, *Running* e *Blocked* può essere utile usare altri stati
- Ogni processo deve essere in memoria principale (*non considereremo la memoria virtuale*)
- Il processore è più veloce dell'I/O, quindi tutti i processi attualmente in memoria potrebbero essere in attesa di I/O

Processi Sospesi

- Per non lasciare il processore inoperoso i processi vengono spostati (*swap*) su disco, così da liberare memoria
- Quando il processo è swappato su disco lo stato da *blocked* diventa **suspended**
- Lo spazio liberato in memoria principale può essere usato per un altro processo:
  - un processo appena creato *oppure*
  - un processo precedentemente sospeso
  - *meglio un processo sospeso* per non sovraccaricare il sistema

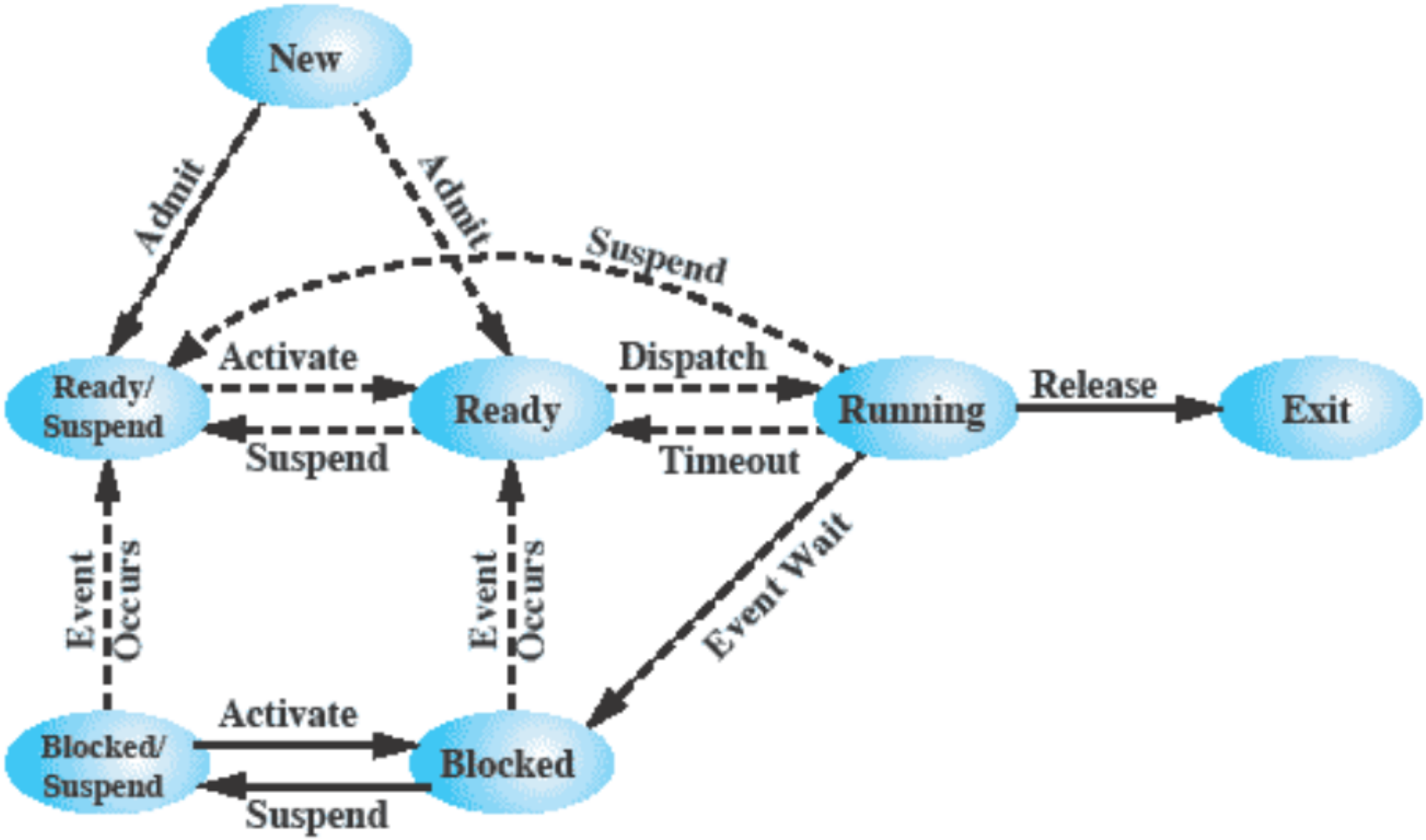
Stato Suspended



Processi Sospesi

- Non conviene riportare in memoria principale processi che non sono ancora pronti per l'esecuzione, cioè *bloccati*
- Ma se un processo era in attesa di un evento, quando l'evento si è verificato diventa pronto per l'esecuzione
- Due nuovi stati
  - **blocked/suspend** (swappato - in attesa dell'evento)
  - **ready/suspend** (swappato - pronto per l'esecuzione)

