UNIVERSITÀ DEGLI STUDI DI NAPOLI
PARTHENOPE

Sistemi Operativi

Processi

LEZIONE 4

prof. Antonino Staiano

Corso di Laurea in Informatica – Università di Napoli Parthenope

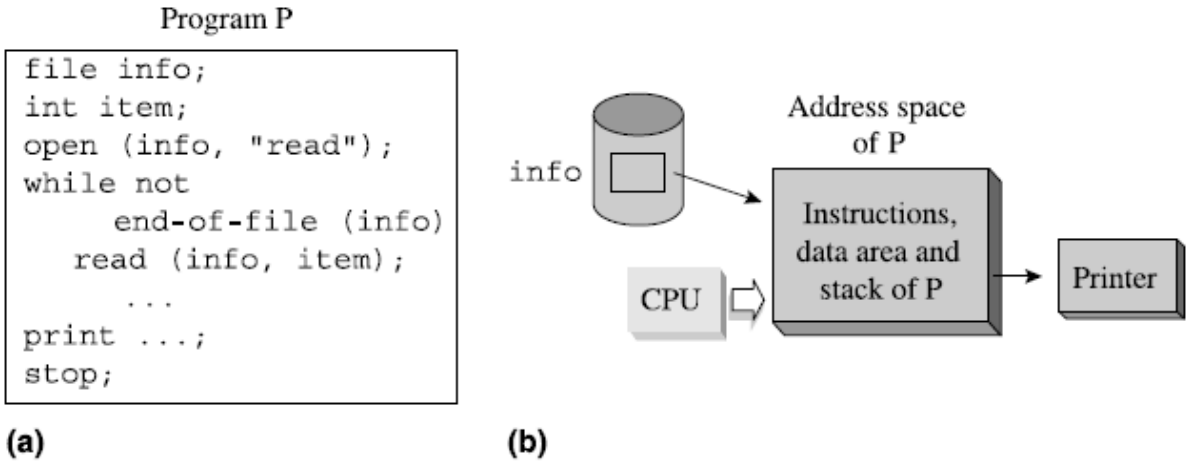
antonino.staiano@uniparthenope.it

Processi e Programmi

- Cosa è un processo
- Relazione tra processi e programmi
- Processi figli
- Concorrenza e parallelismo

Cosa è un processo?

Processo: esecuzione di un programma usando le risorse ad esso allocate



Un programma e una vista astratta delle sua esecuzione

Componenti di un Processo

- Un processo comprende sei componenti
  - (id, codice, dati, stack, risorse, stato CPU)
    - **Id** identificativo assegnato dal SO
    - **Codice** è il codice del programma
    - **Dati** dati usati durante l'esecuzione (inclusi quelli contenuti nei file usati dal programma)
    - **Stack** contiene parametri delle funzioni e procedure invocate durante l'esecuzione e loro indirizzi di ritorno
    - **Risorse** è l'insieme di risorse allocate dal SO
    - **Stato della CPU** composto da PSW e GPR
    - Ricordiamo che PSW

Program counter (PC)	Condition code (CC)	Mode (M)	Memory protection information (MPI)	Interrupt mask (IM)	Interrupt code (IC)

Relazioni tra Processi e Programmi

- Un programma è un insieme di funzioni e procedure
  - Le funzioni possono essere processi separati o possono costituire la parte di codice di un singolo processo

Relationship	Examples
One-to-one	A single execution of a sequential program.
Many-to-one	Many simultaneous executions of a program, execution of a concurrent program.

Processi figli

- Il kernel avvia un'esecuzione di un programma creandogli un processo
  - Il processo originario può effettuare una chiamata di sistema per creare altri processi
    - Processi figli e genitori creano un albero dei processi
- Tipicamente, un processo crea uno o più processi figli e attribuisce parte del proprio lavoro a ciascuno
  - Multi-tasking all'interno di un'applicazione

Processi figli (cont.)

- Benefici dei processi figli

Benefit	Explanation
Computation speedup	Actions that the primary process of an application would have performed sequentially if it did not create child processes, would be performed concurrently when it creates child processes. It may reduce the duration, i.e., running time, of the application.
Priority for critical functions	A child process that performs a critical function may be assigned a high priority; it may help to meet the real-time requirements of an application.
Guarding a parent process against errors	The kernel aborts a child process if an error arises during its operation. The parent process is not affected by the error; it may be able to perform a recovery action.

