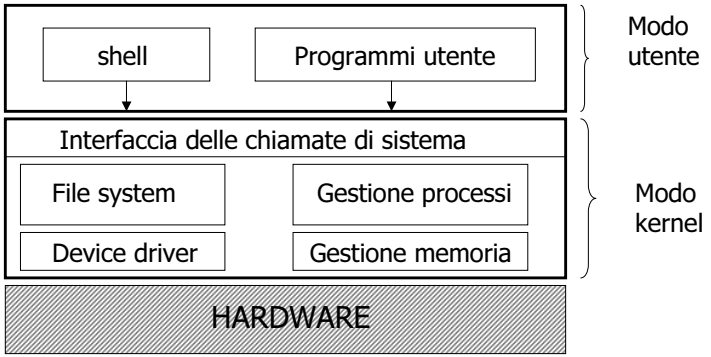
Struttura interna del sistema operativo Linux

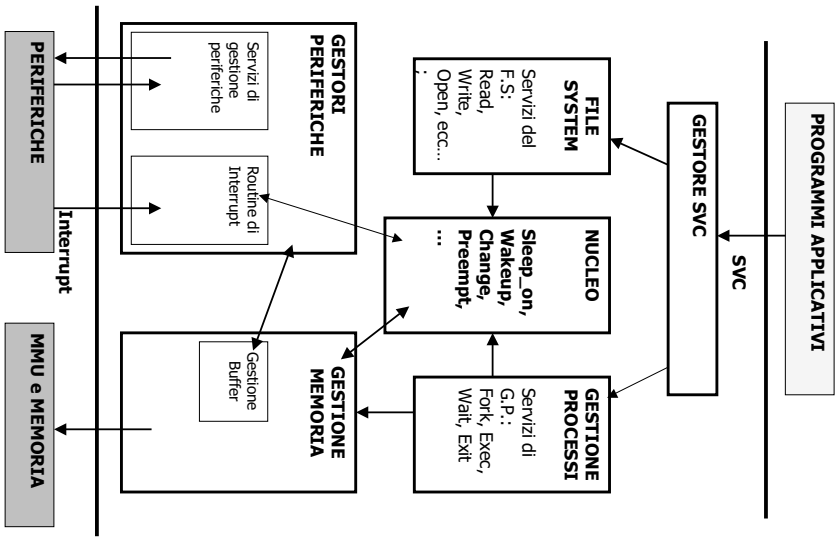
2. Nucleo del sistema operativo

A cura di:
Anna Antola Giuseppe Pozzi
DEI, PoliMI, Milano
anna.antola/giuseppe.pozzi@polimi.it
- versione del 10 marzo 2003 -

Architettura del sistema operativo



Struttura a strati di LINUX



Funzionalità del Nucleo

- Mantenere in memoria il S.O. e i diversi programmi in spazi di indirizzamento separati. Durante l'esecuzione di un programma, accedere solo allo spazio di indirizzamento consentito.
- Permettere il passaggio dall'esecuzione di un programma a quella del S.O. e viceversa.
- Gestire le operazioni di I/O solo da S.O., e quindi tramite chiamate di sistema.
- Gestire i meccanismi di interrupt.
- Gestire la verifica del quanto di tempo associato ai processi.

Che cosa è un processo per il S.O.

- Rappresenta una istanza del programma in esecuzione con tutte le informazioni necessarie:
  - codice eseguibile;
  - dati del programma;
  - spazio di indirizzamento che contiene il codice eseguibile, i dati e lo stack;
  - informazioni relative al suo funzionamento (stato).

Meccanismo base di funzionamento del nucleo (1)

Il S.O. alloca una zona di memoria ad ogni processo che viene creato e lo carica in tale zona.

