Sistemi Operativi

AAF - Secondo anno - 3CFU

A.A. 2024/2025

Corso di Laurea in Matematica

I Processi - Parte 1

Annalisa Massini

Dipartimento di Informatica
Sapienza Università di Roma

Argomenti trattati

1. Che cos'è un processo
   - I processi per il SO
   - Processi e programmi
2. Stati di un processo
   - Processi a due stati
   - Creazione e terminazione di processi
   - Processi a cinque stati

Che cos'è un processo

I processi per il 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
- 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 i concetti più importanti:

- 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
- Le applicazioni sono scritte senza riferimento ad una specifica architettura hardware
- Il SO fornisce un'interfaccia comune per le applicazioni (rappresenta 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 devono essere 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

Processi e programmi

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, programmi e loro esecuzione

- 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 il momento, con *processo in esecuzione* intendiamo che *un utente ha richiesto l'esecuzione di un programma e questo non è ancora 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, programmi e loro esecuzione

- 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
- Un processo ha 3 macrofasi: **creazione**, **esecuzione** e **terminazione**

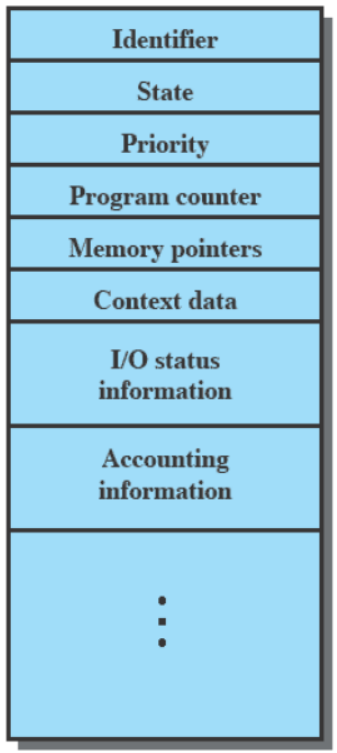
Elementi di un processo

- 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



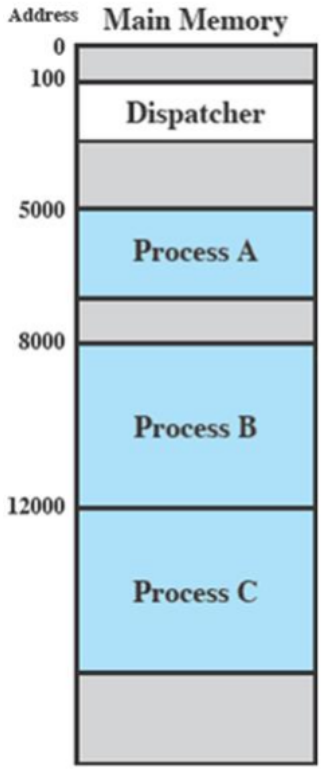
Traccia di un processo

- Come detto, 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**)
- La gestione del cambio di processo è affidata al **dispatcher**, 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
- Sia i processi che il dispatcher sono in memoria principale
- Ignoriamo i meccanismi di gestione della memoria (come ad esempio quello della memoria virtuale)



La traccia dal punto di vista del processo

- Nella tabella vediamo le tracce dei tre processi durante la prima parte della loro esecuzione
- Per i processi A e C vengono eseguite le prime 12 istruzioni
- Il processo B esegue le prime 4 istruzioni, poi invoca un'operazione di I/O e resta in attesa

(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

- I cicli di istruzione dei tre processi vengono eseguiti in modo alternato (*interleaved traces*)
- Le zone colorate si riferiscono all'esecuzione di istruzioni del dispatcher
- Nell'esempio si assume che vengano eseguite 6 istruzioni e poi ci sia un'interruzione per time-out