Esempio: processi figli in un'applicazione real-time

Il processo iniziale, data_logger, deve eseguire tre funzioni:

1. Copiare il campione da uno speciale registro in memoria
2. Copiare il campione dalla memoria in un file
3. Effettuare alcune analisi del campione e memorizzare il risultato in un altro file usato per elaborazioni future

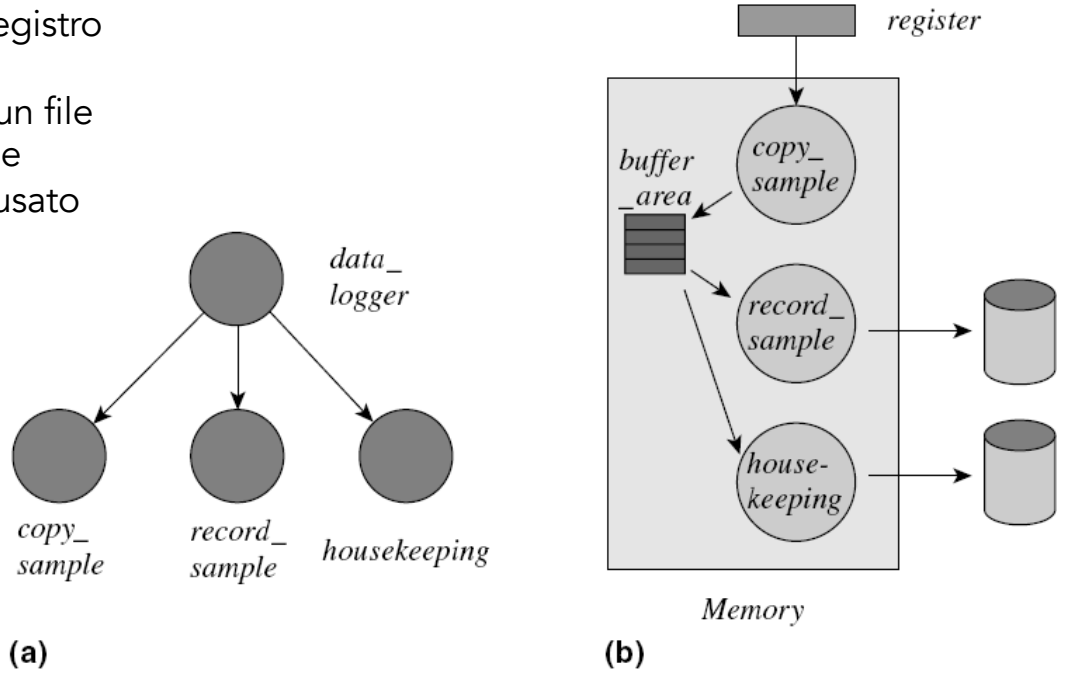


Figure 5.2 Real-time application of Section 3.7: (a) process tree; (b) processes.

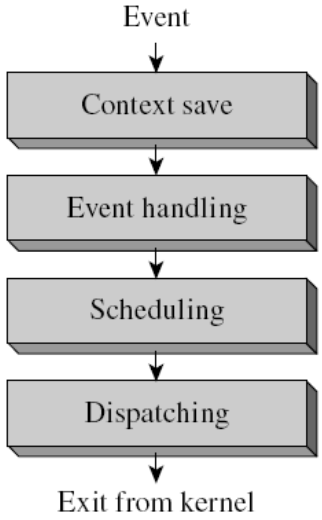
Albero dei processi e processi

Concorrenza e Parallelismo

- **Parallelismo**: qualità che consiste nel *procedere allo stesso momento*
  - Due attività sono parallele se sono eseguite allo stesso tempo
  - Ottenuto usando CPU multiple
    - Come in un sistema multi-processore
- **Concorrenza**: è un'illusione di parallelismo
  - Due attività sono concorrenti se c'è un'illusione che esse sono eseguite in parallelo laddove solo una di loro può essere eseguite in dato momento
  - In un SO, è ottenuta alternando le operazioni di vari processi sulla CPU
- Concorrenza e parallelismo possono fornire un migliore throughput

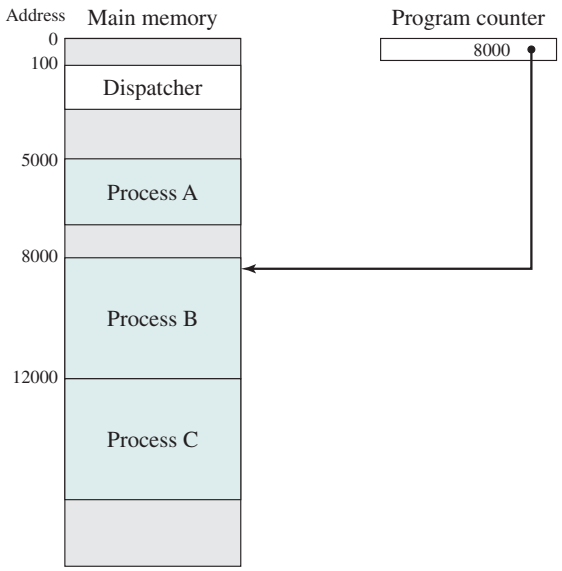
Implementazione dei Processi

- Per un SO, un processo è un'unità di lavoro computazionale
  - Il compito primario del kernel è controllare le operazioni dei processi per assicurare un uso efficace di un sistema di computer