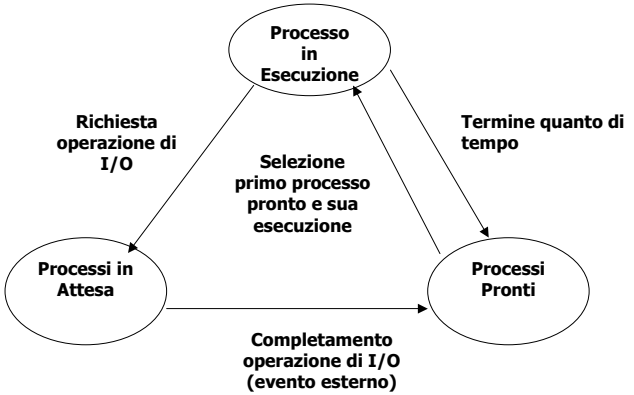
Esecuzione di un processo:

- il nucleo del S.O. sceglie, secondo politiche opportune, un processo da mandare in esecuzione (**stato di esecuzione**);
- quando un processo in esecuzione richiede un servizio di sistema tramite una SVC:
  - viene attivata la funzione relativa e il S.O. esegue il servizio nel **contesto del processo** stesso ;
  - i servizi forniti dal S.O. operativo sono quindi parametrici in quanto vengono svolti *per conto del processo che li ha attivati*.

Meccanismo base di funzionamento del nucleo (2)

- Un processo in stato di esecuzione lo abbandona:
  - per *sospensione volontaria*. Se il servizio di sistema richiesto deve attendere il verificarsi di un evento per completare il servizio stesso, il processo passa da stato di esecuzione a **stato di attesa**. Si noti che un processo passa in stato di attesa quando è in esecuzione in modo S;
  - per la scadenza del quanto di tempo. Quando il S.O. verifica che il quanto di tempo è scaduto, pone il processo in esecuzione in **stato di pronto** (*preemption*).
- Quando un processo in esecuzione è passato in stato di attesa o di pronto, il S.O. seleziona un processo, tra quelli pronti e lo porta in stato di esecuzione (**commutazione di contesto**).

Diagramma degli stati (3) di un processo e relative transizioni



Stati di un processo

- **In esecuzione**: è il processo che utilizza il processore (esiste un solo processo in esecuzione).
- **Pronto**: un processo è in stato di pronto se attende solo la risorsa processore per poter proseguire nell'elaborazione (possono esistere più processi in stato di pronto).
- **Attesa**: un processo è in stato di attesa se attende una risorsa non disponibile, un evento, il completamento di una operazione di I/O (possono esistere più processi in attesa su "eventi" diversi).

Transizioni di stato

ESECUZIONE ➔ PRONTO

- Al termine del quanto di tempo il SO deve salvare tutte le informazioni necessarie (contesto) per poter riprendere l'esecuzione del processo dal punto in cui è stata interrotta.

ESECUZIONE ➔ ATTESA

- Si verifica quando il processo richiede delle risorse che non sono disponibili o attende un evento;
- il SO salva tutte le informazioni necessarie (contesto) a riprendere l'esecuzione e l'informazione relativa all'evento atteso.

Transizioni di stato

ATTESA ➔ PRONTO

- Quando l'evento atteso da un processo si verifica, il SO sposta tutti i processi in attesa di quell'evento o di quella risorsa nella coda dei processi pronti.

PRONTO ➔ ESECUZIONE

- Il SO stabilisce quale dei processi accodati nello stato di PRONTO debba essere mandato in esecuzione (caricamento del contesto).
- La scelta è effettuata dall'algoritmo di scheduling.

Meccanismo base di funzionamento del nucleo (3)

La gestione degli **interrupt**

- Quando si verifica un'interruzione esiste un processo in esecuzione:
  - l'interrupt può interrompere un processo in modalità U;
  - l'interrupt può interrompere un servizio di sistema invocato dal processo in esecuzione;
  - l'interrupt può interrompere una routine di interrupt.