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

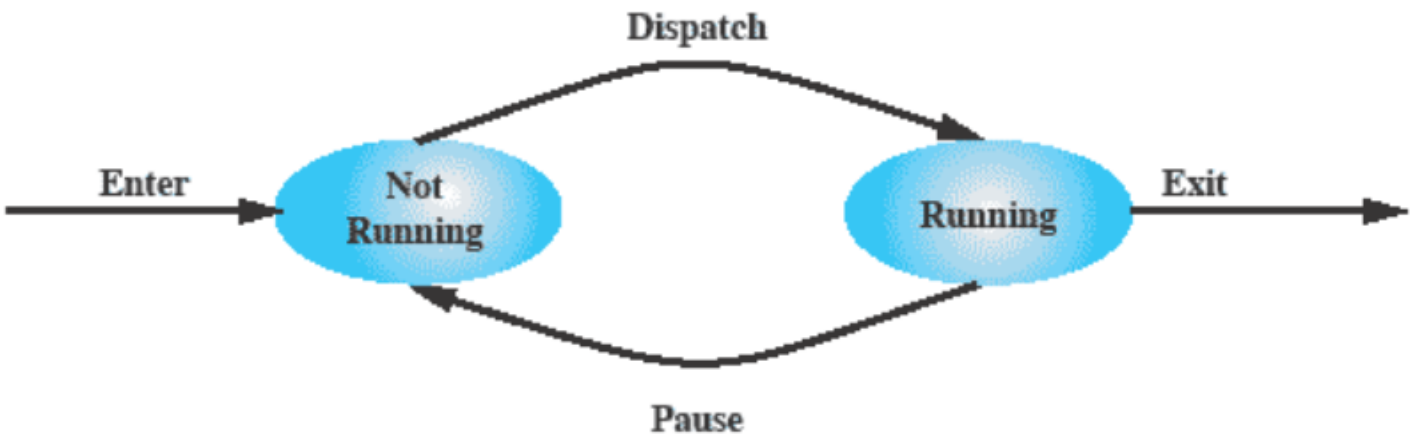
100 = Starting address of dispatcher program

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*

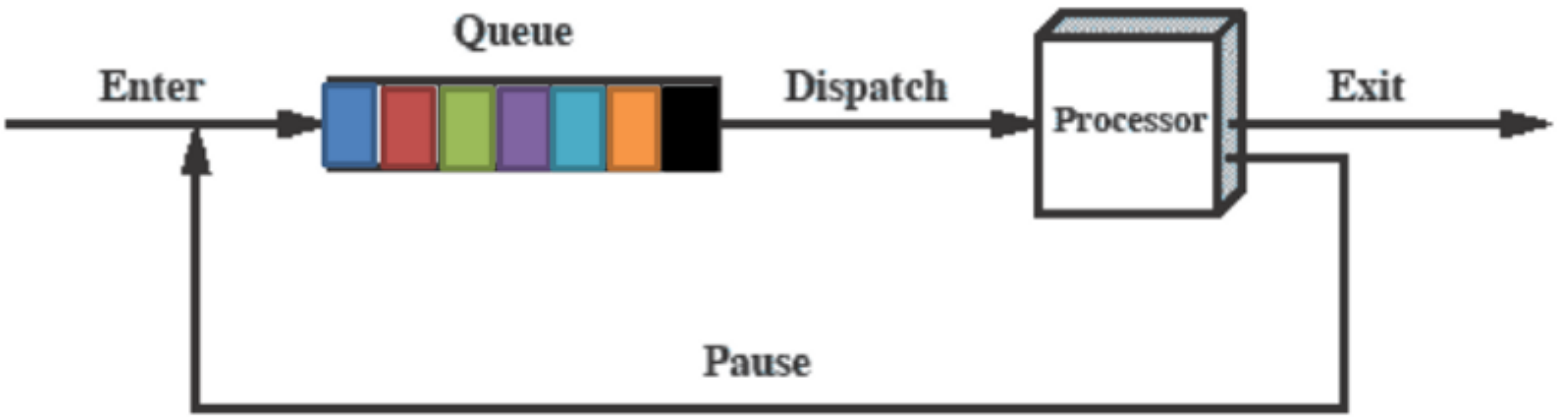
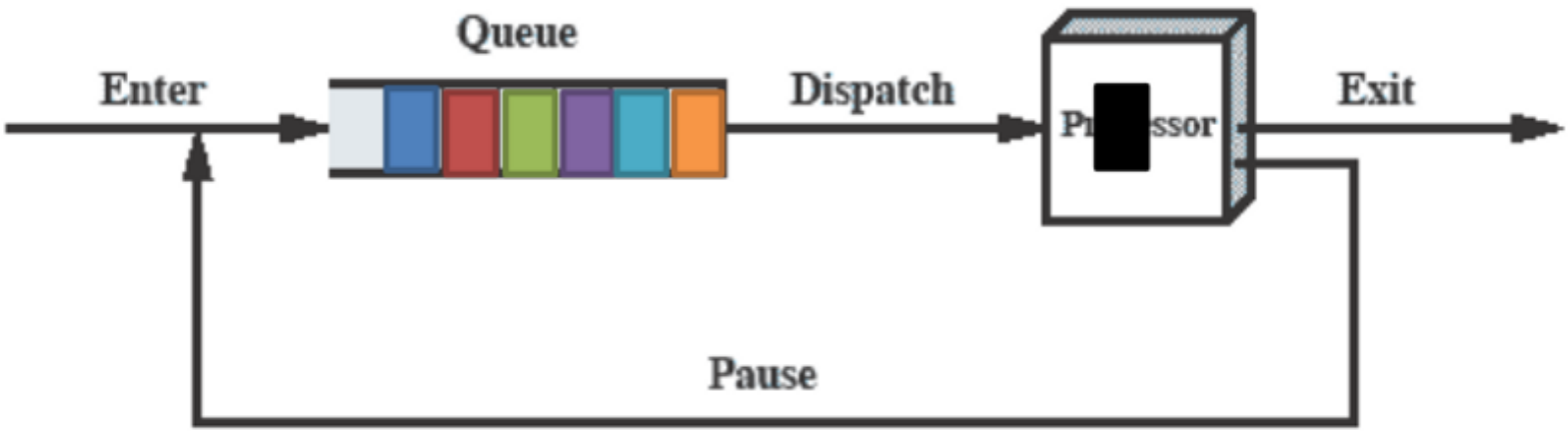


