POLITECNICO DI MILANO

Esercizi
Nucleo & File System

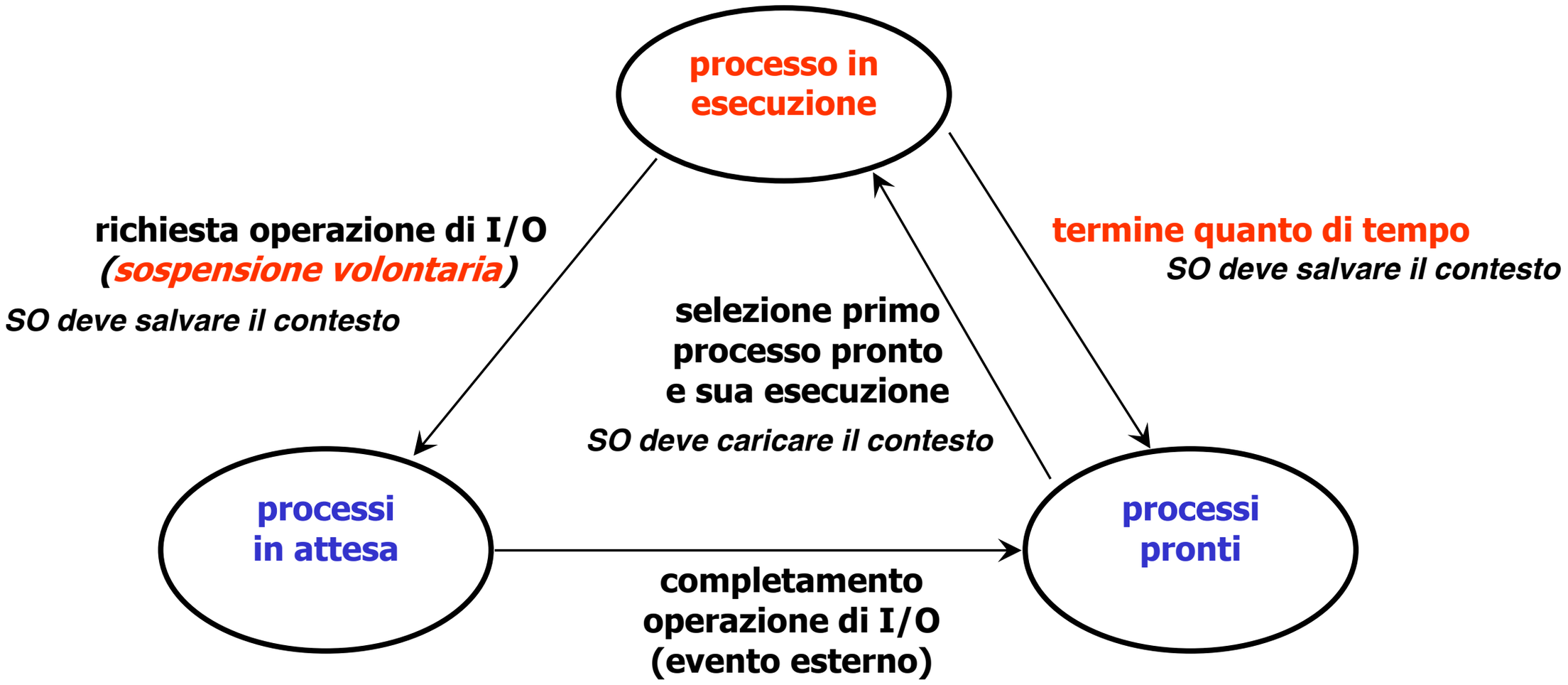
Alessandro A. Nacci
alessandro.nacci@polimi.it

