Lezione 4

Gestione dei processi

Sistemi operativi

27 marzo 2012

Marco Cesati

System Programming Research Group
Università degli Studi di Roma Tor Vergata

Di cosa parliamo in questa lezione?

Cosa è un processo e come viene gestito dal SO

1. Processi e job
2. Il blocco di controllo
3. Struttura di un file eseguibile
4. La schedulazione dei processi
5. Il job scheduler
6. Lo scheduler a breve termine
7. Lo scheduler a medio termine
8. Contesto e cambio di contesto

Definizione di processo

Il *processo* (in inglese *process* o *task*) è una astrazione costruita dal sistema operativo per rappresentare un programma in esecuzione

Un processo è costituito da un insieme di strutture di dati che

- descrivono completamente lo stato d'avanzamento del programma
- consentono di salvare informazioni per gestire la multiprogrammazione
- legano il processo ad un job, ad un utente o gruppo di utenti

Job

Un job è un insieme di processi che cooperano per svolgere un determinato compito

- Il termine job deriva dai sistemi operativi a lotti
- Nei SO Unix il termine job è usato per indicare un gruppo di processi lanciati con lo stesso comando:

```
$ ls | more
$ (date; make; date) &
$ (./eser && echo 'OK')
```


Stati di un processo

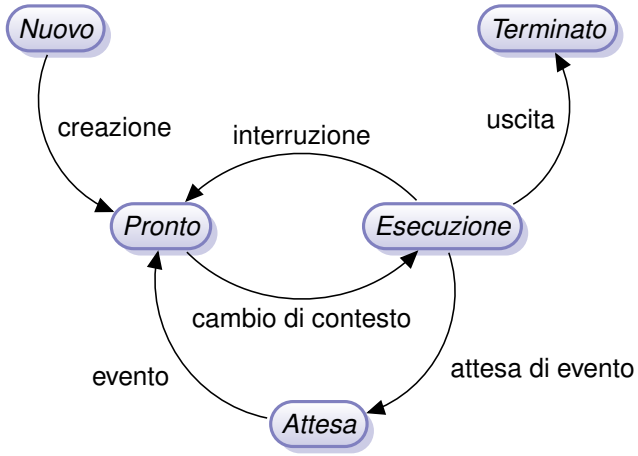
Ciascun processo è caratterizzato sempre da uno stato d'esecuzione

Un generico SO potrebbe utilizzare i seguenti stati di un processo:

- Nuovo: in fase di creazione (inizializzazione)
- Esecuzione: eseguito da una CPU
- Attesa: non eseguibile in quanto in attesa di un evento esterno
- Pronto: eseguibile ma non è attualmente in esecuzione su alcuna CPU
- Terminato: terminato, in fase di cancellazione

Questo elenco di stati è solo un esempio, un SO reale può definire altri stati d'esecuzione o fare a meno di qualcuno

Diagramma base di transizione tra gli stati



Esempio: gli stati dei processi in Linux

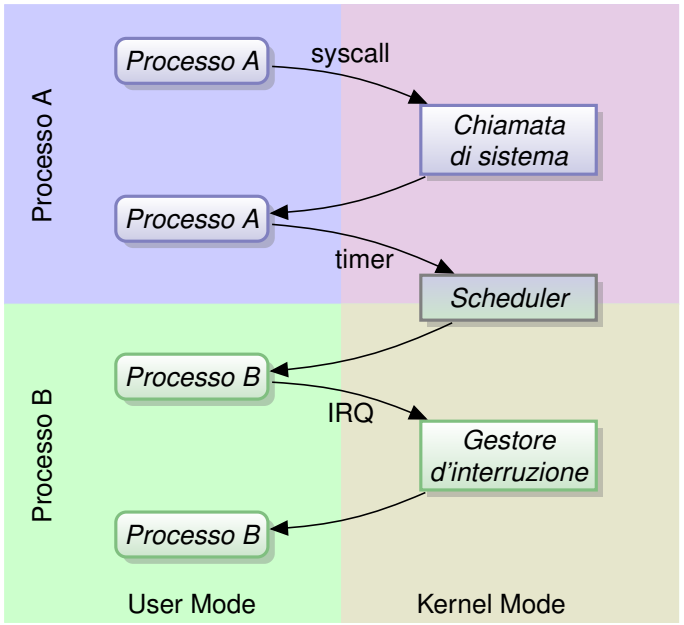
Il nucleo del sistema operativo Linux definisce tra gli altri i seguenti stati di esecuzione dei processi:

- `TASK_RUNNING`: eseguibile (in esecuzione o pronto)
- `TASK_INTERRUPTIBLE`: in attesa di un evento di I/O, e può essere interrotto (ucciso)
- `TASK_UNINTERRUPTIBLE`: in attesa di un evento di I/O, e non può essere interrotto
- `TASK_STOPPED`: bloccato da utente o shell di comandi
- `EXIT_ZOMBIE`: terminato, dati conservati per genitore
- `EXIT_DEAD`: terminato ed in cancellazione

Processi e modalità d'esecuzione

- In ogni istante su ciascuna CPU è eseguito esattamente un processo
  - Se non vi è nulla da fare, la CPU esegue un processo di sistema chiamato *idle task*
- Per implementare meccanismi di sicurezza e protezione per i processi la CPU esegue i programmi in modalità Kernel Mode (privilegiata) o User Mode (non privilegiata)
- Un generico processo alterna fasi di esecuzione in User Mode e fasi di esecuzione in Kernel Mode
  - La suddivisione tra i processi è indipendente dalla modalità d'esecuzione della CPU (UM e KM)
- Eccezione: alcuni processi di sistema (in Linux: *kernel thread*) vengono eseguiti esclusivamente in Kernel Mode
- Non è invece possibile creare processi eseguiti esclusivamente in User Mode

Processi e modalità d'esecuzione (2)



Blocco di controllo

- Il *blocco di controllo* è l'insieme delle strutture di dati del nucleo associate ad uno specifico processo
  - Process Control Block (PCB) o Task Control Block (TCB)
- Il puntatore (indirizzo) al PCB è spesso l'identificatore o descrittore del processo entro il nucleo
- Nelle tabelle del PCB è memorizzato l'intero processo:
  - Stato del processo
  - Contatore di programma (indirizzo della successiva istruzione macchina da eseguire)
  - Valore dei registri della CPU
  - Parametri di schedulazione del processo
  - Informazioni per la gestione della memoria del processo
  - Informazioni per la contabilizzazione delle risorse
  - Informazioni su file aperti e stato delle operazioni di I/O

  ⋮

Esempio: il blocco di controllo in Linux

- L'header file `include/linux/sched.h` contiene la definizione della struttura base del PCB:

```
struct task_struct {
        volatile long state;
        void *stack;
        atomic_t usage;
        unsigned int flags;
        unsigned int ptrace;
        [...]
};
```

- Un generico processo è identificato nel nucleo dal puntatore alla struttura `task_struct`:

```
struct task_struct *task;
```

- La macro `current` espande nell'indirizzo del PCB del processo attualmente in esecuzione sulla CPU
  - In un sistema multiprocessore `current` restituisce un diverso PCB per ogni CPU

Struttura di un file eseguibile

Un file eseguibile è organizzato al suo interno in sezioni e segmenti

- Le sezioni sono porzioni del file omogenee per contenuto
  - codice
  - dati inizializzati
  - simboli
  - ...
- I segmenti sono gruppi di sezioni che possono essere trattate in modo identico in fase di caricamento
  - codice eseguibile e variabili di sola lettura
  - variabili globali inizializzate e variabili automatiche statiche
  - ...

Si noti che variabili automatiche e memoria allocata dinamicamente non sono rappresentate entro il file eseguibile

Sezioni ELF

Il formato di file eseguibile ELF (Executable and Linkable Format) è utilizzato in tutti i principali SO della famiglia Unix

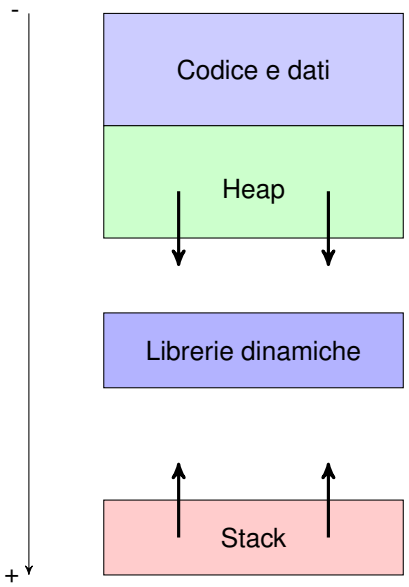
Le principali sezioni del formato ELF:

- `.text`: il codice eseguibile del programma
- `.data`: i dati persistenti del programma
  - variabili globali non inizializzate
  - variabili globali inizializzate a non zero
  - variabili automatiche statiche non inizializzate
  - variabili automatiche statiche inizializzate a non zero
- `.bss` (nome dall'istruzione Block Started by Symbol, IBM 704, ∼1955):
  - variabili globali inizializzate a zero
  - variabili statiche inizializzate a zero
- `.rodata`: i dati di sola lettura
  - costanti
  - stringhe di caratteri