Diagramma a Coda

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* viene eseguito 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

Processi, esecuzioni e programmi

- La **terminazione** può essere:
  - *prevista*
    - esempio: un programma deve leggere numeri, ordinarli e scrivere l'output riordinato; alla fine dell'ultimo passo, il processo è terminato
    - esempio: word processor, 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*
    - 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
    - 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

Stati di un processo

Processi a cinque stati

Modello dei processi a 5 Stati

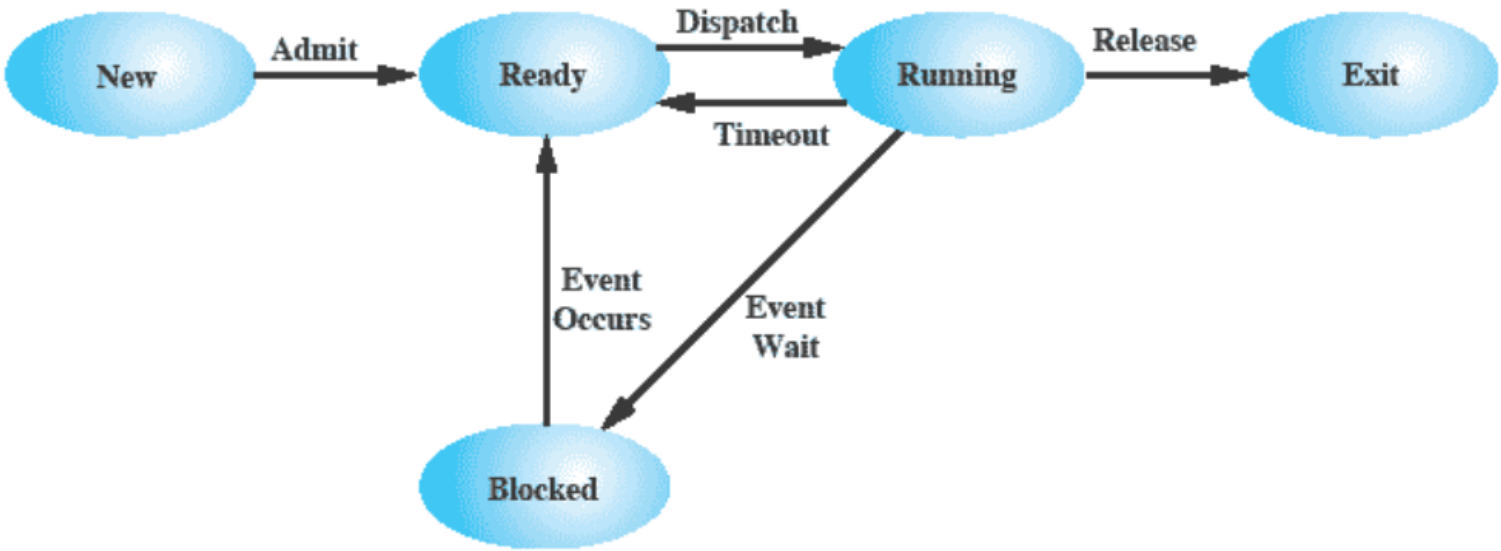
- Consideriamo il modello con la gestione a coda
- La coda è del tipo *first-in-first-out* (o FIFO)
- Il processore gestisce i processi in coda in *round-robin*, cioè ogni processo in coda riceve una certa quantità di tempo, poi torna in coda
- Il problema è che tra i processi *Not Running* ci sono:
  - processi *ready to execute*
  - processi *blocked* (in attesa di operazioni di I/O)

