1 Lezione del 21-03-25

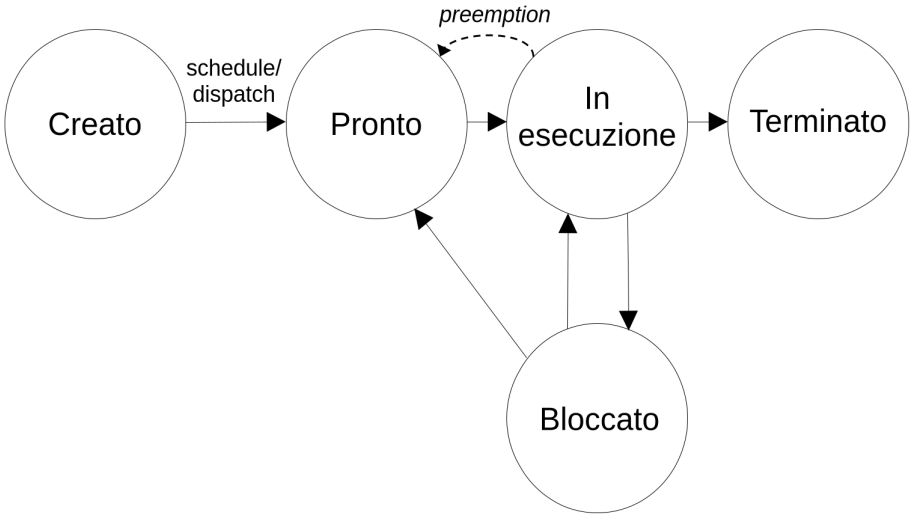
Andiamo a definire più nei dettagli la struttura di un processo e le modalità secondo le quali questi si possono creare e distruggere.

1.1 Descrittori di processo

Un processo è descritto fondalmente da un astrazione, detta **descrittore di processo**, idealmente contenuta in una qualche locazione contigua, assieme ad altri descrittori, in memoria.

- Il primo campo del descrittore sarà un **indice** numerico unico ad ogni processo;
- Dovremo poi tenere conto del **contesto** del processo, inteso come la copia di tutti i registri del processore.

Ogni processo ha un ciclo di vita che rispetta l'andamento del seguente grafico:



Vediamo nel dettaglio il significato delle diverse fasi. In fase di creazione del proesso, il suo descrittore viene posto in una struttura dati che ne consente **schedulazione** e **dispatch**:

- **Schedulazione:** effettivamente la scelta che il kernel fa, assunto il controllo, su qual'è il prossimo processo da portare in esecuzione (passaggio da processo **pronto** a processo in **esecuzione**);
- **Dispatch:** l'esecuzione effettiva di una serie di operazioni di tale processo.

I processi possono anche **bloccarsi**, cioè mettersi in attesa di qualche evento.

Infine, un processo può **terminare**, cioè sparire dal sistema (lui e il suo descrittore). Anche in questo caso il processo deve essere attualmente in esecuzione.

Una transizione che non è prevista da tutti i sistemi è quella di **preemption**, cioè di ritorno allo stato **pronto** a controllo dello scheduler. La maggior parte dei sistemi operativi supporta tale funzionalità, il nucleo che vedremo solo parzialmente.

1.2 Prima vista dell'esecuzione del kernel

Dopo il boot della macchina, il kernel si impadronisce della macchina e lancia il primo processo (il processo utente). Da qui in poi il kernel avrà il controllo solo fra un processo e l'altro, in caso di interruzioni (interne, esterne o eccezioni), e potrà restituirlo solo attraverso il ritorno da gestore con `IRETQ`.

Come abbiamo visto, ad ogni chiamata di gestore di interruzione lascia RIP, CS, RFLAGS e RSP al tempo di chiamata dell'interruzione (facendo le opportune distinzioni fra *fault* e *trap*) in pila. A questo punto il gestore fa una copia dei registri generali, e si ha a quel punto una *"foto"* del processore al momento di attraversamento del gate, che rappresenterà quindi il *contesto* del processo stesso al momento della chiamata dell'interruzione.

In questo, sfrutteremo delle routine (`salva_stato` e `carica_stato`) all'avvio e al termine di ogni gestore, che si occupano di salvare e caricare il contesto del processo attualmente in esecuzione. Per conoscere quale questo processo sia, si mantiene una variabile globale nel sistema, `esecuzione`, che punta al descrittore del processo (che è dove vogliamo mettere il contesto stesso).

I processi pronti staranno in una certa struttura dati (nel nucleo una linked list), ordinata per la **priorità** (un altro valore che manteniamo nel descrittore di processo) di ogni processo. Un gestore di interruzione di base, quindi, si potrebbe magari occupare di passare al contesto e all'esecuzione del processo di priorità più alta a intervalli regolari, magari regolato da un timer (cosiddetto *timeslicing*).

Altre situazioni, più vicine a noi, sono quelle del termine di una gestione di un interruzione esterna, o bloccaggio automatico di un processo, dove il kernel deve selezionare il prossimo processo da eseguire, scegliendo chiaramente quello a priorità più alta.

Inseriamo un processo fittizio, *dummy*, nella lista dei processi pronti con la priorità più bassa possibile. Questo ci assicurerà di non trovarci mai una situazione dove nessun processo è pronto all'esecuzione, e quindi avere sempre qualcosa a cui il kernel può passare (idealmente il processo dummy effettua solo un ciclo a vuoto).

Un ulteriore dettaglio è quello dello stato del processo alla sua creazione. Non è infatti realistico pensare di controllare se quel processo richiede inizializzazione ogni volta che si ritorna da un interruzione gestita a livello sistema. Alla creazione del processo, quindi, vogliamo svolgere le seguenti azioni in modo che il processo venga eseguito per la prima volta già in uno stato completo:

- Allocare una **pila sistema** dedicata al processo;
- Inizializzare la pila sistema. Questo consisterà nell'inizializzare a loro volta:
  - IP alla prima istruzione del processo;
  - CS al segmento livello utente dove si trova il processo;
  - FLAG a quanto viene richiesto dallo standard C++ al momento di avvio (solitamente tutto a 0), con l'eccezione di IF a 1.
- Allocare il **descrittore** di processo, e mettere quel processo fra i processi pronti;
- Inizializzare il descrittore. Questo consiste nell'inizializzare a loro volta:
  - Un puntatore alla pila sistema appena definita;
  - Il contesto del processo;
  - L'**argomento** di chiamata del processo, utile al debug;

– L'**IOPL**, *IO Privilege Level*, che specifica la possibilità o meno del processo di accedere all'IO.