2 – Processi e Thread

Sommario

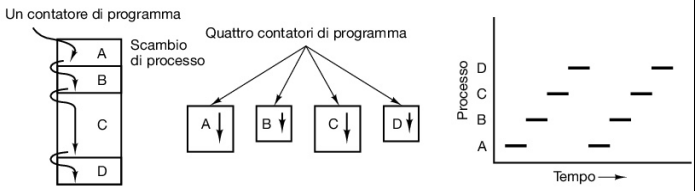
Scopi

- Introdurre al concetto di **processo**
  - ciclo di vita del processo
  - stati di processo e transizioni di stato.
  - blocchi di controllo di processo (PCB) / descrittori di processo.
- come avviene la transizione tra processori tramite cambio di contesto
- come gli interrupt hardware consentono di comunicare con il software
- come i processi dialogano tra loro attraverso la comunicazione tra processi (IPC)
- processi UNIX

Introduzione

- I sistemi eseguono operazioni concorrentemente
  - Ad esempio: compilazione di un programma, invio di un file a una stampante, rendering di una pagina Web, riproduzione di musica e ricezione di e-mail
  - I processi consentono ai sistemi per eseguire e tenere traccia delle attività simultanee
  - I processi cambiano **stato**
  - I sistemi operativi eseguono operazioni sui processi quali
    - creazione
    - distruzione
    - sospensione
    - ripresa
    - risveglio

Modello di processo



- Multiprogrammazione di quattro programmi
- Modello concettuale di quattro processi indipendenti sequenziali
- Un solo programma attivo in ogni momento

A. Tanenbaum – Modern Operating Systems

Modello di processo

- Un insieme di processi sequenziali
  - Programma in esecuzione
  - Registri
  - Variabili
- Ogni processo ha una CPU virtuale
- La CPU è unica e assegnata a turno ad un processo per volta
- Pseudo parallelismo
- Multiprogrammazione

Definizione di Processo

- Un programma in esecuzione
  - Un processo ha un proprio spazio di indirizzamento composto da:
    - Regione **testo**
      - Memorizza il codice che viene eseguito il processore
    - regione **dei dati**
      - Memorizza variabili e memoria allocata dinamicamente
    - Regione **stack**
      - Memorizza istruzioni e variabili locali per le chiamate di procedura attive

Gestione di Processi

- I sistemi operativi forniscono servizi fondamentali per i processi inclusi:
  - Creazione di processi
  - Distruzione di processi
  - Sospensione di processi
  - Ripresa di processi
  - Modifica della priorità di un processo
  - Blocco di processi
  - Risveglio di processi
  - Dispatching di processi
  - Interprocess communication (IPC)

Ai processi spesso si associa la priorità che viene modificata per lo scheduling

← Si interfaccia con SO per scegliere ordine di esecuzione

Creazione di Processi

- Quando
  - Inizializzazione del sistema
  - Dopo una chiamata di sistema di creazione di processo
  - Dopo una richiesta di utente di creazione di processo
  - Per servire attività di job in batch

Processi di sistema
- attivi - solitamente collegati ad utenti
- in background – non associati a utenti, ma con funzioni specifiche
  es. gestione stampe, gestione mail
  detti demoni

Daemon, demoni

Creazione di Processi

- Per creare un processo
  - In Unix ***fork*** ← comando per creare il thread o processo figlio
    - crea un clone del processo chiamante
    - i due processi condividono
      - l'immagine di memoria
      - ambiente
      - file aperti
    - poi solitamente il processo figlio esegue *execve* o altra chiamata per cambiare la propria immagine di memoria ed eseguire un nuovo programma
  - In Windows ***CreateProcess***
    - Una sola chiamata di funzione Win32 crea il processo e carica il programma del nuovo processo
    - Parametri (nome *P* da eseguire, relativi parametri, attributi di sicurezza, controllo file aperti, priorità,..)
    - Molte altre funzioni di gestione e sincronizzazione di processi

Creazione di Processi

- Dopo la creazione di un processo

  Genitore e figlio hanno spazi di indirizzi separati
  possono condividere delle risorse, es. file

  - In Unix il figlio ha una copia dello spazio del padre
    Spazi distinti
    Copy-on-write