Modello dei processi a 5 Stati

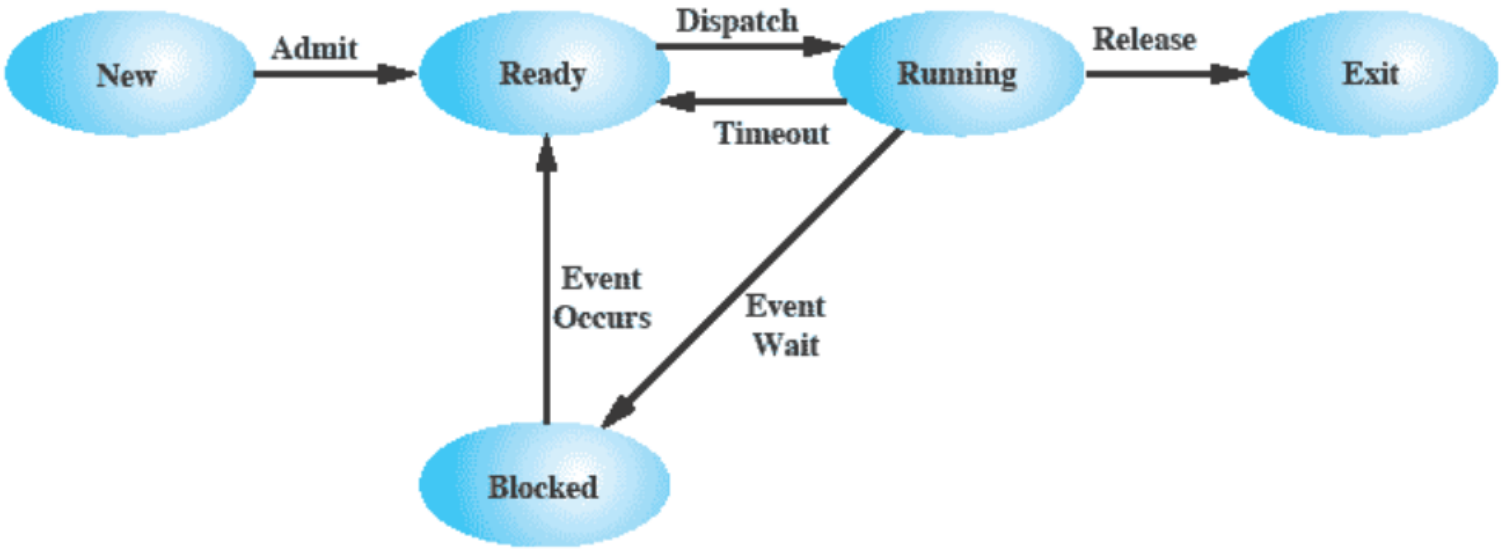
- Il dispatcher deve quindi cercare il processo non bloccato che si trova da più tempo in coda
- Usando una singola coda, il *dispatcher* non riesce a individuare il processo più vecchio in coda efficientemente
- Si può passare dal modello di processo a due stati al modello a cinque stati

Modello dei processi a 5 Stati

- Si risolve il problema di individuare il prossimo processo da mandare in esecuzione suddividendo lo stato *Not Running* nei due stati **Ready** e **Blocked**
- Si introducono gli stati *New* ed *Exit*