ACSO
2014/2014

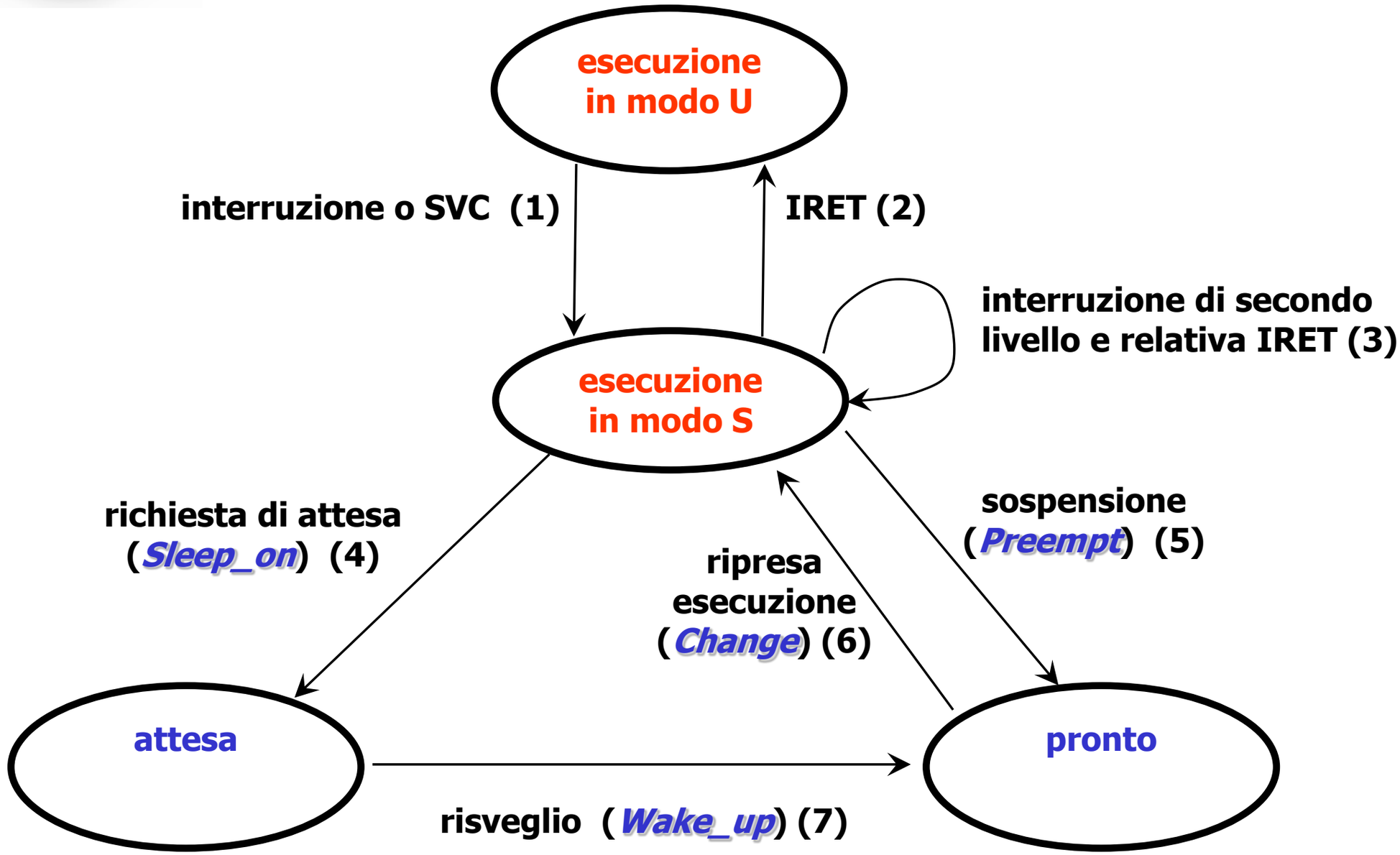

START

Le soluzioni sono sui PDF dei temi d’esame

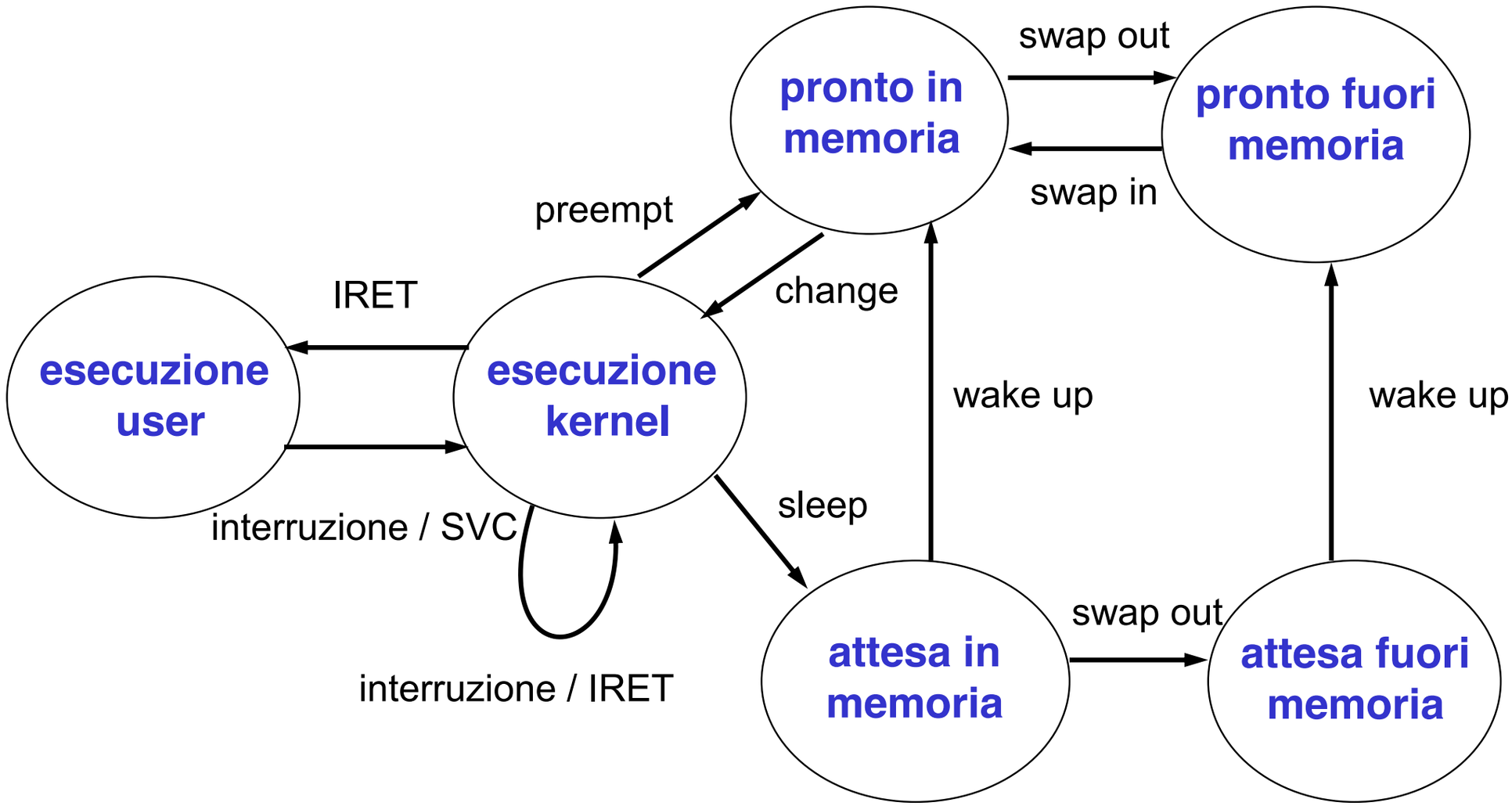
Stati di esecuzione e transizioni



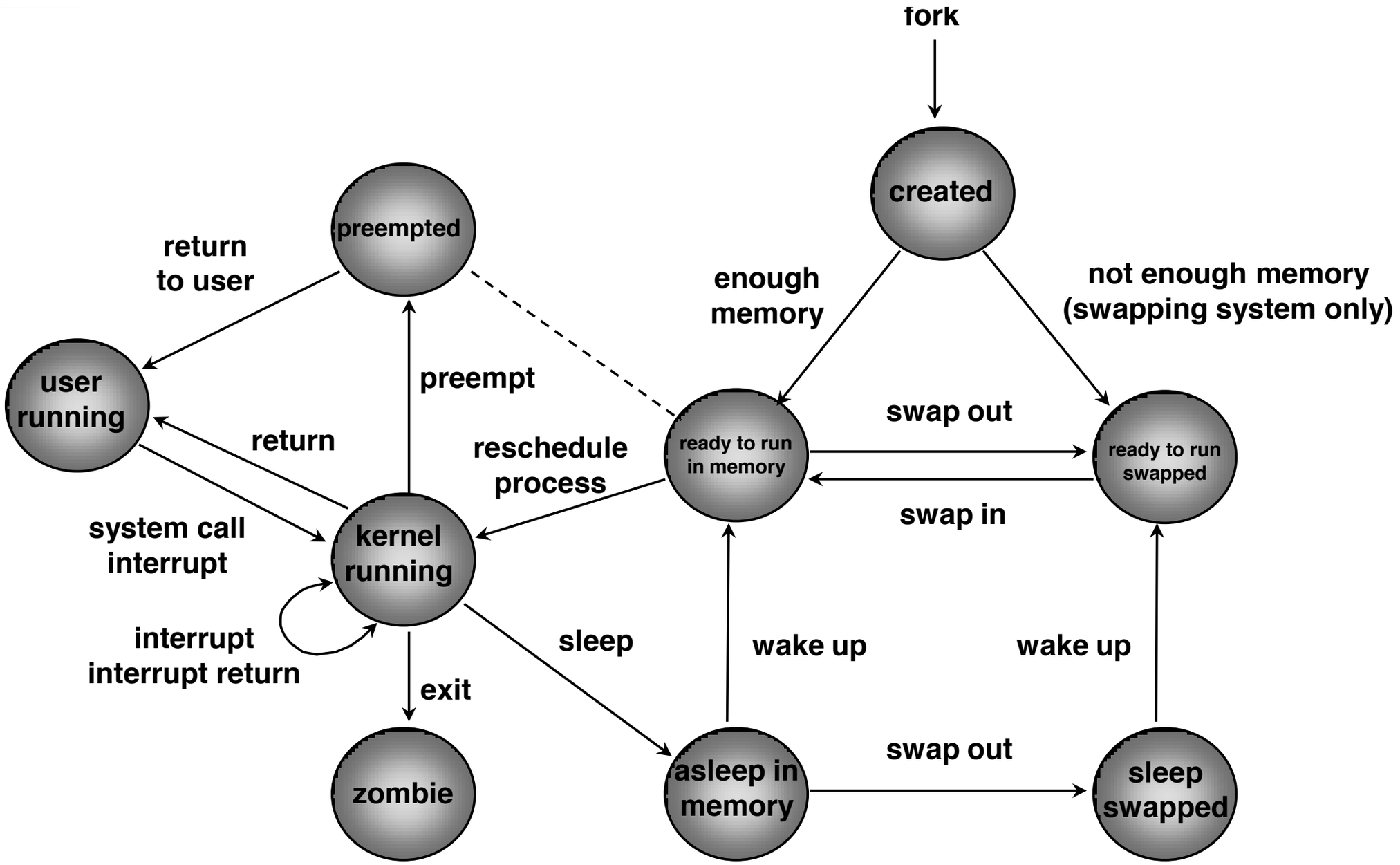
Grafo degli stati



Stato del Processo e Memoria Virtuale



Stato del Processo in Unix



9 settembre 2014

Sono dati due **processi P** e **S**. Il processo **P** esegue il programma **CODICE_UNO** e crea il processo figlio **Q**. Il processo **S** (che non è figlio né di **P** né di **Q**) esegue il programma **CODICE_DUE**. Nel sistema non ci sono altri processi utente oltre a **P**, **S** e **Q**.

```
/* programma CODICE_UNO.c eseguito dai processi P e Q                    */
main ( ) {

(1) fd1 = open ("/acso/file1", O_RDWR);              /* 4 blocchi   */
(2) read (fd1, vet_P, 540);                          /* 2 blocchi   */
    pid = fork ( );
    if (pid == 0) {          /* codice eseguito da Q figlio di P   */
(3)    write (fd1, vet_Q, 1024);                     /* 2 blocchi   */
(4)    lseek (fd1, 30, 0);   /* 0 si riferisce a inizio file       */
(5)    fd2 = open ("/acso/file2", O_RDWR);           /* 2 blocchi   */
(6)    write (fd2, vet_Q, 1024);                     /* 2 blocchi   */
       ...
       exit (1);
    } else {                 /* codice eseguito da P               */