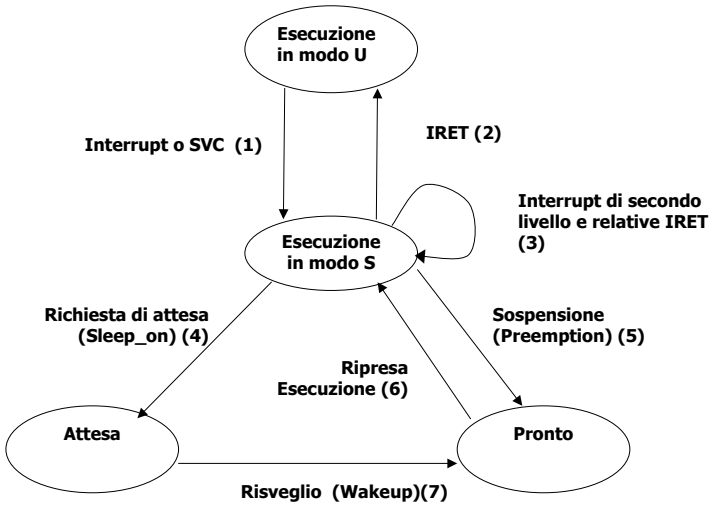
Meccanismo base di funzionamento del nucleo (4)

- In tutti i casi la routine di interrupt (cioè il S.O.) svolge il proprio compito in modo trasparente rispetto al processo in esecuzione e cioè **non** viene svolta una commutazione di contesto durante l'esecuzione di un interrupt (gli interrupt sono svolti nel contesto del processo in esecuzione).
- Se la routine di interrupt è associata al verificarsi di un certo evento sul quale erano in attesa dei processi, allora la routine porta i processi in attesa in stato di pronto.

Modalità user e modalità kernel

- I processi possono essere eseguiti in modalità kernel o user:
  - è necessario sdoppiare lo stato di esecuzione in due stati.
- Transizione di stato da esecuzione in modalità user a modalità kernel avviene in due casi:
  - chiamata di sistema (SVC);
  - interrupt.
- La transizione di stato da esecuzione in modalità kernel a modalità user avviene quando:
  - termina l'esecuzione della routine di risposta all'interrupt (se non ci sono interrupt annidati) o si rientra da SVC.

Transizioni di stato



La gestione del quanto di tempo

- Il **quanto di tempo** è gestito da un particolare **interrupt** generato dall'orologio di sistema.
- Ad una frequenza definita, il dispositivo che realizza l'orologio di sistema genera un interrupt. La routine di risposta relativa incrementa una opportuna variabile che contiene il tempo di esecuzione del processo corrente.
- Se il quanto di tempo non è scaduto la routine ritorna e il processo prosegue nell'esecuzione.
- Se invece il quanto di tempo è scaduto, la routine invoca una particolare funzione del nucleo (**preempt**) che cambia lo stato del processo da esecuzione a pronto, salva il contesto del processo e attiva una commutazione di contesto.

Ulteriore specifica sul ritorno da interrupt o da SVC

- **Prima di eseguire un'istruzione IRET che porti al modo di esecuzione U viene invocata la funzione preempt.**
- La funzione verifica se il quanto di tempo del processo corrente (quello a cui si torna) è scaduto e, in caso positivo, esegue la commutazione di contesto.
- In questo modo, l'operazione di preemption non viene dilazionata in caso di interrupt annidati.

Strutture dati fondamentali del nucleo (1)

- Il SO dispone di una **tabella dei processi** (**`Proctable`**) che contiene un elemento per ogni processo che è stato creato.
- Gli elementi della tabella possono essere visti come *record* (di tipo ProcRec): il record contiene i campi indispensabili a caratterizzare il processo relativamente alla gestione dei processi, gestione della memoria e gestione dei file.
- Una variabile **`CurrProc`** contiene l'indice del processo correntemente in esecuzione.

Strutture dati fondamentali del nucleo (2)

- Lo stack di sistema può essere visto come un "tabella" di stack di sistema: esiste quindi **uno stack di sistema per ogni processo che è stato creato**.
- La variabile **`CurrProc`** individua l'elemento nell'array di stack relativo al processo correntemente in esecuzione.
- Il registro **sSP** punta di volta in volta allo stack di sistema del processo correntemente in esecuzione.