  - In Windows il figlio ha dall'inizio uno spazio separato

Chiusura di Processi

- Quando
  - Uscita normale - volontaria
  - Uscita per un errore – volontaria
  - Errore critico – non volontaria
  - Uscita forzata da un altro processo – non volontaria

vol. vol. non vol. non vol. ← tipi di terminazione

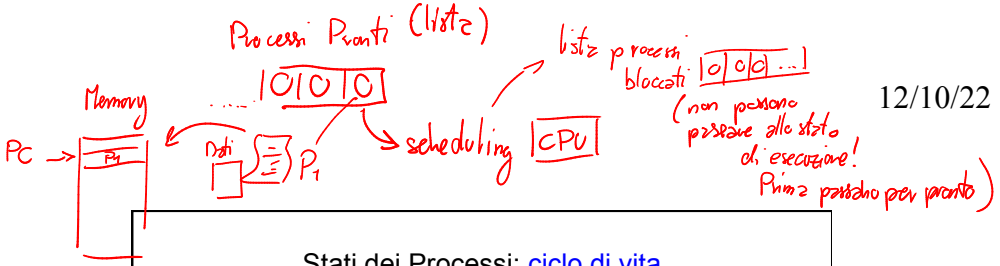
Terminazione normale:
- In Unix *exit* chiamata per la terminazione normale
- In Windows *ExitProcess*

Es. di errore critico: tentata esecuzione di una istruzione non valida, accesso a memoria errato, errore aritmetico,…

Terminazione forzata da un altro processo: un processo chiude un altro processo con kill();
- In Unix *kill* chiamata per la terminare un altro processo
- In Windows *TerminateProcess*

Creazione, chiusura e gerarchia di Processi

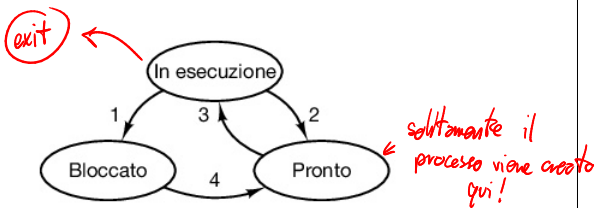
- Quando un processo genera (crea) un nuovo processo
  - Il processo che genera è chiamato processo padre
  - Il processo creato è chiamato il processo figlio
  - Esattamente un processo padre crea un figlio
  - Quando un processo padre viene distrutto, il S.O. può rispondere in uno di due modi:
    - Distrugge tutti i processi figli di quel genitore
    - Consentire ai processi figli di procedere indipendentemente dai loro genitori
- Processi padri e figli, a loro volta i figli possono creare altri processi
- Gerarchie
  - In Unix process group processo padre, i figli e discendenti
  - In Window non ha il concetto di gerarchie, tutti i processi sono creati uguali ma un processo padre ha un *handle* per controllare il figlio che può passare ad altri processi



Stati dei Processi: ciclo di vita

- Un processo si sposta attraverso una successione di stati discreti:
  - Stato *Running* (***In esecuzione***)
    - Il processo è in esecuzione su un processore
  - Stato *Ready* (***Pronto***)
    - Il processo potrebbe essere in esecuzione su un processore se ce ne fosse uno disponibile
  - Stato *Blocked* (***Bloccato***)
    - Il processo è in attesa di qualche evento che deve accadere prima che possa proseguire
- Il SO mantiene una **lista** *ready* e una **lista** *blocked* per mantenere i riferimenti ai processi non in esecuzione

Stati dei Processi: ciclo di vita



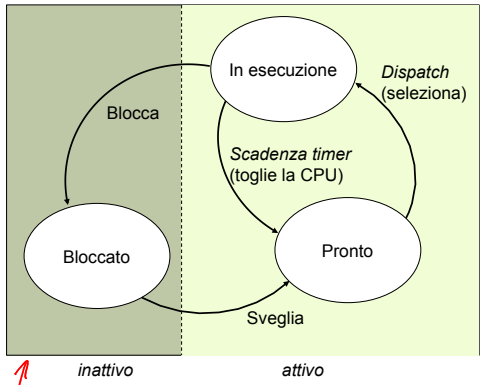
solitamente il processo viene creato qui!