Due stati Suspended



Si può andare anche direttamente ad exit da un qualsiasi stato diverso da new (un processo ne *killa* un altro)

Motivi per Sospendere un Processo

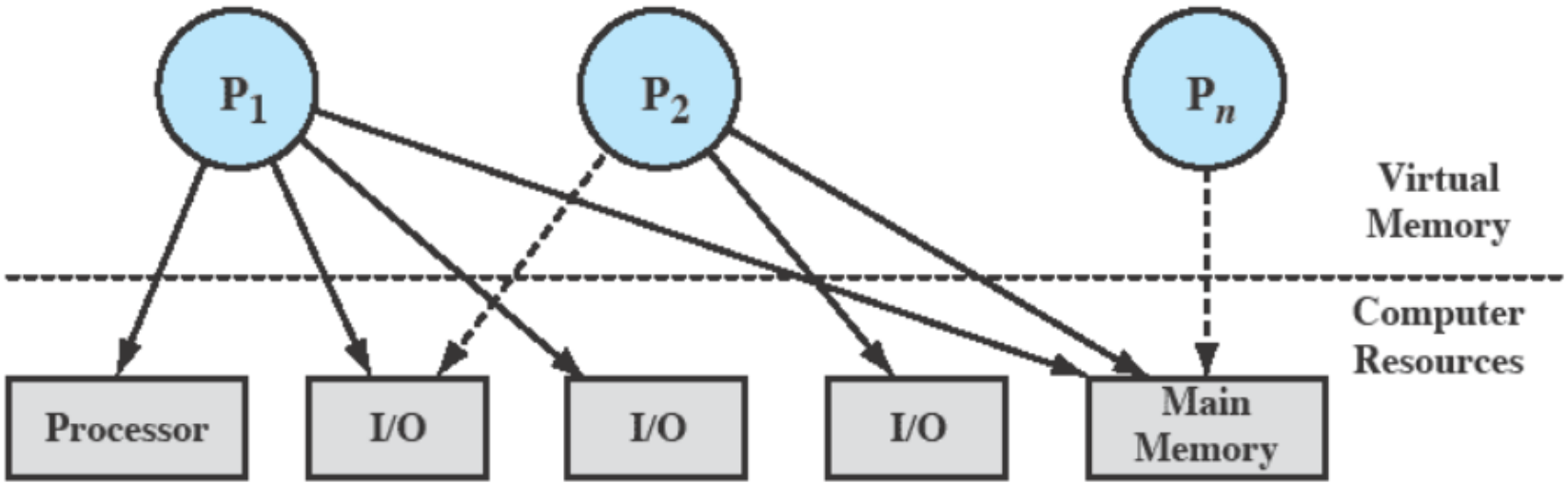
Motivo	**Commento**
Swapping	Il SO ha bisogno di liberare abbastanza memoria per caricare un processo ready
Interno al SO	Il SO sospetta che il processo stia causando problemi
Richiesta utente interattiva	Ad esempio: debugging o motivi legati all'uso di risorse
Periodicità	Il processo viene eseguito periodicamente (p.e. monitoraggio di sistema o accounting) e può venire sospeso in attesa della prossima esecuzione
Richiesta del padre	Il padre potrebbe voler sospendere l'esecuzione di un figlio per esaminarlo o modificarlo o per coordinare l'attività tra più figli