(7)    read (fd1, vet_P, 10);                        /* 1 blocco    */
       pid = wait (&status);
       ...
       exit (0);
    } /* if */
} /* CODICE_UNO */
```

```
/* programma CODICE_DUE.c eseguito dal processo S                        */
main ( ) {

(8)  fd = open ("/acso/file2", O_RDWR);              /* 4 blocchi   */
(9)  lseek (fd, 430, 1);    /* 1 si riferisce a posizione corr. */
(10) write (fd, vet_S, 1030);                        /* 2 blocchi   */
     ...
     exit (2);
} /* CODICE_DUE */
```

9 settembre 2014

Ulteriori specifiche:

1. i processi utente hanno una priorità associata, il processo "idle" ha priorità minima e nel sistema non ci sono altri processi oltre a quelli utente e a "idle"
2. quando è necessario, sono indicate le priorità dei processi attivi nel sistema
3. quando un processo diventa pronto e ha priorità maggiore di quello in esecuzione, va attivato lo scheduler
4. il buffer del driver di standard output ha dimensione di **50 caratteri**
5. per le operazioni sui file che – nell'esercizio considerato – implicano trasferimento di blocchi:
    a. l'interruzione di fine DMA è associata al trasferimento di un singolo blocco di file: evento *DMA_in* per lettura di un blocco da file ed evento *DMA_out* per scrittura di un blocco su file
    b. il numero di blocchi da trasferire è indicato esplicitamente come commento nel codice
6. le chiamate di sistema "wait" e "waitpid" invocano la funzione "Sleep_on" su un evento opportuno
7. per completare la tabella delle commutazioni, si faccia riferimento alla notazione vista a lezione

Si completino le parti mancanti della tabella di commutazione dei processi. Nella tabella sono previste righe da completare, dove:

a. è specificato l'evento (con informazioni aggiuntive); qui sono da completare le parti relative allo stato dei processi e – se richiesto – ai moduli del S.O. con il contesto

b. è specificato lo stato dei processi raggiunto dopo il verificarsi dell'evento; qui sono da completare i campi relativi all'evento (con eventuali informazioni aggiuntive) e – se richiesto – ai moduli del S.O. con il contesto

NOTA BENE:

- l'evento è sempre determinabile univocamente dallo stato raggiunto, dall'evoluzione precedente dei processi, dal codice dei programmi e dalle ulteriori specifiche di sistema
- se l'evento è un interrupt, va usata la notazione "*n* interrupt", indicando esattamente il numero di interruzioni che si sono verificate

9 settembre 2014

seconda parte – funzioni del file system

Si consideri nuovamente il codice dei programmi dati nella prima parte e in particolare le chiamate di sistema identificate dai numeri d'ordine da **1** a **10**.