1. Il processo si blocca in attesa di un evento esterno (es. di un input)
2. Lo scheduler seleziona un altro processo
3. Lo scheduler seleziona questo processo
4. L'evento esterno si verifica (es. l'input è pronto)

Stati di processo e transizione fra stati

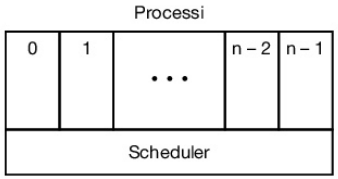
- Stati di processo
  - Dispatching: assegnare un processore per il primo processo della lista pronta
  - Il S.O. può utilizzare un timer a intervalli per consentire l'esecuzione di un processo per un dato intervallo di tempo o quanto
  - Il multitasking cooperativo consente l'esecuzione di ogni processo fino al completamento
- Transizione fra stati
  - Possibili casi
    - Se il processo è *dispatched*, transizione da *ready* a *running*
    - Se il quanto del processo termina, transizione da *running* a *ready*
    - Se il processo è bloccato, transizione da *running* a *blocked*
    - Se accade un evento, transizione da *blocked* a *ready*

Stati di processo e transizioni fra stati



un processo diventa inattivo e il processore passa ad un altro processore

Scheduler



Modello ideale di processo *scheduler* del sistema operativo e processi utente sequenziali

Process Control Blocks (PCBs)/Descrittori di Processo

!! Importante !!

- PCBs mantiene informazioni utili al OS per gestire il processo
  - Tipicamente include informazioni quali
    - Process identification number (PID)
    - Stato del processo (ready, running, blocked)
    - Contatore di programma (Program counter)
    - Puntatore allo Stack (Stack Pointer)
    - Priorità per lo Scheduling
    - Diritti
    - Puntatore al processo padre
    - Puntatore ai processi figli
    - Puntatore per localizzare i dati e istruzioni del processo in memoria
    - Puntatore alle risorse allocate.

Process Control Blocks (PCBs)/Descrittori di Processo

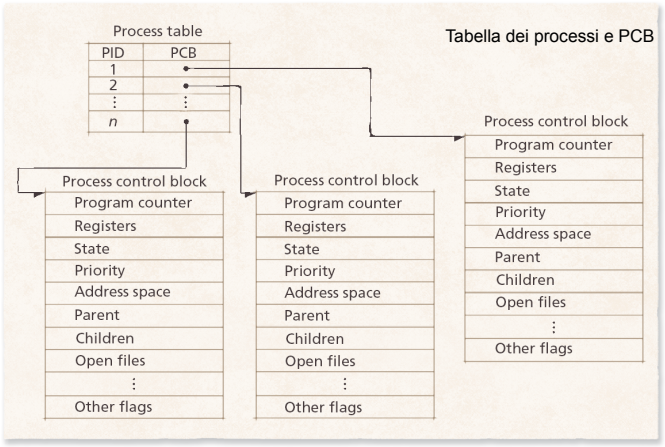
Gestione del processo	Gestione della memoria	Gestione dei file
Registri	Puntatore alle informazioni del segmento testo	Directory principale
Contatore di programma	Puntatore alle informazioni del segmento dati	Directory di lavoro
Parola di stato del programma	Puntatore alle informazioni del segmento stack	Descrittori dei file
Puntatore allo stack		ID utente
Stato del processo		ID del gruppo
Priorità		
Parametri di scheduling		
ID del processo		
Processo genitore		
Gruppo del processo		
Segnali		
Data e ora di avvio del processo		
Tempo di CPU usato		
Tempo di CPU dei figli		
Data e ora dell'allarme successivo		

Esempio di descrittore di processo

Process Control Blocks (PCBs)/Descrittori di Processo

- Tabella dei processi
  - Il SO mantiene i puntatori ai PCB di ogni processo in una tabella di sistema o una tabella di processo per utente
  - Permette un rapido accesso alle PCB
  - Quando un processo termina, il SO rimuove il processo dalla tabella dei processi e libera tutte le risorse dei processi