Descrizione del processo

Processi e risorse

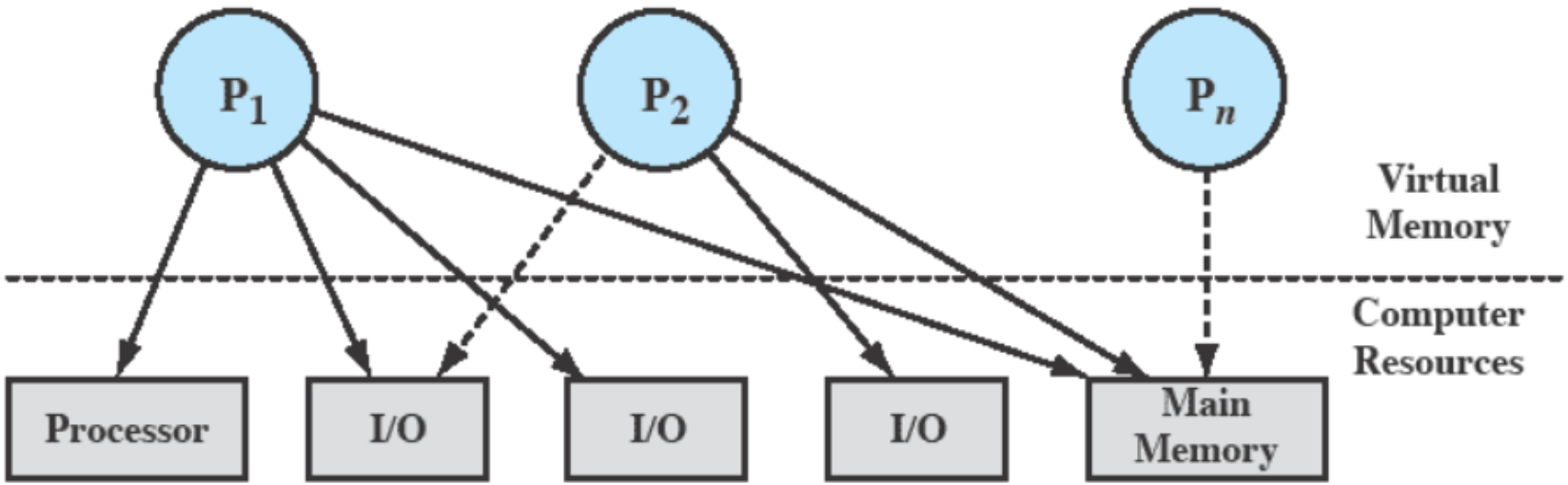
Processi e Risorse

- Il compito del sistema operativo è fondamentalmente la *gestione dell'uso delle risorse di sistema da parte dei processi* (processore *in primis*)
- In un sistema multiprogrammato si ha un insieme di processi che competono per l'utilizzo delle risorse comuni



Processi e Risorse

- P_1 è running, quindi almeno in parte è in memoria principale e usa processore e dispositivi di I/O
- P_2 è in attesa dell'I/O utilizzato da P_1
- P_n è stato swappato ed è sospeso