Relativamente a questa seconda parte valgono le specifiche seguenti:

- l'area buffer in memoria centrale gestita dal SO per le operazioni su file è costituita da **4 buffer** di dimensione opportuna che possono essere assegnati indifferentemente a un qualsiasi file del sistema. In caso di mancanza di spazio l'area buffer viene gestita con politica **LRU**.
- per **tutte le operazioni** su file
  - (a) la dimensione del blocco trasferito da o verso file tramite **DMA** è di **512 byte**
  - (b) l'interruzione di fine DMA è associata al trasferimento di un singolo blocco
  - (c) le chiamate di sistema sono dipendenti e pertanto i blocchi allocati/letti dalle precedenti chiamare di sistema rimangono a disposizione in memoria finché è possibile
  - (d) gli *I-node*, una volta trasferiti da disco, vengono copiati in memoria centrale nella *I-lista* e il buffer occupato per il loro trasferimento viene subito considerato "libero"
- per l'**apertura** del file è **sempre** necessario accedere a:
  - (a) un blocco per l'accesso allo *I-node* di ogni cartella (catalogo) o file presente nel nome-percorso (pathname)
  - (b) un blocco per il contenuto di ogni cartella (catalogo) presente nel nome-percorso (pathname)
- i blocchi possono trovarsi in memoria (se già acceduti) o necessitare di trasferimento DMA da disco

9 settembre 2014

Nota: **lo *I-node* di root è sempre in memoria** e il contenuto dei blocchi dei file normali è omesso poiché non è significativo ai fini dell'esercizio.

Si ipotizzi che la sequenza di ordine delle chiamate di sistema sia **1 2 8 9 3 10**.

Per ciascuna della chiamate, **si indichino** nella tabella predisposta:

- il numero totale di interruzioni di fine DMA che si verificano affinché l'operazione si completi
- e la sequenza di accessi:
  - agli *I-node* su disco (notazione "*I-node [X]*", con X = numero *I-node*)
  - agli *I-node* nella lista in memoria centrale (notazione "*I-lista [X]*", con X = numero *I-node*)
  - e ai blocchi (con la notazione "*blocco Y*"), specificando se a memoria (**M**) o a disco (**D**)

Nell'ultima colonna si indichi anche lo stato dei quattro buffer alla fine della gestione di ogni chiamata, indicando per ciascuno il numero di blocco presente e il file a cui appartiene, oppure se è libero (**L**).

9 settembre 2014

I-lista: ⟨ 0, dir, 4 ⟩ ⟨ 6, dir, 20 ⟩

⟨ 32, norm, (600, 601, 602, 603, 604, 605, 606) ⟩

⟨ 48, norm, (800, 812, 700, 703, 705) ⟩

blocco 4: ... ⟨ 6, acso ⟩ ...

blocco 20: ... ⟨ 32, file1 ⟩ ⟨ 48, file2 ⟩ ...

ordine delle chiamate di sistema sia **1 2 8 9 3 10**.

```
/* programma CODICE_UNO.c eseguito dai processi P e Q          */
main ( ) {

(1) fd1 = open ("/acso/file1", O_RDWR);          /* 4 blocchi  */
(2) read (fd1, vet_P, 540);                      /* 2 blocchi  */
    pid = fork ( );
    if (pid == 0) {        /* codice eseguito da Q figlio di P  */
(3)    write (fd1, vet_Q, 1024);                 /* 2 blocchi  */
(4)    lseek (fd1, 30, 0);   /* 0 si riferisce a inizio file    */
(5)    fd2 = open ("/acso/file2", O_RDWR);       /* 2 blocchi  */
(6)    write (fd2, vet_Q, 1024);                 /* 2 blocchi  */
       ...
       exit (1);
    } else {               /* codice eseguito da P             */
(7)    read (fd1, vet_P, 10);                    /* 1 blocco   */
       pid = wait (&status);
       ...
       exit (0);
    } /* if */
} /* CODICE_UNO */
```

```
/* programma CODICE_DUE.c eseguito dal processo S              */
main ( ) {

(8)  fd = open ("/acso/file2", O_RDWR);          /* 4 blocchi  */
(9)  lseek (fd, 430, 1);    /* 1 si riferisce a posizione corr. */
(10) write (fd, vet_S, 1030);                    /* 2 blocchi  */
     ...
     exit (2);
} /* CODICE_DUE */
```