Funzioni fondamentali del kernel per controllare i processi

Esempio esecuzione processi



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

Indirizzi delle istruzioni di A, B e C

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

100 = indirizzo di partenza del dispatcher

Supponiamo un ciclo di 6 istruzioni per ogni processo prima di essere interrotto

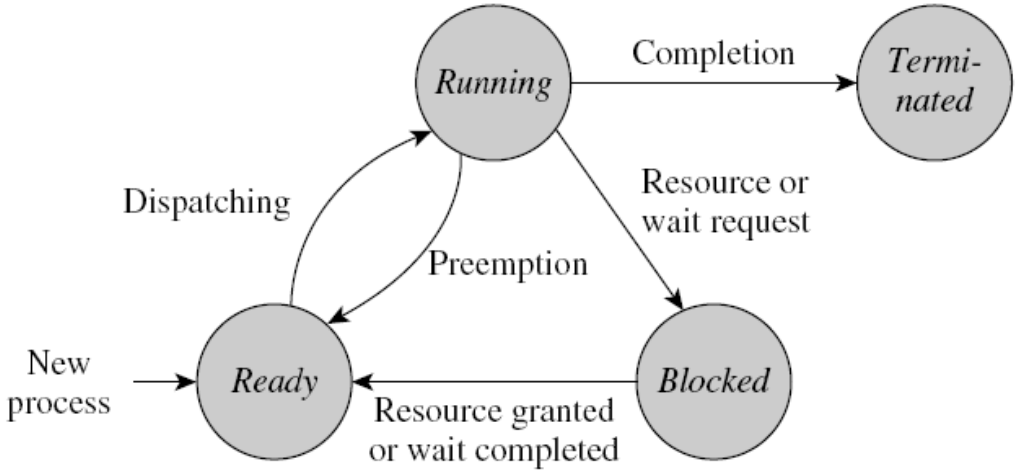
Stati di un Processo e Transizioni di Stato

Stato di un processo: l'indicatore che descrive la natura dell'attività corrente di un processo

State	Description
Running	A CPU is currently executing instructions in the process code.
Blocked	The process has to wait until a resource request made by it is granted, or it wishes to wait until a specific event occurs.
Ready	The process wishes to use the CPU to continue its operation; however, it has not been dispatched.
Terminated	The operation of the process, i.e., the execution of the program represented by it, has completed normally, or the OS has aborted it.

Stati di un Processo e Transizioni di Stato (cont.)

- Una *transizione di stato* per un processo è un cambio del proprio stato
  - Causata dall'occorrenza di qualche evento come l'inizio o la fine di un'operazione di I/O



Stati di un Processo e Transizioni di Stato (cont.)

Cause delle transizioni di stato fondamentali di un processo

State transition	Description
ready → *running*	The process is dispatched. The CPU begins or resumes execution of its instructions.
blocked → *ready*	A request made by the process is granted or an event for which it was waiting occurs.
running → *ready*	The process is preempted because the kernel decides to schedule some other process. This transition occurs either because a higher-priority process becomes *ready*, or because the time slice of the process elapses.
running → *blocked*	The process in operation makes a system call to indicate that it wishes to wait until some resource request made by it is granted, or until a specific event occurs in the system. Five major causes of blocking are: • Process requests an I/O operation • Process requests a resource • Process wishes to wait for a specified interval of time • Process waits for a message from another process • Process waits for some action by another process.

State transition	Description
running → *terminated*	Execution of the program is completed. Five primary reasons for process termination are: • *Self-termination:* The process in operation either completes its task or realizes that it cannot operate meaningfully and makes a "terminate me" system call. Examples of the latter condition are incorrect or inconsistent data, or inability to access data in a desired manner, e.g., incorrect file access privileges. • *Termination by a parent:* A process makes a "terminate P_i" system call to terminate a child process P_i, when it finds that execution of the child process is no longer necessary or meaningful. • *Exceeding resource utilization:* An OS may limit the resources that a process may consume. A process exceeding a resource limit would be aborted by the kernel. • *Abnormal conditions during operation:* The kernel aborts a process if an abnormal condition arises due to the instruction being executed, e.g., execution of an invalid instruction, execution of a privileged instruction, arithmetic conditions like overflow, or memory protection violation. • *Incorrect interaction with other processes:* The kernel may abort a process if it gets involved in a deadlock.