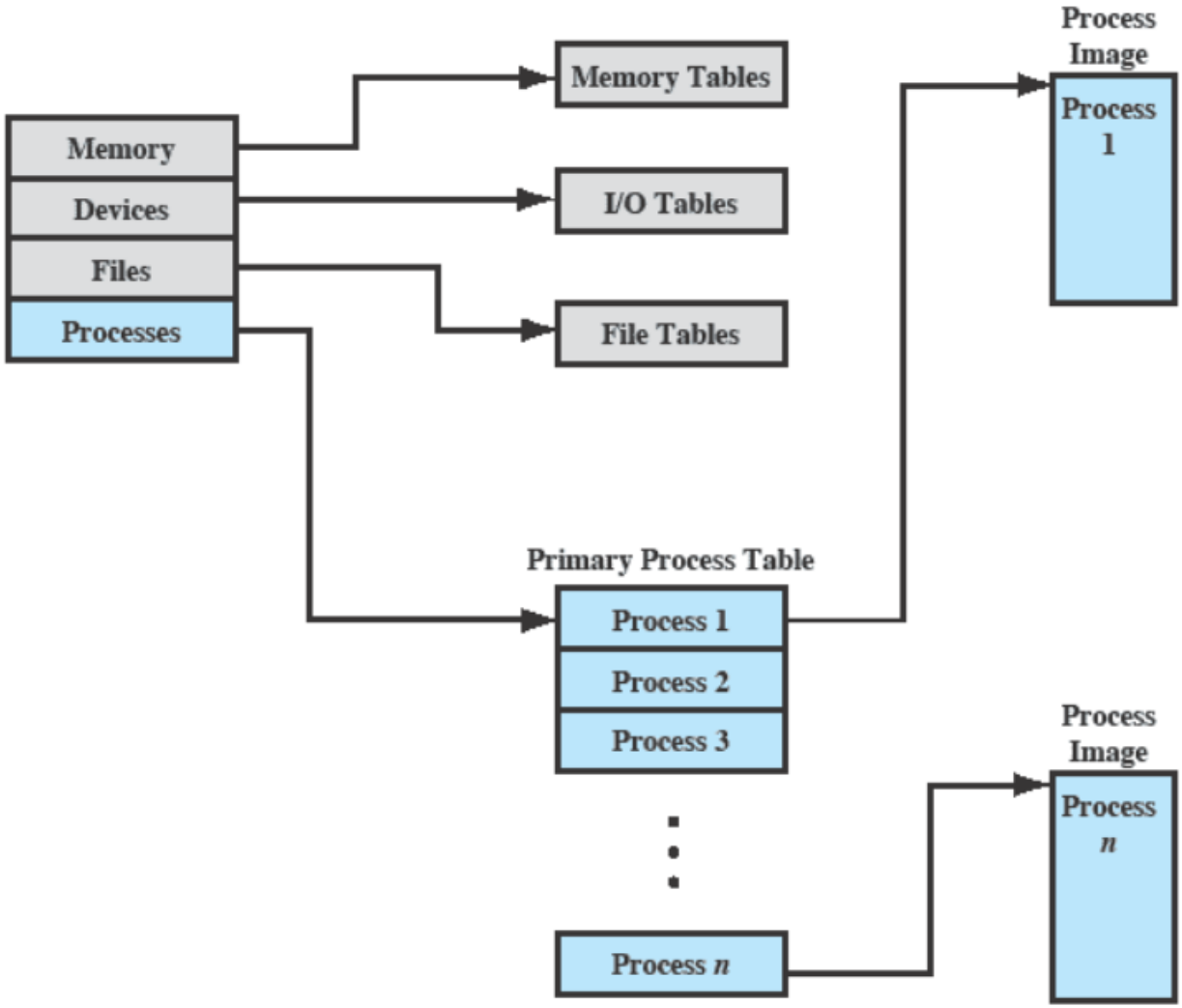
Descrizione del processo

Strutture di controllo del SO

Strutture di Controllo del SO

- Per gestire sia i processi che le risorse, il SO deve conoscere lo stato di ogni processo e di ogni risorsa
- Il SO costruisce e mantiene una o più tabelle per ogni entità da gestire
- I quattro tipi di tabelle mantenute dal SO sono:
  - memoria
  - I/O
  - file
  - processi

Tabelle di controllo del SO



Ovviamente, ci sono molti riferimenti incrociati

Tabelle di Memoria

- Le tabelle di memoria sono usate per gestire sia la memoria principale che quella secondaria
  - quella secondaria serve per la memoria virtuale, ci torneremo
- Devono comprendere le seguenti info:
  - allocazione di memoria principale da parte dei processi
  - allocazione di memoria secondaria da parte dei processi
  - attributi di protezione per l'accesso a zone di memoria condivisa
  - informazioni per gestire la memoria virtuale

Tabelle per l'I/O

- Usate dal SO per gestire i dispositivi e i canali di I/O
- Il SO deve sapere:
  - se il dispositivo è disponibile o già assegnato
  - lo stato dell'operazione di I/O
  - la locazione in memoria principale usata come sorgente o destinazione del trasferimento di I/O

Tabelle dei File

- Queste tabelle forniscono informazione su:
  - esistenza di files
  - locazioni in memoria secondaria
  - stato corrente
  - altri attributi
- Memorizzate parte su disco e parte in RAM