9 settembre 2014

I-lista: ⟨ 0, dir, 4 ⟩ ⟨ 6, dir, 20 ⟩

⟨ 32, norm, (600, 601, 602, 603, 604, 605, 606) ⟩

⟨ 48, norm, (800, 812, 700, 703, 705) ⟩

blocco 4: ... ⟨ 6, acso ⟩ ...

blocco 20: ... ⟨ 32, file1 ⟩ ⟨ 48, file2 ⟩ ...

ordine delle chiamate di sistema sia **1 2 8 9 3 10**.

```
/* programma CODICE_UNO.c eseguito dai processi P e Q                    */
main ( ) {

(1)  fd1 = open ("/acso/file1", O_RDWR);          /* 4 blocchi   */
(2)  read (fd1, vet_P, 540);                      /* 2 blocchi   */
     pid = fork ( );
     if (pid == 0) {          /* codice eseguito da Q figlio di P   */
(3)     write (fd1, vet_Q, 1024);                 /* 2 blocchi   */
(4)     lseek (fd1, 30, 0);   /* 0 si riferisce a inizio file       */
(5)     fd2 = open ("/acso/file2", O_RDWR);       /* 2 blocchi   */
(6)     write (fd2, vet_Q, 1024);                 /* 2 blocchi   */
        ...
        exit (1);
     } else {                 /* codice eseguito da P               */
(7)     read (fd1, vet_P, 10);                    /* 1 blocco    */
        pid = wait (&status);
        ...
        exit (0);
     } /* if */
} /* CODICE_UNO */
```

```
/* programma CODICE_DUE.c eseguito dal processo S                        */
main ( ) {

(8)  fd = open ("/acso/file2", O_RDWR);           /* 4 blocchi   */
(9)  lseek (fd, 430, 1);     /* 1 si riferisce a posizione corr. */
(10) write (fd, vet_S, 1030);                     /* 2 blocchi   */
     ...
     exit (2);
} /* CODICE_DUE */
```

27 febbraio 2014

Sono dati due **processi P** e **S**. Il processo **P** esegue il programma **CODICE_UNO** e crea il processo figlio **Q**. Il processo **S** (che non è figlio né di **P** né di **Q**) esegue il programma **CODICE_DUE**. Nel sistema non ci sono altri processi utente oltre a **P**, **S** e **Q**.

```
/* programma CODICE_UNO.c eseguito dai processi P e Q                */
main ( ) {

     char vet_P, vet_Q [...];    int fd1, fd2;    pid_t pid;

(1)  fd1 = open ("/acso/file1", O_RDWR);              /* 4 blocchi */
(2)  write (fd1, vet_P, 540);                         /* 2 blocchi */
     pid = fork ( );
     if (pid == 0) {          /* codice eseguito da Q figlio di P */
(3)     lseek (fd1, 48, 0); /* 0 si riferisce a inizio file      */
(4)     read (fd1, vet_Q, 1024);                      /* 4 blocchi */
(5)     fd2 = open ("/acso/file2", O_RDWR);           /* 4 blocchi */
(6)     write (fd2, vet_Q, 1024);                     /* 4 blocchi */
        ...
        exit (1);
     } else {                 /* codice eseguito dal padre P       */
(7)     read (fd1, vet_P, 10);                        /* 1 blocco  */
        pid = wait (&status);
        ...
        exit (0);
     } /* if */
} /* CODICE_UNO */
```

```
/* programma CODICE_DUE.c eseguito dal processo S                    */
main ( ) {

     char vet_S [...];    int fd;

(8)  fd = open ("/acso/file2", O_RDWR);               /* 4 blocchi */
(9)  lseek (fd, 560, 1);   /* 1 si riferisce a posizione corr.  */
(10) write (fd, vet_S, 512);                          /* 1 blocco  */
     ...
     exit (2);
} /* CODICE_DUE */
```

27 febbraio 2014

Ulteriori specifiche:

1. i processi utente hanno una priorità associata, il processo "idle" ha priorità minima e nel sistema non ci sono altri processi oltre a quelli utente e a "idle"
2. quando è necessario, sono indicate le priorità dei processi attivi nel sistema
3. quando un processo diventa pronto e ha priorità maggiore di quello in esecuzione, va attivato lo scheduler
4. il buffer del driver di standard output ha dimensione di **50 caratteri**
5. per le operazioni sui file che – nell'esercizio considerato – implicano trasferimento di blocchi:
    a. l'interruzione di fine DMA è associata al trasferimento di un singolo blocco di file: evento *DMA_in* per lettura di un blocco da file ed evento *DMA_out* per scrittura di un blocco su file
    b. il numero di blocchi da trasferire è indicato esplicitamente come commento nel codice
6. le chiamate di sistema "wait" e "waitpid" invocano la funzione "Sleep_on" su un evento opportuno
7. per completare la tabella delle commutazioni, si faccia riferimento alla notazione vista a lezione