Esempio: transizioni di stato per un processo

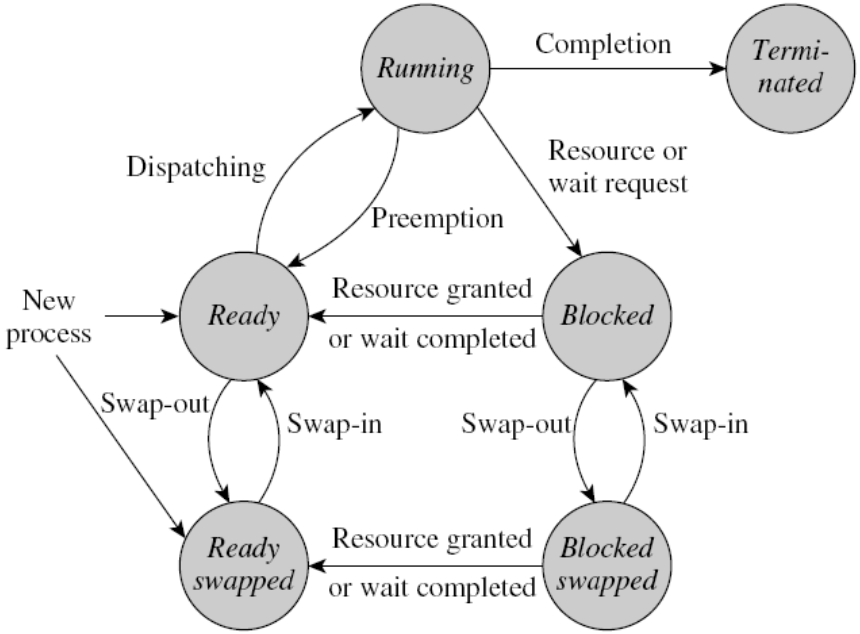
- Un sistema time-sharing ha due processi P1 e P2
  - Time slice= 10ms
  - P1 -> CPU burst 15 ms e operazione di I/O 100 ms
  - P2 -> CPU burst 30 ms e operazione di I/O di 60 ms

			New states	
Time	**Event**	**Remarks**	P_1	P_2
0		P_1 is scheduled	*running*	*ready*
10	P_1 is preempted	P_2 is scheduled	*ready*	*running*
20	P_2 is preempted	P_1 is scheduled	*running*	*ready*
25	P_1 starts I/O	P_2 is scheduled	*blocked*	*running*
35	P_2 is preempted	—	*blocked*	*ready*
		P_2 is scheduled	*blocked*	*running*
45	P_2 starts I/O	—	*blocked*	*blocked*

Transizioni di stato in un sistema time-sharing

Processi Sospesi

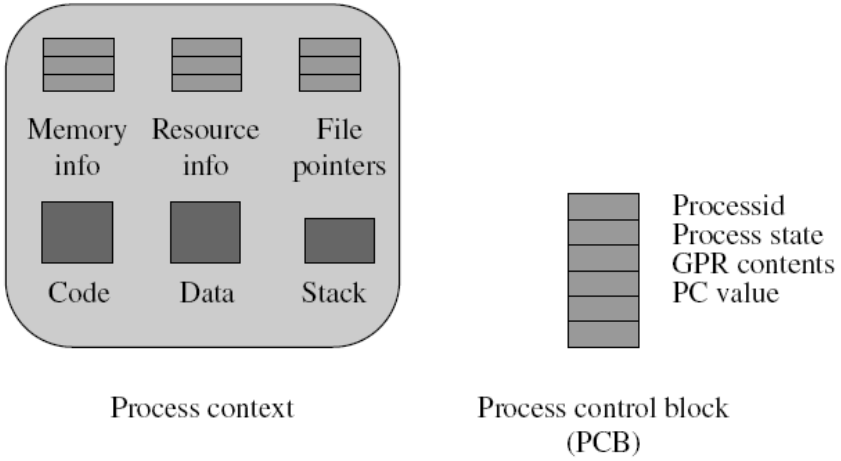
- Un kernel ha bisogno di stati aggiuntivi per descrivere i processi sospesi per lo swapping