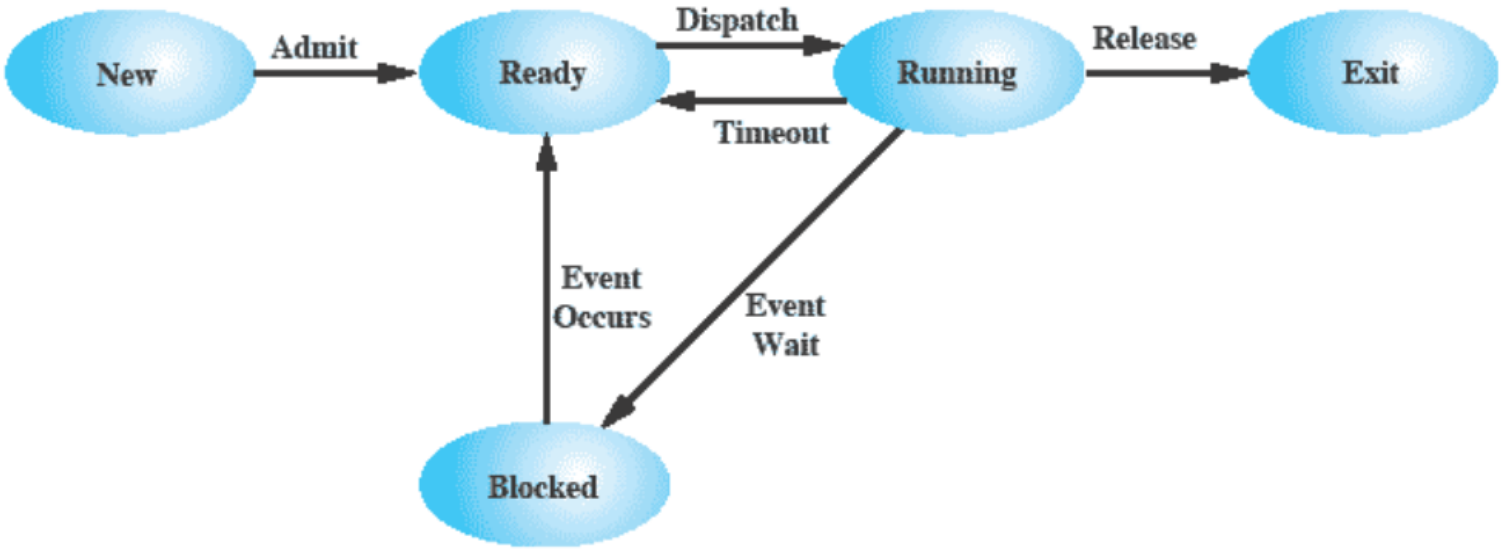


Modello dei Processi a 5 Stati



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

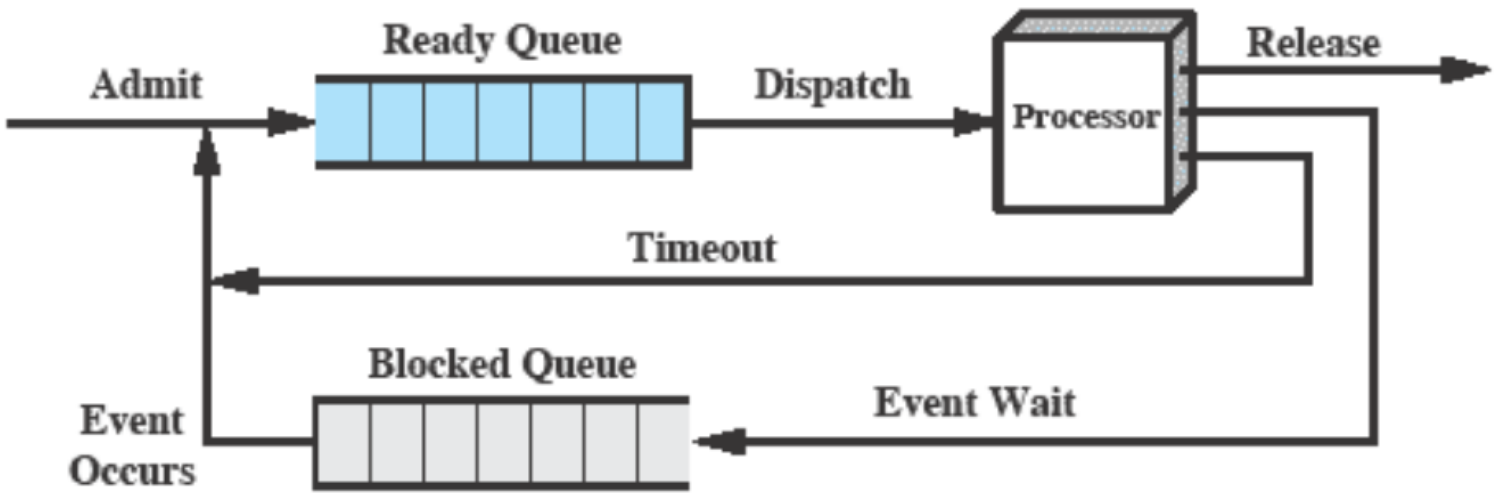
Modello dei Processi a 5 Stati



- Si può passare da ready a running
- Si può passare da running a ready oppure blocked
- Si può anche passare da ready o blocked ad exit (se un processo ne *uccide* un altro)
- *Waiting* è spesso usato in alternativa a *blocked*

Gestione a Due Code

- Il processo *running*, cioè in esecuzion,e 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