Si completino le parti mancanti della tabella di commutazione dei processi. Nella tabella sono previste righe da completare, dove:

a. è specificato l'evento (con informazioni aggiuntive); qui sono da completare le parti relative allo stato dei processi e – se richiesto – ai moduli del S.O. con il contesto

b. è specificato lo stato dei processi raggiunto dopo il verificarsi dell'evento; qui sono da completare i campi relativi all'evento (con eventuali informazioni aggiuntive) e – se richiesto – ai moduli del S.O. con il contesto

NOTA BENE:

- l'evento è sempre determinabile univocamente dallo stato raggiunto, dall'evoluzione precedente dei processi, dal codice dei programmi e dalle ulteriori specifiche di sistema
- se l'evento è un interrupt, va usata la notazione "*n* interrupt", indicando esattamente il numero di interruzioni che si sono verificate

27 febbraio 2014

seconda parte – funzioni del file system

Si consideri nuovamente il codice dei programmi dati nella prima parte e in particolare le chiamate di sistema identificate dai numeri d'ordine da **1** a **10**.

Relativamente a questa seconda parte valgono le specifiche seguenti:

- l'area buffer in memoria centrale gestita dal SO per le operazioni su file è costituita da **4 buffer** di dimensione opportuna che possono essere assegnati indifferentemente ai blocchi di un qualsiasi file o catalogo del sistema; in caso di mancanza di spazio l'area buffer viene gestita con **politica LRU**
- per **tutte le operazioni** su file
  (a) la dimensione del blocco trasferito da o verso file tramite **DMA** è **512 byte**
  (b) l'interruzione di fine DMA è associata al trasferimento di un singolo blocco
  (c) le chiamate di sistema sono dipendenti e pertanto i blocchi allocati/letti dalle precedenti chiamate di sistema rimangono a disposizione in memoria finché possibile
  (d) per poter scrivere su file è necessario avere caricato i blocchi in memoria
- per l'**apertura** del file è **sempre** necessario accedere a:
  (a) un blocco per l'accesso allo *I-node* di ogni cartella (catalogo) o file presente nel nome-percorso (pathname); **una volta letti da disco, gli *I-node* dei file e delle cartelle vengono scritti e mantenuti sempre in memoria in lista apposita non facente parte dell'area buffer**, fino a quando il file/cartella è referenziato da almeno un processo
  (b) un blocco per il contenuto di ogni cartella (catalogo) presente nel nome-percorso (pathname)

Si consideri la seguente sequenza di esecuzione delle chiamate di sistema: **2 3 4 7 5 6**

Per ciascuna della chiamate, **si indichino** nella tabella predisposta (a pagina seguente):

- il numero totale di interruzioni di fine DMA che si verificano affinché l'operazione possa completarsi
- la sequenza di accessi agli I-node (notazione "I-node [X]", con X = numero i-node) e ai blocchi (con la notazione "blocco Y") specificando se a memoria (M) o a disco (D) e per i blocchi se in lettura o scrittura
- lo stato dei 4 buffer alla fine della gestione di ogni chiamata riportando per ciascun buffer il blocco presente e il file/cartella a cui appartiene, oppure se è libero (L)

N.B.: La prima riga della tabella è già compilata.

27 febbraio 2014

I-lista: ⟨ 0, dir, 4 ⟩ ⟨ 6, dir, 20 ⟩

⟨ 32, norm, (600, 601, 602, 603, 604, 605, 606) ⟩

⟨ 48, norm, (800, 812, 700, 703, 705) ⟩

blocco 4: ... ⟨ 6, acso ⟩ ...

blocco 20: ... ⟨ 32, file1 ⟩ ⟨ 48, file2 ⟩ ...

```
/* programma CODICE_UNO.c eseguito dai processi P e Q                    */
main ( ) {

     char vet_P, vet_Q [...];    int fd1, fd2;    pid_t pid;

(1)  fd1 = open ("/acso/file1", O_RDWR);                  /* 4 blocchi */
(2)  write (fd1, vet_P, 540);                             /* 2 blocchi */
     pid = fork ( );
     if (pid == 0) {          /* codice eseguito da Q figlio di P */
(3)     lseek (fd1, 48, 0); /* 0 si riferisce a inizio file     */
(4)     read (fd1, vet_Q, 1024);                          /* 4 blocchi */
(5)     fd2 = open ("/acso/file2", O_RDWR);               /* 4 blocchi */
(6)     write (fd2, vet_Q, 1024);                         /* 4 blocchi */
        ...
        exit (1);
     } else {                 /* codice eseguito dal padre P      */
(7)     read (fd1, vet_P, 10);                            /* 1 blocco  */
        pid = wait (&status);
        ...
        exit (0);
     } /* if */
} /* CODICE_UNO */
```

```
/* programma CODICE_DUE.c eseguito dal processo S                        */
main ( ) {

     char vet_S [...];    int fd;

(8)  fd = open ("/acso/file2", O_RDWR);                   /* 4 blocchi */
(9)  lseek (fd, 560, 1);   /* 1 si riferisce a posizione corr.  */
(10) write (fd, vet_S, 512);                              /* 1 blocco  */
     ...
     exit (2);
} /* CODICE_DUE */
```