Stati di processo e transizioni di stati usando due stati swapped

Contesto di un processo e Process Control Block

- Il kernel alloca risorse ad un processo e lo schedula per l'uso della CPU
  - La visione di un processo nel kernel consiste del *contesto del processo* ed il *process control block*



Contesto di un Processo

- **Spazio di indirizzamento del processo**: componenti del processo relativi a codice, dati e stack
- **Informazioni di allocazione della memoria**: le aree di memoria allocate al processo. Usate dalla MMU
- **Stato delle attività sui file**: info su file in uso, ad esempio, posizioni nei file
- **Informazioni sulle interazioni del processo**: info necessarie per controllare le interazioni del processo con altri processi
- **Informazioni sulle risorse**: info sulle risorse allocate al processo
- **Informazioni di miscellanea**: mix di informazioni necessarie per il funzionamento del processo

Process Control Block

- Contiene tre tipologie di informazioni
  1. Info di identificazione
     - Id del processo, id del processo genitore, id utente che lo ha creato
  2. Informazioni di stato del processo
     - Stato, contenuto del PSW e GPR
  3. Info per il controllo delle operazioni
     - Priorità, interazione con altri processi

- Contiene anche un campo puntatore usato dal kernel per costruire la lista di PCB per lo scheduling (i processi ready)

Campi del Process Control Block (PCB)

PCB field	Contents
Process id	The unique id assigned to the process at its creation.
Parent, child ids	These ids are used for process synchronization, typically for a process to check if a child process has terminated.
Priority	The priority is typically a numeric value. A process is assigned a priority at its creation. The kernel may change the priority dynamically depending on the nature of the process (whether CPU-bound or I/O-bound), its age, and the resources consumed by it (typically CPU time).
Process state	The current state of the process.
PSW	This is a snapshot, i.e., an image, of the PSW when the process last got blocked or was preempted. Loading this snapshot back into the PSW would resume operation of the process. (See Fig. 2.2 for fields of the PSW.)
GPRs	Contents of the general-purpose registers when the process last got blocked or was preempted.
Event information	For a process in the *blocked* state, this field contains information concerning the event for which the process is waiting.
Signal information	Information concerning locations of signal handlers (see Section 5.2.6).
PCB pointer	This field is used to form a list of PCBs for scheduling purposes.

Salvataggio del Contesto, Scheduling e Dispatching

- Funzione di salvataggio del contesto
  - Salva lo stato della CPU del processo interrotto nel PCB e salva informazioni riguardo al contesto
  - Cambia lo stato del processo da *running* a *ready*
- Funzione di scheduling
  - Usa le informazioni sullo stato dai PCB per selezionare un processo ready per l'esecuzione e passa il suo id alla funzione di dispatching
- Funzione di dispatching
  - Imposta il contesto del processo selezionato, cambia il suo stato a running e carica lo stato salvato della CPU dal PCB nella CPU
  - Scarica i buffer di traduzione dell'indirizzo usati dalla MMU

Esempio: Salvataggio del Contesto, Scheduling e Dispatching

- Un SO contiene due processi P_1 e P_2 (priorità $P_2 > P_1$)
  - P_2 *bloccato* su un'operazione di I/O e P_1 in *esecuzione*
- Quando l'I/O di P_2 è completata:
  1. E' eseguita la funzione di salvataggio del contesto per P_1 ed il suo stato è cambiato in *ready*
  2. Dal campo info evento del PCB, il gestore sa che l'I/O è stato iniziato da P_2, quindi cambia lo stato di P_2 da *blocked* a *ready*
  3. E' eseguito lo scheduling che seleziona P_2 poiché ha priorità maggiore tra i processi *pronti*
  4. Lo stato di P_2 è cambiato in *running* ed avviene il dispatch

Commutazione di Processo

- Le azioni 1, 3 e 4 insieme costituiscono la commutazione tra i processi P_1 e P_2
- La commutazione avviene anche quando lo stato di un processo in *running* diventa *bloccato* in conseguenza di una richiesta o della prelazione al termine di un time slice
- L'occorrenza di un evento non comporta la commutazione se esso
  - Causa una transizione di stato solo in un processo la cui priorità è più bassa del processo interrotto dall'evento
  - Non causa alcuna transizione di stato, ad esempio, se l'evento è causato da una richiesta immediatamente soddisfatta

Commutazione di Processo (cont.)