La tabella dei processi

```
ProcRec     ProcTable[MAXPROC];
int         CurProc;
struct  ProcRec {
  Stato /*indica lo stato del processo*/
  Pila  /*contiene il valore del puntatore alla pila di modo S, salvato
    quando l'esecuzione  del processo viene sospesa*/
  Evento /*contiene l'identificatore dell'evento sul quale il
    processo è in attesa*/
  Base /*contiene la base del processo */
  File Aperti  /* è una tabella che contiene i riferimenti ai file aperti del
    processo (trattata nel capitolo sul file system);  dato che le
    periferiche sono associate a file speciali, anche il terminale del
    processo (standard input e standard output) è definito da questa
    tabella */
  RegistriUtente /*valori dei registri usati dal processo*/
        /*altre variabili che non interessano per il momento*/  }
```

Funzioni che manipolano la tabella dei processi

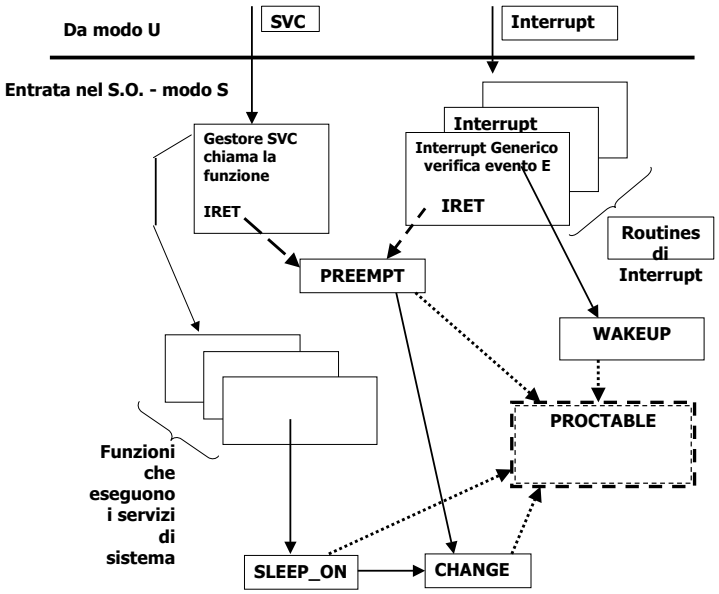
- **Sleep_on**: pone il processo corrente in stato di attesa, con salvataggio del contesto.
- **Change**: esegue una commutazione di contesto.
- **Wakeup**: porta un processo da stato di attesa a stato di pronto.
- **Preempt**: sospende il processo in esecuzione per scadenza del quanto di tempo, con salvataggio del contesto.

Contesto di un processo

- Contesto: insieme delle informazioni che caratterizzano lo stato di un processo:
  - se il processo è fisicamente in esecuzione **parte del contesto** si trova nei registri della CPU (Program Counter, Stack Pointer, PSW, registri utente);
  - se il processo non è in esecuzione il contesto è tutto in memoria.

Contesto di un processo

- Corrisponde allo stato del processo ed è caratterizzato dalle seguenti informazioni:
  - codice;
  - valori delle variabili globali e strutture dati a livello user;
  - i valori contenuti nei registri del processore;
  - le informazioni memorizzate nella tabella dei processi;
  - il contenuto dello stack user e dello stack kernel.



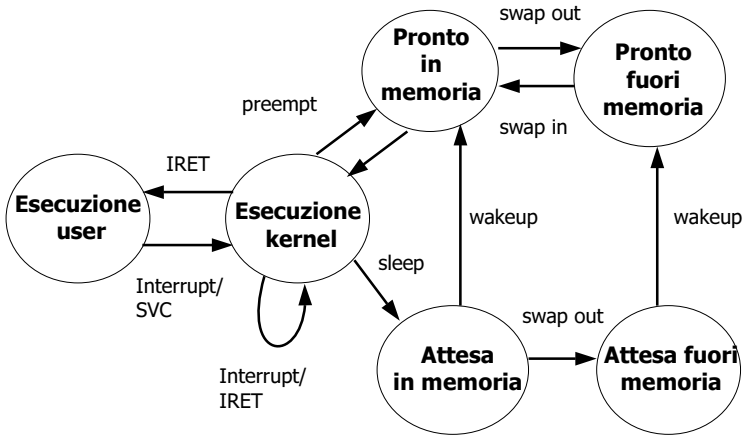
Stati di un processo

- Nel caso di memoria virtuale i processi possono trovarsi in memoria o fuori memoria (su disco).
- Gli stati di attesa e pronto si sdoppiano:
  - attesa in memoria e attesa fuori memoria;
  - pronto in memoria e pronto fuori memoria.
- Lo spostamento di un processo fuori o in memoria centrale si chiama **swapping.**
- Il sistema operativo sposta un processo fuori memoria se necessita di spazio in memoria.