Descrizione del processo

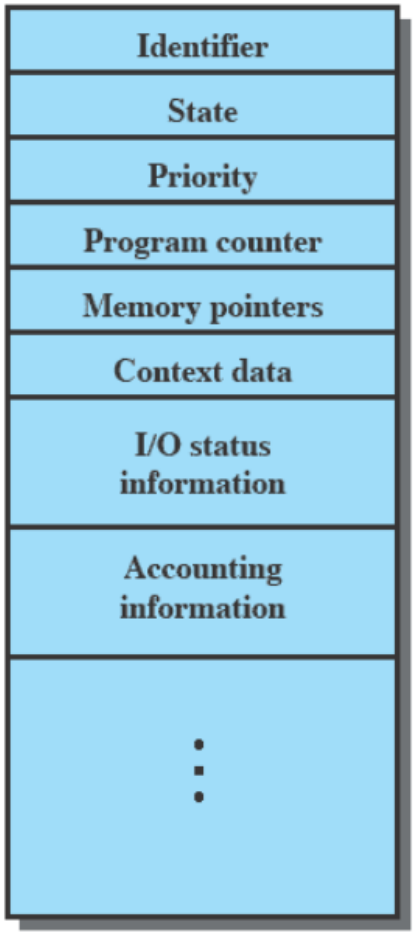
Strutture di controllo del processo

Tabelle dei Processi

- Per gestire i processi il SO deve conoscerne i dettagli (detti *attributi*):
  - stato corrente
  - identificatore
  - locazione in memoria
  - etc.
- Gli attributi del processo sono contenuti nel *blocco di controllo del processo* - **Process Control Block, PCB**
- L'insieme di programma sorgente, dati, stack delle chiamate e PCB è detto **process image** (immagine del processo)
  - eseguire un'istruzione cambia l'immagine: per esempio, modificando un registro o una cella di memoria

Attributi dei Processi

- Le informazioni in ciascun blocco di controllo (PCB) possono essere raggruppate in 3 categorie:
  - identificazione
  - stato
  - controllo
- Ogni sistema può organizzare queste informazioni in modo diverso



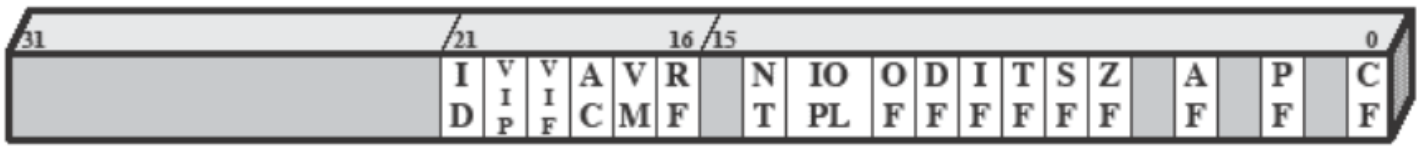
Come si Identifica un Processo

- Ad ogni processo è assegnato un numero identificativo, quindi unico: il **PID** (Process IDentifier)
- Molte tabelle del SO usano i PID per realizzare collegamenti tra le varie tabelle e la tabella dei processi
  - ad esempio, la tabella dei dispositivi I/O deve mantenere, per ogni dispositivo, quale processo lo sta usando
  - basta mettere il PID e implicitamente si può accedere alle informazioni sul processo corrispondente

Lo Stato del Processore

- Lo stato del processore non va confuso con lo stato, o meglio la modalità, del processo (ready, blocked, ...)
- È dato dai contenuti dei registri del processore stesso
  - registri visibili all'utente
  - registri di controllo e di stato
  - puntatori allo stack
- Tutti i processori includono il registro per la parola dello stato di programma (*Program status word*, **PSW**)
  - contiene informazioni di stato
  - esempio: il registro EFLAGS sui processori Pentium

EFLAGS nel Pentium 2



ID = Identification flag
VIP = Virtual interrupt pending
VIF = Virtual interrupt flag
AC = Alignment check
VM = Virtual 8086 mode
RF = Resume flag
NT = Nested task flag
IOPL = I/O privilege level
OF = Overflow flag

DF = Direction flag
IF = Interrupt enable flag
TF = Trap flag
SF = Sign flag
ZF = Zero flag
AF = Auxiliary carry flag
PF = Parity flag
CF = Carry flag

Control Block del Processo

- Riassumendo, il **PCB** contiene informazioni di cui il SO ha bisogno per controllare e coordinare i vari processi attivi
- Identificatori:
  - del processo (PID)
  - del processo padre (Parent PID, o PPID)
  - dell'utente proprietario
- Informazioni sullo stato del processo:
  - registri utente (accessibili in linguaggio macchina/assembler)
  - program counter
  - stack pointer
  - registri di stato: risultati di operazioni aritmetico/logiche, modalità di esecuzione, interrupt abilitati/disabilitati