- L'overhead dipende dalla dimensione delle informazioni sullo stato del processo
- Alcune architetture di computer forniscono istruzioni speciali per ridurre l'overhead della commutazione
  - Salvano o caricano il PSW e tutti i GPR; scaricano i buffer di traduzione indirizzo usati dalla MMU
- Osservazione:
  - la commutazione può comportare anche un overhead indiretto
  - Il nuovo processo potrebbe non avere alcuna parte del suo spazio di indirizzamento nella cache e quindi le sue prestazioni sono scadenti fino a quando sono memorizzate sufficienti info nella cache

Gestione degli Eventi

- Gli eventi occorrono durante il funzionamento di un SO
  1. Evento creazione del processo
  2. Evento terminazione del processo
  3. Evento del timer
  4. Evento richiesta risorsa
  5. Evento rilascio risorsa
  6. Evento richiesta avvio I/O

Gestione degli Eventi (cont.)

- Gli eventi occorrono durante il funzionamento di un SO (cont.)
  7. Evento completamento I/O
  8. Evento Invio messaggio
  9. Evento ricezione messaggio
  10. Evento invio segnale
  11. Evento ricezione segnale
  12. Interrupt di un programma
  13. Evento malfunzionamento hardware

Eventi: Creazione Processi

- Quando un **utente invoca un comando** o un **processo intende creare un processo figlio** per eseguire un programma…
  - System call per la creazione del processo e nome programma da eseguire
  - La routine di gestione evento crea un PCB per il nuovo processo
    - Assegna un id unico ed una priorità al processo
    - Inserisce tali info nel PCB con anche l'id del processo padre
    - Determina la quantità di memoria da allocare per lo spazio di indirizzamento del processo e dispone il tutto per l'allocazione della memoria
  - Eventualmente, dal kernel sono associate al processo alcune risorse standard (canali di I/O standard)
    - Inserisce le info sulla memoria e le risorse allocate nel contesto del nuovo processo
    - Imposta il processo allo stato *ready* nel suo PCB e lo immette in una lista di PCB

Eventi: Terminazione Processi

- Quando un processo invoca una system call per la terminazione di se stesso o di un processo figlio
  - Il kernel ritarda la terminazione fino al momento in cui le operazioni di I/O avviate dal processo sono terminate
  - Rilascia la memoria e le risorse allocate
    - Sulla base delle info nel contesto del processo
  - Lo stato del processo è cambiato in terminated
  - Il PCB del processo terminato non viene rimosso fino a quando il processo padre non ne preleva il suo stato di terminazione
  - Nel caso, il padre era in attesa della terminazione del processo figlio, il kernel lo deve attivare
    - Il kernel prende l'id del processo padre dal PCB del processo terminato e controlla il campo info evento del PCB del processo padre per verificare se il processo padre è in attesa della terminazione del processo figlio

Eventi: Prelazione di Processi

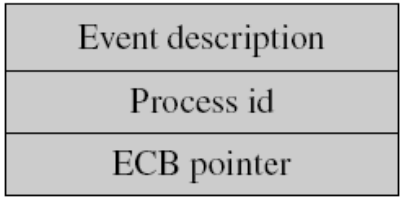
- Un processo nello stato *running* deve essere prelazionato se scade il suo time slice
- La funzione di salvataggio contesto cambia lo stato del processo *running* in *ready* prima di invocare il gestore dell'evento *timer interrupt*
  - Il gestore sposta solo il PCB del processo in un'opportuna lista di scheduling
- La prelazione deve avvenire anche quando un processo con maggiore priorità diventa *ready*
  - Ciò è fatto implicitamente quando è schedulato un processo a priorità maggiore (il gestore dell'evento non deve fare alcuna azione esplicita)

Eventi: Utilizzo Risorse

- Quando un processo richiede una risorsa che non può essere allocata subito
  - Il gestore dell'evento cambia lo stato del processo in *blocked* ed annota l'id della risorsa richiesta nel campo *info evento* del PCB
- Se un processo rilascia una risorsa con una system call, il gestore non deve cambiare lo stato di tale processo
  - Deve controllare se altri processi sono bloccati in attesa di tale risorsa ed allocarla eventualmente ad uno di essi cambiandone lo stato da blocked a ready
- Una system call per l'avvio di un'operazione di I/O e un interrupt per segnalare il completamento di un'operazione di I/O comportano azioni di gestione dell'evento simili alle precedenti

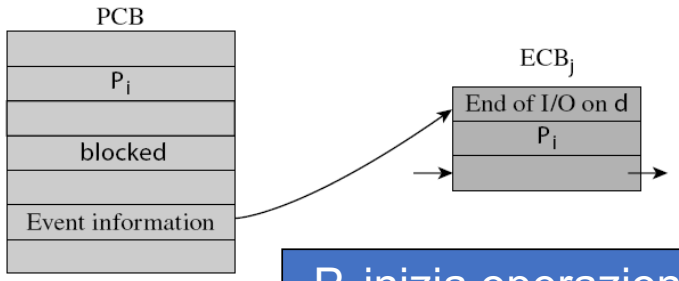
Gestione degli Eventi (cont.)