Heap e stack di un processo

- Lo *heap* di un processo è una zona dello spazio di memoria del processo dedicata alle strutture di dati allocate dinamicamente
  - `malloc()`, `calloc()`, `new`, ...
- Lo *stack* di un processo è una zona dello spazio di memoria dedicata a
  - variabili automatiche delle funzioni
  - indirizzi di ritorno delle funzioni
  - area temporanea gestita dal compilatore
- Un processo può anche fare uso di uno stack Kernel Mode utilizzato durante l'esecuzione delle chiamate di sistema e dei gestori di interruzione
  - Generalmente di lunghezza fissa e limitata
  - Contenuto nello spazio di memoria dedicato al nucleo

Tipica struttura della memoria di un processo



Code di processi

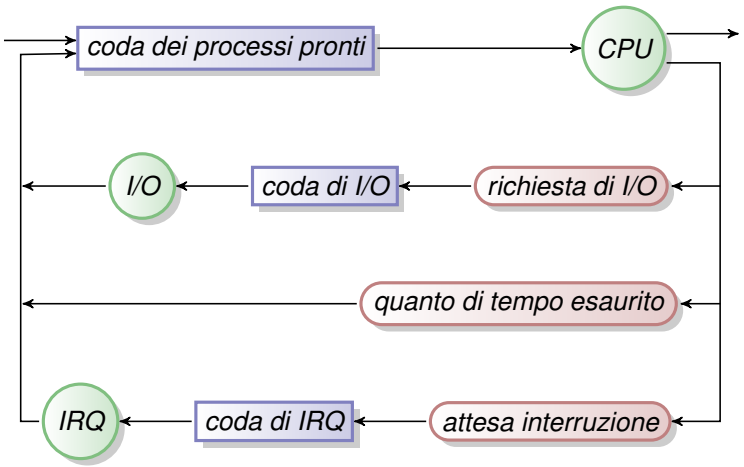
I processi del sistema sono inseriti in diverse strutture di dati dinamiche del nucleo, tipicamente code d'attesa

Ad esempio:

- Coda dei processi pronti (ready queue): contiene tutti i processi che possono essere immediatamente eseguiti ma non sono assegnati ad una CPU
- Coda di dispositivo di I/O (I/O queue): ciascun dispositivo è associato ad una coda di processi in attesa che si completino operazioni di I/O
- Coda di primitiva di sincronizzazione: ad esempio, un semaforo è associato con una coda contenente i processi in attesa che la risorsa diventi disponibile

Una coda d'attesa è in genere una lista di PCB:

Diagramma di accodamento



Schedulazione dei processi

Lo *schedulatore* (*scheduler*) dei processi è il componente del SO che determina quali programmi debbono essere eseguiti in ogni istante dalle CPU

Si può suddividere in tre tipi di scheduler:

- scheduler a lungo termine (o job scheduler)
- scheduler a breve termine
- scheduler a medio termine

Le differenze principali tra i vari tipi di scheduler:

- frequenza di invocazione
- obiettivi dello schedulazione

Il job scheduler

Lo scheduler a lungo termine o job scheduler ha origine nei sistemi a lotti:

- seleziona tra un insieme di programmi eseguibili in memoria secondaria quelli da caricare in memoria centrale
- mantiene sotto controllo il livello di multiprogrammazione del sistema
- è tipicamente invocato quando un processo termina
- deve mantenere bilanciato il carico del sistema con un buon mix di processi:
  - CPU-bound (a prevalenza d'elaborazione): molti calcoli, poche operazioni di I/O
  - I/O-bound (a prevalenza di I/O): pochi calcoli, molte operazioni di I/O

Caratterizzare "a priori" un programma come CPU-bound o I/O-bound è difficile!

Scheduler a breve termine

Lo scheduler a breve termine (o semplicemente scheduler) ha origine con i sistemi a partizione di tempo (time sharing):

- seleziona il processo da eseguire tra quelli caricati in memoria e presenti nella coda dei processi pronti
- invocato con un frequenza tipica tra 5 Hz e 20 Hz
  - tipicamente lo scheduler deve essere invocato diverse volte in un secondo per poter dare all'utente l'illusione che tutti i processi siano eseguiti in parallelo
  - la frequenza di invocazione è legata alla durata massima in cui un processo può restare in esecuzione su una CPU senza essere sostituito
- maggiore è la frequenza di invocazione, più rapida deve essere l'esecuzione dello scheduler
  - uno scheduler con frequenza di 10 Hz che impiega 10 msec per terminare consuma il 10% del tempo della CPU