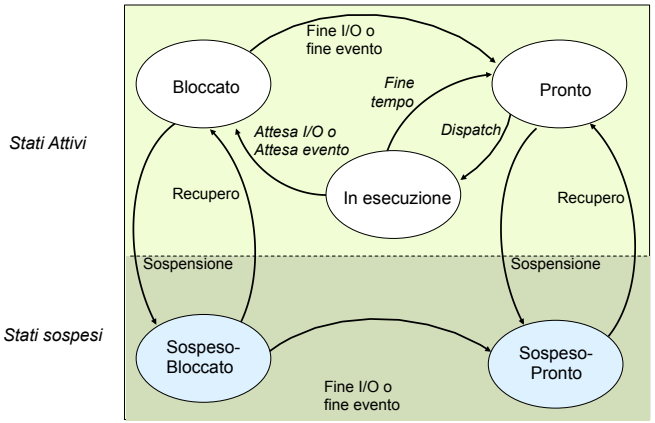
Process Control Blocks (PCBs)/Descrittori di Processo



Sospensione e recupero

- Sospendere un processo
  - Rimuove il processo dalla contesa per il tempo sul processore senza distruggerlo (e senza previsione sulla durata)
  - Utile per rilevare le minacce alla sicurezza e per debugging del software
  - Una sospensione può essere
    - richiesta dal processo stesso che viene sospeso o
    - richiesta da un altro processo
  - Un processo sospeso deve essere riavviato da un altro processo
  - Due stati sospesi:
    - *suspendedready*
    - *suspendedblocked*

Stati di processo e transizioni fra stati

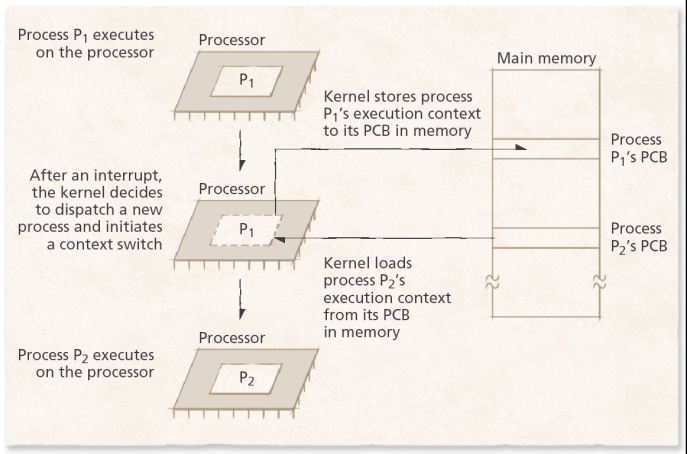


stati di sospensione

Cambio di contesto

- Cambio di contesto *(context switching)*
  - Eseguita dal S.O. per fermare l'esecuzione di un processo *running* e iniziare l'esecuzione di un processo *ready*
  - Salva il contesto di esecuzione del processo *running* nel suo PCB
  - Carica il contesto di esecuzione del processo *ready* dal PCB
  - Deve essere trasparente ai processi
  - Richiede al processore di non eseguire alcuna computazione "utile"
    - OS deve quindi ridurre al minimo il tempo-cambio di contesto
  - Eseguita in hardware per alcune architetture

Cambio di contesto



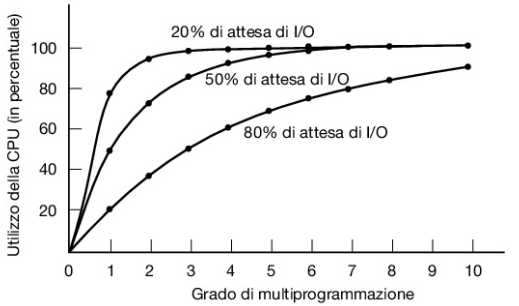
Multiprogrammazione

Valutazione delle prestazioni del S.O. in termini di utilizzazione della CPU in funzione del grado di multiprogrammazione

n processi in memoria

p frazione di tempo in cui un processo è in attesa di I/O (probabilità)

Assumendo indipendenza dei processi



Utilizzazione della CPU

$1 - p^n$

Nota:
modelli di valutazione di prestazioni più sofisticati dalla teoria delle code

Prob CPU occupata = 1 − Prob tutti i processi richiedono I/O

p^n ; $1 - p^n$

Interrupts

- Interrupts abilita il software a rispondere ai segnali hardware
  - Può essere iniziato da un processo in esecuzione
    - Interrupt viene chiamato *trap*
    - Sincrono con le operazioni del processo
    - Esempi: divisione per zero o riferimento ad area di memoria protetta
  - Possono essere avviate da qualche evento che può o può non essere correlato al processo in esecuzione
    - Asincrono con le operazioni del processo
    - Esempi: un tasto viene premuto su una tastiera o un mouse è spostato
  - Basso overhead
- Polling è un approccio alternativo
  - Il processore richiede ripetutamente lo stato di ogni dispositivo
  - Aumento dell'overhead al crescere della complessità del sistema