- Quando si verifica un evento, il kernel deve trovare il processo il cui stato è affetto dalla sua occorrenza
  - I SO usano vari schemi per velocizzare il tutto
    - Ad esempio, *Event Control Block* (ECB)



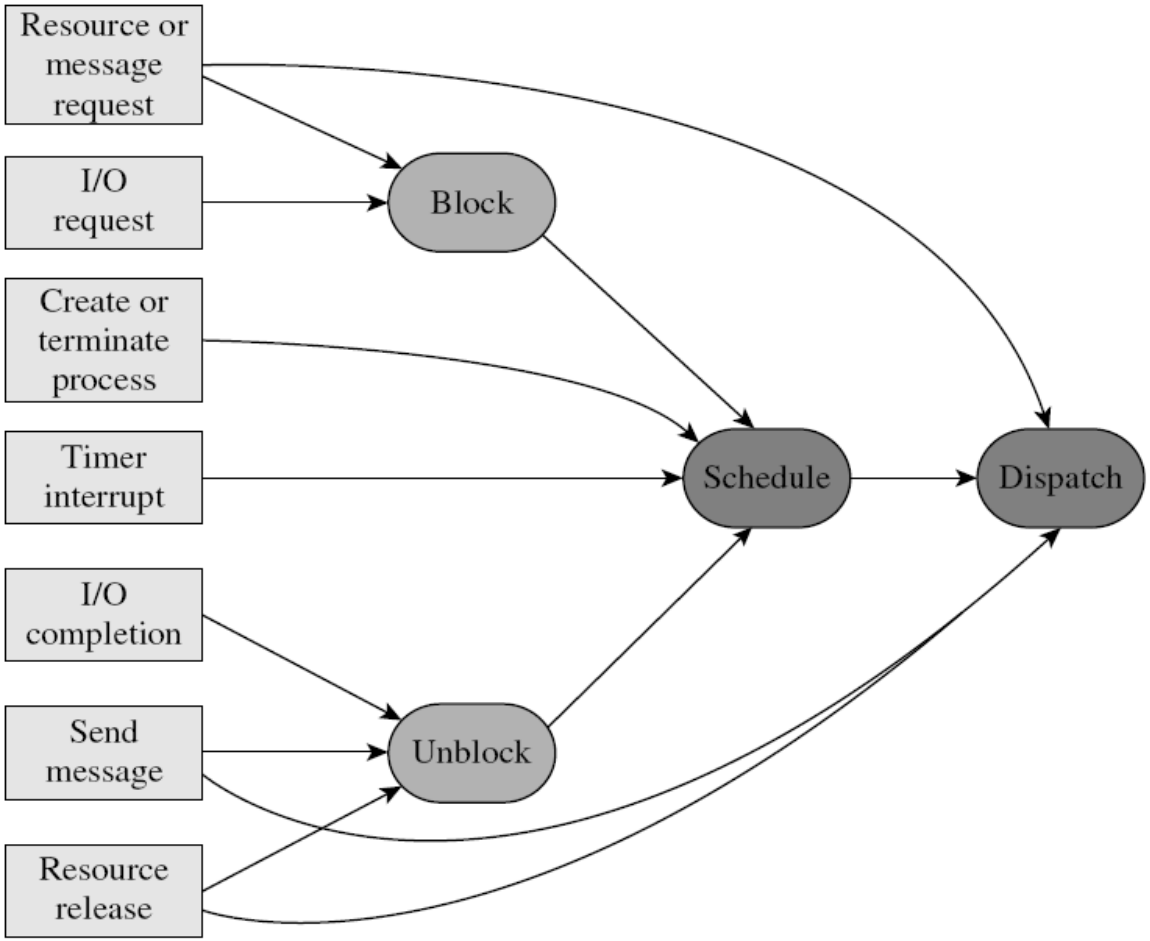
Esempio: ECB per il Completamento di un I/O

- Il processo P_i richiede un'operazione di I/O su una periferica d
  1. Il kernel crea un ECB e lo inizializza
     a. Descrizione evento:= fine I/O su periferica d
     b. Processo in attesa evento := P_i
  2. Il nuovo ECB, ECB_j, è aggiunto alla lista di ECB
  3. Lo stato di P_i è cambiato in *blocked* e l'indirizzo di ECB_j è messo nel campo *event information* del PCB di P_i
  4. Quando si verifica l'interrupt «fine I/O su d», ECB_j vine individuato nella lista degli ECB, cercando l'evento nel campo descrizione evento
  5. E' estratto l'id del processo P_i affetto dall'evento da ECB_j. Il PCB di P_i è localizzato ed il suo stato è modificato in *ready*