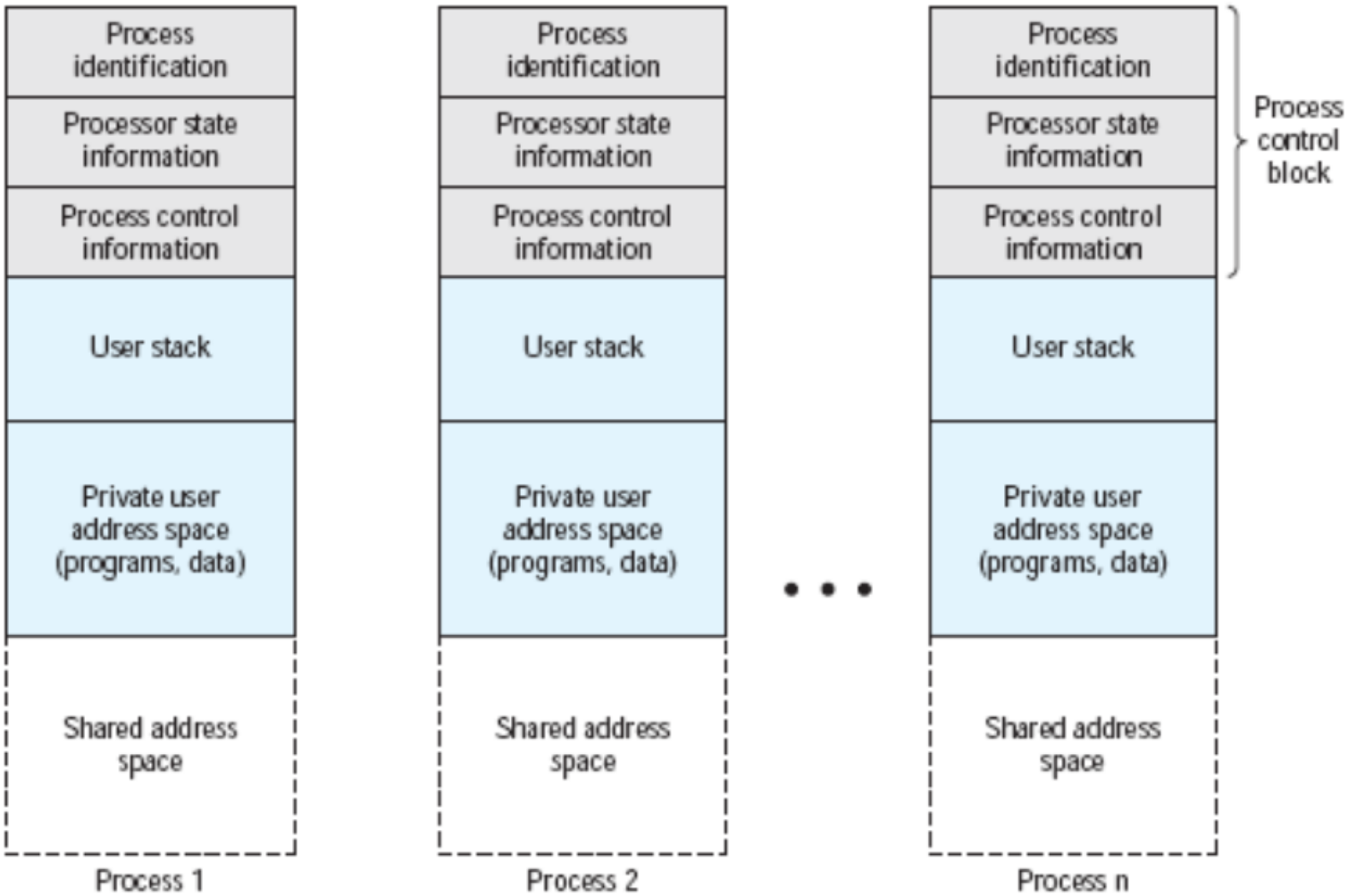
Control Block del Processo

- Informazioni per il controllo del processo:
  - stato del processo (ready, suspended, blocked, ...)
  - priorità
  - informazioni sullo scheduling (ad es.: per quanto tempo è stato in esecuzione l'ultima volta)
  - l'evento da attendere per tornare ad essere ready, se attualmente in attesa
- Supporto per strutture dati
  - puntatori ad altri processi
  - per mantenere liste concatenate di processi nei casi in cui siano necessarie (es., code di processi per qualche risorsa)
- Comunicazioni tra processi
  - flag, segnali, messaggi per supportare comunicazioni tra processi