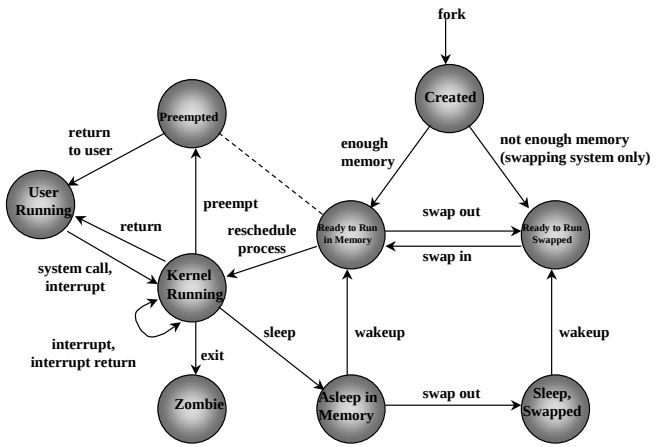
Transizioni fuori memoria

- Lo spazio di memoria può essere richiesto nei seguenti casi:
  - l'esecuzione di una fork richiede l'allocazione di spazio di memoria per il processo figlio;
  - una richiesta di memoria dell'area dati (mediante *brk*) incrementa le dimensioni del processo;
  - crescita del segmento stack implica la crescita delle dimensioni globali del processo;
  - il Sistema Operativo vuole liberare spazio in memoria per i processi che ha portato fuori memoria precedentemente e che sono pronti.

Stati di un processo



Stati di un processo in UNIX



Differenze rispetto al modello generale

- Esiste una distinzione tra lo stato *Ready to Run in Memory* e *Preempted.* Si tratta sostanzialmente dello stesso stato. Serve per enfatizzare come si entra nello stato di preempted, al termine dell'esecuzione in modo kernel da parte del processo.
- *Created:* il processo è appena stato creato ed è pronto per l'esecuzione.
- *Zombie*: il processo non esiste più, ma lascia delle informazioni per il processo padre.

Creazione di un processo (fork)

- Realizzazione della *fork* da parte del kernel:
  1. verifica se esistono sufficienti risorse per completare la fork con successo (es. Verifica se esiste spazio sufficiente in memoria o su disco ed eventualmente deve allocare delle pagine di memoria);
  2. alloca una riga nella tabella dei processi per il nuovo processo figlio;
  3. assegna un PID univoco al processo figlio;
  4. esegue una copia del contesto del processo padre (u-area, codice, dati, stack);
  5. incrementa di 1 i contatori nelle tabelle dei file degli i-node per i file associati con il processo padre e ora anche con il figlio;
  6. pone lo stato del processo a PRONTO;
  7. ritorna il PID al processo padre e 0 al processo figlio.

Terminazione di un processo

- Avviene con l'esecuzione della chiamata di sistema *exit (status).*
- Realizzazione da parte del kernel:
  - chiude tutti i file aperti (close) e rilascia gli i-node associati al direttorio corrente;
  - libera la memoria dello spazio utente;
  - scrive informazioni relative a statistiche run-time riguardanti l'esecuzione del processo in un file globale;
  - in Unix il processo passa nello stato *zombie*;
  - sgancia il processo dall'albero dei processi facendo in modo che il processo 1 adotti tutti i suoi processi figli (se esistono);
  - esegue un cambiamento di contesto in modo da poter mandare in esecuzione un altro processo pronto.