Elaborazione dell'Interrupt

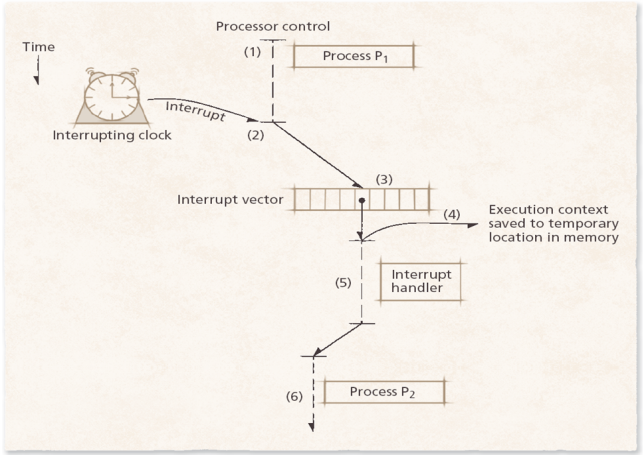
- **Gestione dell'interrupt**
  - Dopo aver ricevuto un interrupt, il processore completa l'esecuzione dell'istruzione corrente, poi interrompe il processo corrente
  - Il processore esegue una delle funzioni di interrupt di gestione del kernel
  - Il gestore di interrupt determina come il sistema dovrebbe rispondere
  - I gestori di interrupt sono memorizzati in un array di puntatori chiamato il **vettore di interrupt**
  - Dopo che il gestore di interrupt termina, viene ripristinato ed eseguito il processo interrotto o viene eseguito il processo successivo

Elaborazione dell'Interrupt

I passi del S.O. a basso livello per la gestione dell'interrupt

1. L'hardware mette nello Stack il PC (Program Counter), etc..
2. L'hardware carica il nuovo PC dal **vettore di interrupt**
3. La procedura in linguaggio assembly salva i registri
4. La procedura in linguaggio assembly imposta il nuovo Stack
5. Si esegue la procedura di servizio dell'interrupt in C
6. Si attiva lo Scheduler per selezionare il prossimo processo da eseguire
7. La procedura in C ritorna al codice in linguaggio assembly
8. La procedura in linguaggio assembly avvia il nuovo processo da eseguire

Gestione dell'Interrupt



Classi di Interrupt

- Gli interrupt supportati dipendono dalla architettura del sistema
  - La specifica dell'architettura Instel IA-32 distingue due tipi di segnali che possono essere ricevuti da un processore:
    - Interrupts
      - Notifica al processore che si è verificato un evento o che lo stato di dispositivo esterno è cambiato
      - Generata da dispositivi esterni ad un processore
    - Eccezioni
      - Indicare che si è verificato un errore, hardware o come risultato di un'istruzione software
      - Classificato come guasti (*faults*), *traps* or *abort*

Classi di Interrupt

Tipi comuni di interrupt riconosciute nell'architettura Intel IA-32

Tipo di Interruzione	Descrizione
I/O	Attivate dall'hw I/O, notificano al processore la variazione di stato di un canale o di una periferica. Es. fine di operazione di I/O
Timer	Periferiche che generano interrupt ad intervalli periodici, usati per gestire il tempo o per monitorare le prestazioni. Es. fine di un quanto di tempo di un processo
Interruzione per comunicazione interprocessor	Un processo può inviare messaggi ad altri processi. In sistemi multiprocessori i processi possono essere contemporaneamente in esecuzione.

Classi di Interrupt

Classi di eccezioni nell'architettura Intel IA-32

Classe di Eccezione	Descrizione
Fault (guasto)	Diverse possibili cause durante l'esecuzione di istruzioni di linguaggio macchina. Es. divisione per zero, dati utilizzati che sono in formato errato, tentativo di eseguire un codice di operazione non valido, tentativo di riferire una locazione di memoria in una zona proibita, tentativo da parte di un processo utente di eseguire istruzioni privilegiate o riferire risorse protette.
Trap	Generate da eccezioni quali overflow e quando il programma si trova in un breakpoint nel codice.
Abort	Un processore identifica in errore dal quel non si può recuperare l'esecuzione di un processo. Es. quando l'esecuzione di una routine che gestisce una eccezione a sua volta causi una eccezione, il processore potrebbe non riuscire a gestire entrambi gli errori sequenzialmente. Si parla di eccezione double-fault che porta alla terminazione del processo che l'ha attivata.