Control Block del Processo

- Permessi speciali
  - non tutti i processi possono accedere a tutto
- Gestione della memoria
  - puntatori ad aree di memoria che gestiscono l'uso della memoria virtuale
  - es: pagine virtuali attualmente in uso
- Uso delle risorse
  - file aperti
  - uso di risorse (compreso il processore) fino ad ora

Immagini dei Processi in Memoria Virtuale



Control Block del Processo

In conclusione il **PCB**:

- È a struttura dati più importante di un sistema operativo
  - definisce lo stato del SO stesso
- Richiede protezioni
  - una funzione scritta male potrebbe danneggiare il blocco, rendendo il SO incapace di gestire i processi
  - ogni cambiamento nella progettazione del blocco ha effetti su molti moduli del SO

Controllo del processo

Modi di esecuzione

Modalità di Esecuzione

La maggior parte dei processori supporta almeno due modalità di esecuzione

- **Modo sistema**
  - pieno controllo: ad es., si possono eseguire istruzioni macchina che bloccano gli interrupt
  - si può accedere a qualsiasi locazione di RAM
  - per il kernel
- **Modo utente**
  - molte operazioni sono vietate
  - per i programmi utente
- *Pentium* ha addirittura 4 modalità
- *Linux* usa solo la modalità *ristretta*, *modalità utente*, e la modalità *senza limitazioni*, *modalità sistema o kernel*

Kernel Mode

- Il **kernel mode** è per le operazioni effettuate dal kernel
- Gestione dei processi (tramite PCB)
  - creazione e terminazione
  - pianificazione di lungo, medio e breve termine (*scheduling* e *dispatching*)
  - avvicendamento (*process switching*)
  - sincronizzazione e comunicazione
- Gestione della memoria principale
  - allocazione di spazio per i processi
  - gestione della memoria virtuale
- Gestione dell'I/O
  - gestione dei buffer e delle cache per l'I/O
  - assegnazione risorse I/O ai processi
- Funzioni di supporto
  - Gestione interrupt/eccezioni, accounting, monitoraggio

Creazione di un processo

Per creare un processo, il SO deve:

- Assegnargli un PID unico
- Allocargli spazio in memoria principale
- Inizializzare il process control block
- Inserire il processo nella giusta coda
  - ad es., ready oppure ready/suspended
- Creare o espandere altre strutture dati
  - ad es., quelle per l'accounting

Process Switching

Il **Process Switching** pone vari problemi

- Occorre distinguere tra switch di modalità di un processo e switching di processi
  - *switch di modalità*: da modalità utente a sistema e viceversa
    - tipicamente tramite system call
  - *switching tra processi*: per qualche motivo l'attuale processo non deve più usare il processore, che va concesso invece ad un altro processo
- Quali eventi determinano uno switch?
- Cosa deve fare il SO per tenere aggiornate tutte le strutture dati in seguito ad uno switch tra processi?

Quando effettuare uno switch

Un Process switch avviene quando il SO riprende il controllo, togliendolo al processo in esecuzione, per vari motivi.

Meccanismo	**Causa**	**Uso**
Interruzione	Esterna all'esecuzione dell'istruzione corrente	Reazione ad un evento esterno asincrono; include i quanti di tempo per lo scheduler
Eccezione	Associata all'esecuzione dell'istruzione corrente	Gestione di un errore sincrono
Chiamata al SO	Richiesta esplicita	Chiamata a funzione di sistema (caso particolare di eccezione)

Process Switch: Passaggi

1. Salvare il contesto del programma (registri e PC)
2. Aggiornare il process control block, attualmente in running
3. Spostare il process control block nella coda appropriata: ready; blocked; ready/suspend
4. Scegliere un altro processo da eseguire
5. Aggiornare il process control block del processo selezionato
6. Aggiornare le strutture dati per la gestione della memoria
7. Ripristinare il contesto del processo selezionato

Tutto in kernel mode

Esecuzione del sistema operativo

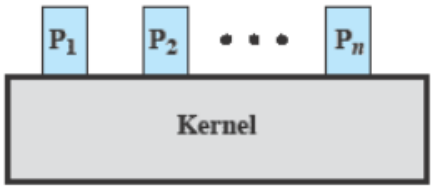
Il kernel

Il SO è un Processo?