Attesa della terminazione di un processo

- Un processo può sincronizzare la propria esecuzione con la terminazione di un processo figlio mediante l'esecuzione della chiamata di sistema wait.
- L'esecuzione della wait comporta la ricerca da parte del kernel di un figlio zombie del processo, e se il processo non ha figli, ritorna un errore.
- Se identifica un figlio in stato di zombie, ne estrae il PID e il parametro fornito dalla exit del processo figlio e ritorna questi valori.
- Il kernel libera la riga della tabella dei processi occupata dal processo figlio zombie.

Esecuzione di wait

- Se il processo che esegue la wait ha processi figli ma nessuno è zombie, il processo esegue una sleep e verrà risvegliato al momento della terminazione di un suo processo figlio.
- In questo caso inizia nuovamente l'esecuzione della wait e trova un processo figlio zombie.

Inizializzazione del sistema operativo

- Obiettivo della fase di inizializzazione è caricare una copia del sistema operativo in memoria e iniziarne l'esecuzione.
- La procedura di bootstrap sulle macchine UNIX legge il blocco di bootstrap (blocco 0) da un disco e lo carica in memoria.
- Il programma contenuto nel blocco di bootstrap carica il kernel dal file system e poi trasferisce il controllo all'indirizzo di inizio del kernel che va in esecuzione.

Inizializzazione del sistema operativo

- Il kernel inizializza le strutture dati interne
  - Costruisce le liste di i-node e buffer, inizializza la tabella delle pagine,....
- *Mount* del file system di root e creazione del contesto per il *processo 0*, inizializzando la riga 0 della tabella dei processi.
- Quando il contesto del processo 0 è pronto il sistema è in esecuzione come processo 0.
- Il processo 0 esegue una *fork* e crea il processo 1 (processo *init*).
- Dopo aver creato il processo 1, il processo 0 diventa il processo di *swapper* (per la gestione della memoria virtuale).

Inizializzazione del sistema operativo

- Il processo 1 viene creato artificialmente.
- Il processo 1, in esecuzione in modo kernel, crea il suo spazio di indirizzamento utente e copia il codice da eseguire dallo spazio di indirizzamento kernel a quello utente.
- Il processo 1 esegue il codice copiato in modo utente.
- Questo codice consiste una chiamata di sistema *exec* per mandare in esecuzione il programma "`/etc/init`" che è responsabile dell'inizializzazione dei nuovi processi.
- Il processo 1 è anche chiamato `init`.

Inizializzazione del sistema operativo

- Il processo `init` legge il file `/etc/inittab` che indica quali processi creare.
- In un sistema multiutente `inittab` richiede la creazione di tanti processi `getty` quanti sono i terminali disponibili.

Bibliografia

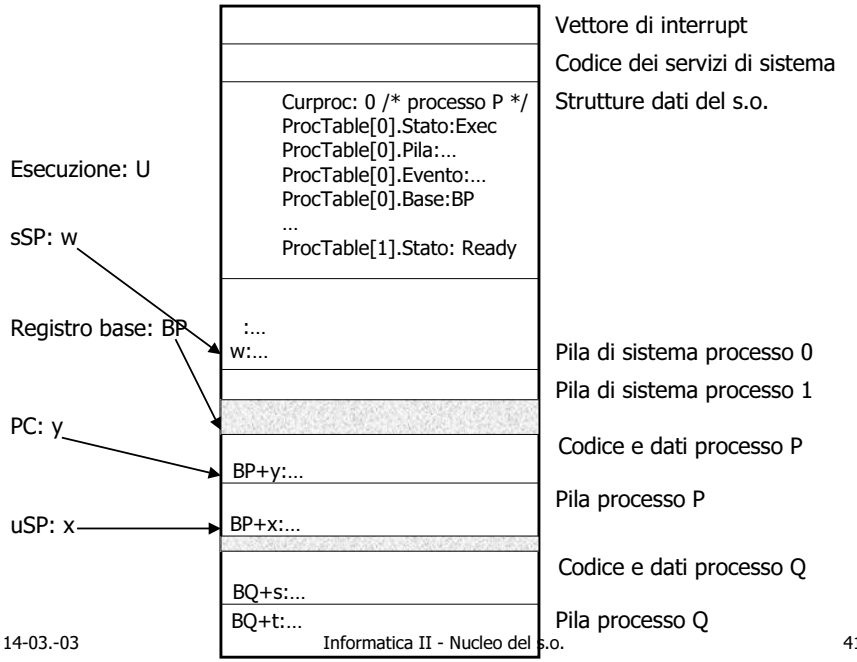
- Pelagatti G., Sistema Operativo Linux e TCP/IP, Progetto Leonardo, Bologna, 2002 - capitolo 9.