Scheduler a breve termine (2)

Le scelte operate dallo scheduler sono essenziali per le prestazioni dell'intero sistema di calcolo

Tra gli obiettivi da perseguire:

- ottenere tempi di risposta rapidi per i processi interattivi
- garantire tempi di esecuzione ragionevoli per i processi non interattivi
- assicurare che nessun processo sia escluso dalla CPU
- conciliare le esigenze dei processi con bassa e alta priorità

Una classificazione dei processi alternativa a CPU-bound / I/O-bound (migliore per gli scheduler dei moderni SO):

- Processi interattivi: interagiscono costantemente con l'utente, debbono avere tempi di risposta minimi
- Processi batch: job o processi lanciati in background, senza terminale di controllo
- Processi real-time: hanno scadenze temporali stringenti e richiedono estrema predicibilità di comportamento

Schedulatore a medio termine

Lo scheduler a medio termine è presente tipicamente in sistemi a partizione di tempo

- elimina processi dalla memoria centrale per
  - ridurre il livello di multiprogrammazione
  - liberare pagine di memoria primaria per i processi restanti
- lo schema si chiama avvicendamento dei processi (swapping)
  - in alcuni SO lo scheduler a medio termine elimina un intero processo dalla memoria primaria salvandolo in quella secondaria
  - nei SO moderni con paginazione della memoria è possibile eliminare una frazione di un processo (le pagine non immediatamente necessarie per l'esecuzione)

Esempio: lo scheduler di Linux

Nel SO Linux:

- il job scheduler è assente
  - non esiste un meccanismo di ammissione dei programmi in memoria
  - è possibile creare nel sistema un numero eccessivo di processi
  - gli utenti si auto-regolano!
- lo scheduler di medio termine lavora a livello di singola pagina di memoria:
  - un processo di sistema esamina tutte le pagine di memoria alla ricerca di quelle che possono essere "swappate" su disco
  - le pagine di memoria swappate su disco vengono caricate automaticamente quando vengono accedute

Lo scheduler utilizzato in MS Windows ed altri SO moderni è sostanzialmente analogo

Ruolo delle interruzioni hardware

Nelle architetture moderne le interruzioni hardware svolgono un ruolo essenziale

- Sono generate da quasi tutti i dispositivi hardware
- Possono essere generate periodicamente da circuiti timer
- Vengono utilizzate dai SO per
  - pilotare i dispositivi hardware
  - misurare il trascorrere del tempo
  - implementare la partizione del tempo d'elaborazione

Ogni occorrenza di una interruzione provoca generalmente

- Il passaggio della modalità d'esecuzione a Kernel Mode
- Il salto ad una procedura di gestione dedicata

Affinché sia possibile ripristinare la procedura interrotta, il gestore dell'interruzione deve per prima cosa salvare lo stato della CPU (tipicamente i registri)

Contesto di esecuzione

Si definisce contesto di un processo l'insieme delle informazioni che debbono essere salvate in memoria dal SO all'occorrenza di un cambio di processo

Fanno tipicamente parte del contesto:

- Il contenuto dei registri della CPU
- Lo stato del processo
- L'indirizzo delle tabelle di paginazione del processo

Il contesto di un processo non coincide con il PCB:

- Il contesto di un processo è costituito dallo stato della CPU nel momento in cui si è verificata l'interruzione
- Il contesto viene solitamente salvato sullo stack Kernel Mode e/o in opportuni campi del PCB
- Le altre informazioni nel PCB sono già in memoria e non è necessario salvarle

Cambio di contesto

1. Il processo A è in esecuzione su una CPU
2. Si verifica una interruzione hardware (IRQ)
3. Il gestore dell'interruzione salva il contenuto di alcuni registri della CPU sullo stack KM del processo A
4. Viene eseguita la ISR (Interrupt Service Routine) dell'interruzione
5. Prima di ripristinare l'esecuzione del processo A interrotto si controlla se A deve essere sostituito
6. In questo caso, si invoca lo scheduler a breve termine per scegliere un nuovo processo B da eseguire sulla CPU
7. Si salva la porzione del contesto del processo A che non era stata salvata sullo stack (tipicamente nel PCB)
8. Si ripristina la stessa porzione di contesto dal PCB del processo B
9. Si ripristina il contenuto dei registri della CPU salvati sullo stack KM del processo B
10. Si conclude l'esecuzione del gestore dell'interruzione
11. Si riprende l'esecuzione del processo B