Interprocess Communication

- Molti SO forniscono meccanismi per la comunicazione tra processi (IPC)
  - I processi devono comunicare tra loro in ambienti multiprogrammati e in rete
    - Esempio: un Web browser recupera dati da un server remoto
  - Essenziale per i processi che devono coordinare le attività per raggiungere un obiettivo comune

Segnali

- Interruzioni software che notificano ad un processo l'occorrenza di un evento
  - Non permette ai processi di specificare dati da scambiare con altri processi
  - I processi possono ricevere, ignorare o mascherare un segnale
    - Ricevere un segnale comporta specificare una routine che il SO chiama quando manda il segnale
    - Ignorare un segnale si appoggia sull'azione di default del SO per gestire il segnale
    - Mascherare un segnale indica al SO di non consegnare segnali di quel tipo fino a quando il processo cancella quel mascheramento di segnale

Message Passing

- Interprocess communication basata su scambio di messaggi
  - I messaggi possono essere trasmessi in una direzione alla volta
    - Un processo è il mittente e l'altro è il ricevitore
  - Lo scambio di messaggi può essere bidirezionale
    - Ogni processo può agire sia come un mittente o ricevitore
  - I messaggi possono essere bloccanti o non bloccanti
    - Il blocco richiede al ricevente di notificare al mittente quando viene ricevuto il messaggio
    - Non bloccante permette al mittente di continuare altre elaborazioni
  - L' implementazione comune è **pipe**
    - Una regione di memoria protetta dal SO che funge da buffer, consentendo a due o più processi di scambio dati

Message Passing

- IPC in sistemi distribuiti
  - I messaggi trasmessi possono essere corrotti o persi
    - protocolli di conferma confermano che le trasmissioni siano stati correttamente ricevuti (acknowledgement)
    - Meccanismi di timeout con ritrasmissione messaggi se gli ack non vengono ricevuti
  - I processi denominanti in modo ambiguo portano ad errori di riferimento di messaggio
    - I messaggi sono passati tra sistemi che utilizzano porte numerate sulle quali i processi ascoltano, evitando questo problema
  - La sicurezza: problema
    - garantire l'autenticazione

Caso di studio: processi UNIX

- Processo UNIX
  - Tutti i processi hanno un insieme di indirizzi di memoria, chiamato spazio di indirizzi virtuali
  - Il PCB di un processo è mantenuto dal kernel in una regione protetta della memoria che i processi utente non possono accedere
  - Un PCB UNIX memorizza:
    - Il contenuto dei registri del processore
    - PID
    - Il contatore di programma
    - Lo stack di sistema
  - Tutti i processi sono elencati nella tabella dei processi

Caso di studio: processi UNIX

- Processo UNIX
  - Tutti i processi interagiscono con il SO con chiamate di sistema
  - Un processo può generare un processo figlio usando la chiamata di sistema fork, che crea una copia del processo padre
    - Il processo figlio riceve una copia delle risorse del genitore (immagini di memoria distinte)
    - I file aperti dal processo padre sono condivisi con il figlio
    - La fork restituisce 0 al figlio e il PID del figlio al genitore
  - le priorità dei processi sono numeri interi in [-20, 19]
    - Un valore di priorità numerica inferiore indica una priorità più alta di scheduling
  - UNIX fornisce meccanismi IPC, come pipes, per consentire a processi diversi di trasferire i dati

Caso di studio: processo UNIX

Chiamate di sistema UNIX

Chiamata di sistema	Descrizione
`fork`	Crea un processo figlio e alloca una copia delle risorse del processo padre al figlio
`exec`	Carica da un file le istruzioni e dati di un processo nel suo spazio di indirizzamento
`wait`	Il processo chiamante si blocca fino a quando il processo figlio non ha terminato
`signal`	Permette ad un processo di specificare un gestore di segnalazione per un dato tipo di segnale
`exit`	Termina il processo chiamante
`nice`	Modifica la priorità del processo usata dallo scheduling