Esempio

- Esempio di utilizzo della tabella `procTable` da parte del sistema operativo.

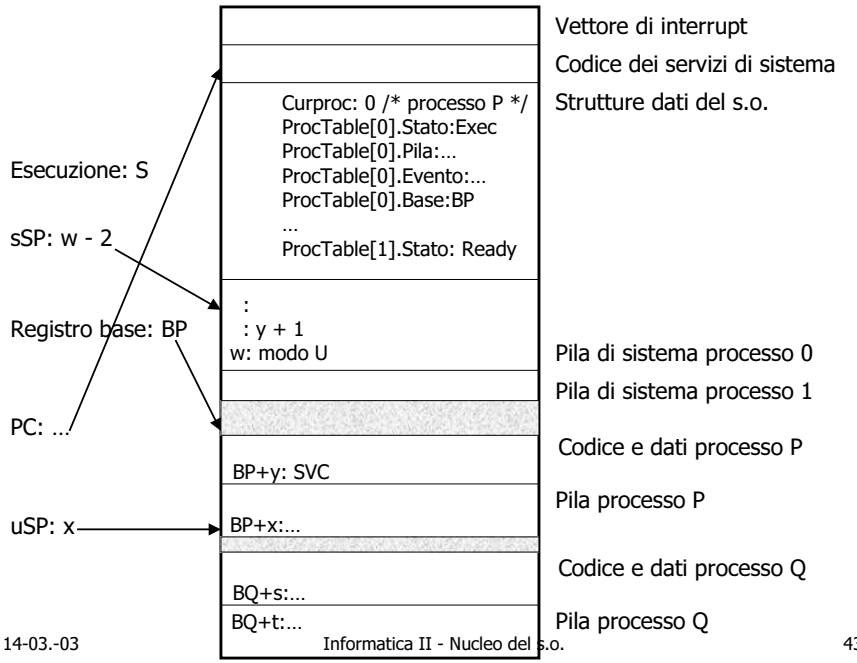
Situazione iniziale

- Il processo P è in esecuzione:
  - uSP punta a x + BP;
  - PC punta a y + BP;
  - l'istruzione in esecuzione è nella cella y -1 + BP;
  - BP punta all'inizio dell'area di memoria dedicata al processo P;
  - sSP punta a w nella pila di sistema del processo P che ha `curproc` = 0.