26 novembre 2013

Su un calcolatore dotato di sistema operativo Linux viene eseguito il programma seguente, che parte come processo **P** e crea il processo **Q**.

NOTA: si ricorda che in **lseek** (fd, offset, riferimento), riferimento = 0 indica inizio file, riferimento = 1 indica posizione corrente e riferimento = 2 indica fine file.

`int main ( ) { /* processo P */`	**sequenza**
`...`	
`fd1 = open (“/acso/esame/file1”, O_RDWR);`	**1**
`read (fd1, buf1, 10);`	**2**
`write (fd1, buf2, 600);`	**3**
`pid1 = fork ()`	**4**
`if (pid1 == 0) { /* processo Q */`	
`fd2 = open (“/acso/esame/file2”, O_RDONLY);`	**5**
`lseek (fd2, 1024, 1);`	**6**
`read (fd1, buf1, 100);`	**9**
`close (fd1)`	**10**
`exit (0);`	**11**
`} else { /* processo P */`	
`read (fd1, buf1, 10);`	**7**
`close (fd1);`	**8**
`} /* if */`	
`exit (0);`	**12**
`} /* main */`	

Durante l'esecuzione è stata osservata la **sequenza di chiamate di sistema indicata nella colonna di destra**.

Si completi il contenuto delle tabelle seguenti, indicando la sequenza dei valori assunti fino alla conclusione della chiamata di sistema numero **12** (si scriva **L** oppure **NE** per indicare che una cella si è liberata o non esiste più). Per la determinazione degli I-node, si veda il contenuto del volume a pagina seguente:

Al momento dell'esecuzione, il contenuto del volume è il seguente:

I-Lista: < 0, dir, 4 > < 6, dir, 20 > < 7, dir, 31 > < 98, norm, { 800, 801, 802, 803, 804, … } >
< 110, norm, { 1718, 1719, … } > < 232, norm, { 2733, 2734, … } >
blocco 4: … < 6, acso > …
blocco 20: … < 7, esame > …
blocco 31: … < 98, file1 > < 110, file2 > < 232, file3 > …

26 novembre 2013

`int main ( ) { /* processo P */`	**sequenza**
`...`	
`fd1 = open (“/acso/esame/file1”, O_RDWR);`	**1**
`read (fd1, buf1, 10);`	**2**
`write (fd1, buf2, 600);`	**3**
`pid1 = fork ()`	**4**
`if (pid1 == 0) { /* processo Q */`	
`fd2 = open (“/acso/esame/file2”, O_RDONLY);`	**5**
`lseek (fd2, 1024, 1);`	**6**
`read (fd1, buf1, 100);`	**9**
`close (fd1)`	**10**
`exit (0);`	**11**
`} else { /* processo P */`	
`read (fd1, buf1, 10);`	**7**
`close (fd1);`	**8**
`} /* if */`	
`exit (0);`	**12**
`} /* main */`	

Si completi la tabella seguente, riportando, per ciascuna delle chiamate al File System indicate, la posizione corrente raggiunta dopo l’esecuzione del servizio, la sequenza di blocchi utilizzati e il tipo di accesso al disco eventualmente richiesto, con la notazione seguente:

- L (NB): legge il blocco numero NB dal disco
- S (NB): scrive il blocco numero NB sul disco

Si tenga conto delle ipotesi indicate in testa alla colonna relativamente al **numero di buffer disponibili** per contenere i blocchi del file.

Considerando le regole seguenti:

- le chiamate al File System sono eseguite nella sequenza indicata
- la dimensione di un blocco trasferito in DMA da o su file è di **512 byte**
- un blocco viene letto dal disco **solamente quando è necessario**
- per poter scrivere su un blocco è necessario averlo caricato in memoria, e un blocco viene scritto su disco **solamente quando è necessario** (per esempio quando il buffer che lo contiene va liberato)
- tutti gli I-node utilizzati **sono già presenti in memoria**

NOTA: i blocchi possono trovarsi in memoria (se già acceduti) o necessitare di trasferimento DMA da disco.

Alla prossima lezione

alessandro.nacci@polimi.it