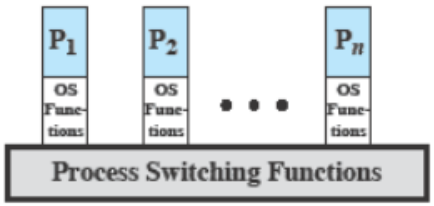
- Il SO è *solo* un insieme di programmi ed è eseguito dal processore come ogni altro programma
- Semplicemente, lascia che ogni tanto (in realtà, molto spesso) altri programmi vadano in esecuzione, per poi riprendere il controllo tramite interrupt
- Quindi, è esso stesso un processo?
- Se sì, come viene controllato?

Esecuzione del SO

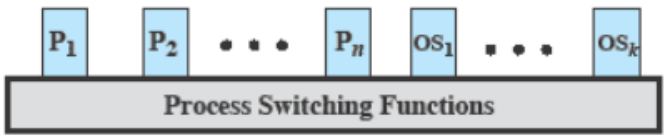
Ci sono vari approcci



(a) Separate kernel

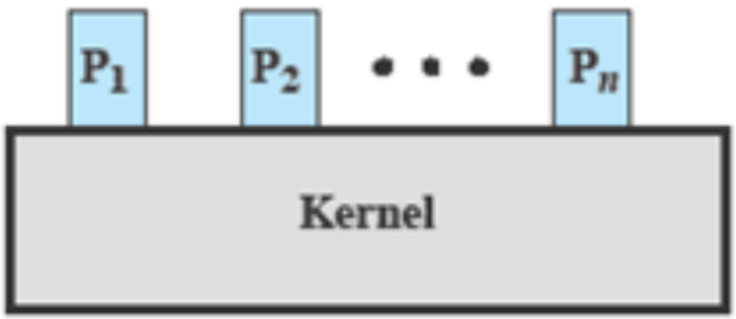


(b) OS functions execute within user processes



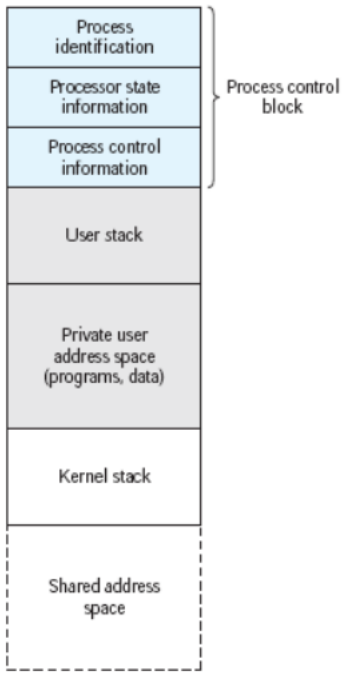
Kernel come non-processo

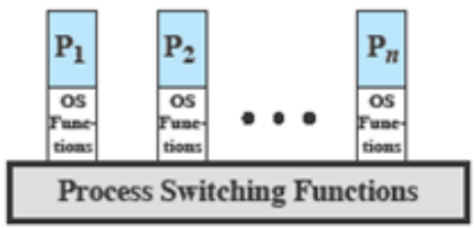
- **Kernel eseguito al di fuori dei processi**
- Il concetto di processo si applica solo ai programmi utente: quando un processo viene interrotto, viene salvato il contesto e il controllo passa al kernel
- Il SO è eseguito come un'entità separata, con privilegi più elevati
- Ha una sua zona di memoria dedicata sia per i dati che per il codice sorgente che per lo stack



Esecuzione *all'interno* dei Processi Utente

- **SO eseguito nel contesto di processo utente** (cambia solo la modalità di esecuzione)
- Non c'è bisogno di un *process switch* per eseguire una funzione del SO, solo del *mode switch*
- C'è uno stack delle chiamate separato usato per gestire chiamate e ritorni
- Dati e codice macchina sono condivisi tra i processi
- Process switch solo, eventualmente, alla fine, se lo scheduler decide che tocca ad un altro processo





SO è Basato sui Processi

- **SO implementato come un insieme di processi di sistema**
- Partecipano alla competizione per il processore accanto ai processi utente
- Le funzioni del kernel vengono eseguite in *kernel mode*, con privilegi più alti
- Una piccola parte del codice di switching-process è eseguito fuori da ogni processo (lo switch tra processi, però, non è un processo)