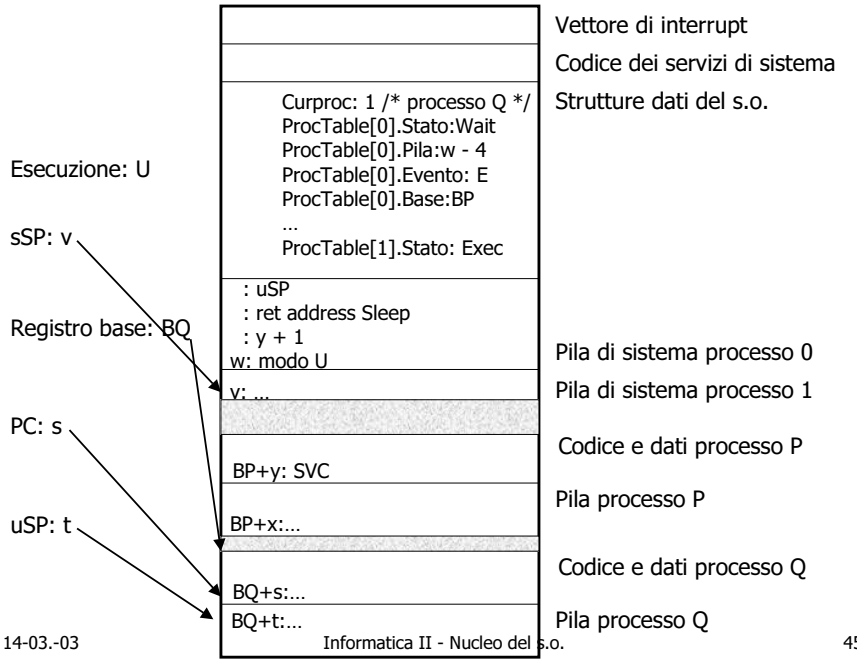
P richiede un servizio di sistema

- Sulla pila S viene memorizzato il modo di ritorno del processo (U), l'indirizzo di ritorno y + 1.
- Il processore commuta in modo S e carica nel PC l'indirizzo di inizio della routine del servizio di sistema specificato.
- Se il servizio richiesto è una `read`, il processo P si sospenderà e sarà attivato il processo Q.



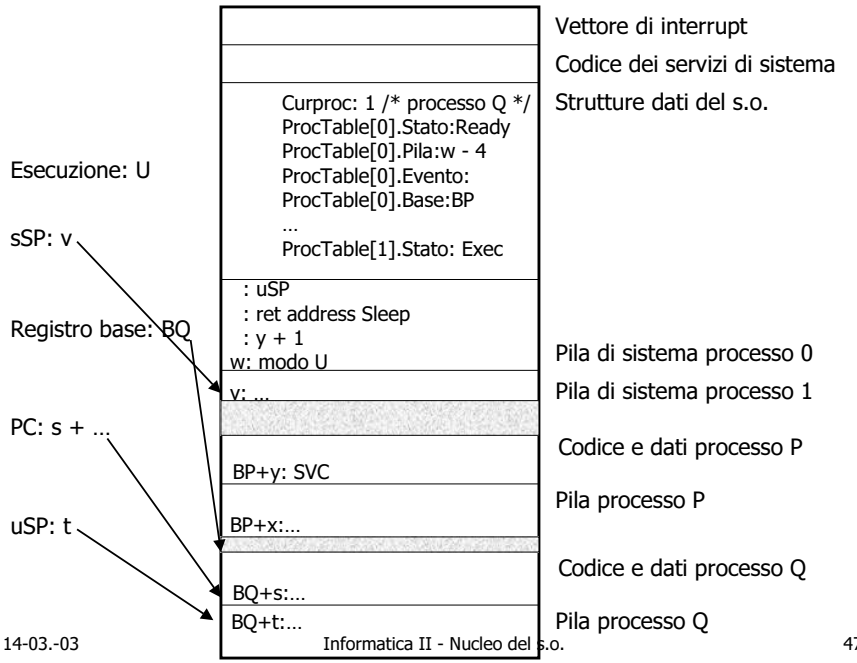
P si sospende, Q va in esecuzione

- Il servizio richiesto da P è una `read`, il processo P si sospende, invocando la `sleep_on(evento)` sull'evento atteso: sullo stack S sono salvati l'indirizzo di ritorno della Sleep ed il **valore di uSP**.
- La funzione `change` commuta il contesto (`curproc=1`).
- L'esecuzione riprende con un ritorno alla funzione chiamante prelevando l'indirizzo dalla pila S (perchè il processore è in modo S). Il ritorno avviene a quella funzione di Q che aveva invocato `change` quando era stata sospesa l'esecuzione di Q.
- Il processore entra in modalità U ed il processo Q è attivato.



Durante l'esecuzione di Q si verifica l'evento atteso da P

- La routine di interrupt del terminale eseguita nel contesto di Q invoca la `Wakeup(evento)` che risveglia tutti i processi in attesa dell'evento riportato come parametro.



Il processo Q esaurisce il quanto di tempo

- Dopo alcuni segnali dall'orologio di sistema, Q esaurisce il suo quanto di tempo: la routine di interrupt invocata dall'orologio esegue la funzione `preempt`, che forza in `ready` il processo Q e manda in esecuzione il processo P.
- L'esecuzione di P riprende dall'istruzione di completamento della `sleep_on` (che trasferisce dal buffer al programma il dato letto) e continua poi con l'istruzione y + 1.