P_i inizia operazione di I/O su d

Azioni per la gestione di un evento del kernel



Condivisione, Comunicazione e Sincronizzazione tra Processi

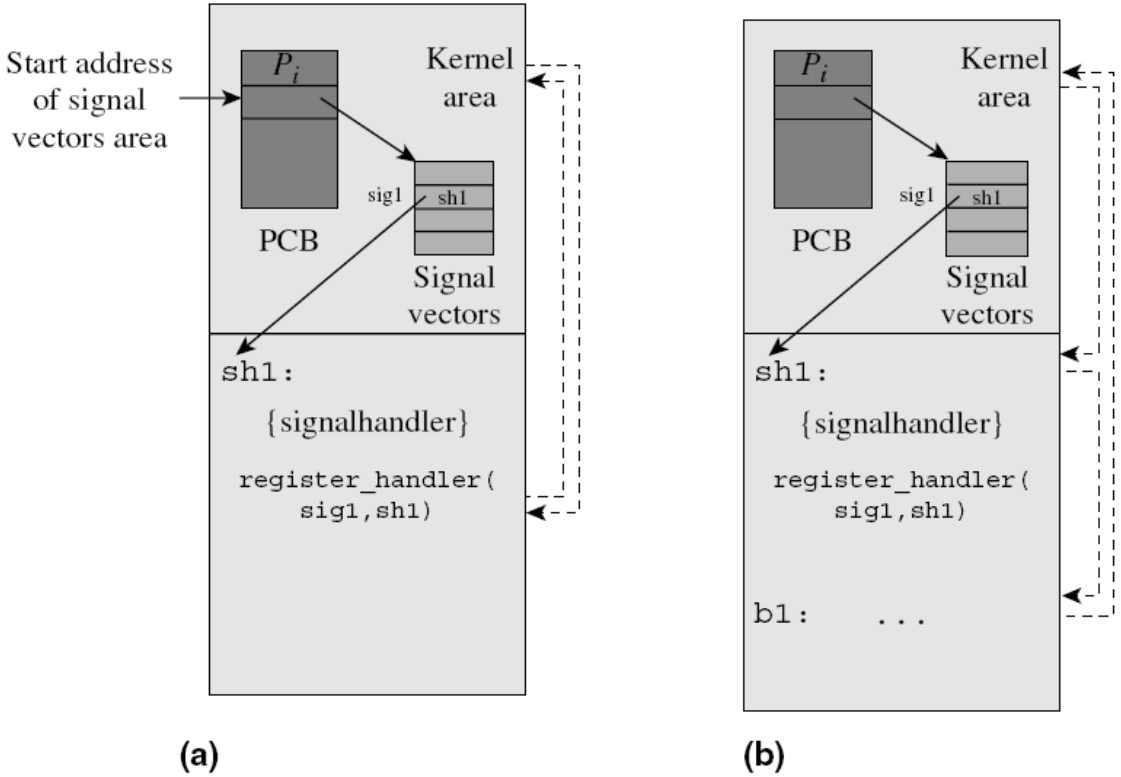
- Quattro tipi di interazione tra processi

Kind of interaction	Description
Data sharing	Shared data may become inconsistent if several processes modify the data at the same time. Hence processes must interact to decide when it is safe for a process to modify or use shared data.
Message passing	Processes exchange information by sending messages to one another.
Synchronization	To fulfill a common goal, processes must coordinate their activities and perform their actions in a desired order.
Signals	A signal is used to convey occurrence of an exceptional situation to a process.

Segnali

- Un segnale è usato per notificare una situazione eccezionale ad un processo e permettergli di occuparsene immediatamente
  - Situazioni e nomi/numeri dei segnali definiti nel SO
    - Condizioni della CPU come gli overflow
    - Condizioni legate ai processi figli
    - Utilizzo di risorse
    - Comunicazioni di emergenza da un utente ad un processo
- Possono essere sincroni o asincroni
- Gestiti da gestori di segnale (signal handler) definiti dal processo o da handler di default definiti dal SO

Esempio: gestione di un segnale



- Supponiamo che
  - il processo P_i sia prelazionato appena prima di eseguire b1
  - Successivamente, un processo P_j invia il segnale sig1 a P_i
    - signal(sig1, P_i)

Gestione segnale del processo Pi: (a) registrazione di un handler; (b) invocare un gestore