Politecnico di Milano
Dipartimento di Elettronica, Informazione e Bioingegneria

prof.ssa Anna Antola — **prof. Giuseppe Pelagatti**
prof. Luca Breveglieri — **prof.ssa Donatella Sciuto**
prof. Roberto Negrini — **prof.ssa Cristina Silvano**

AXO – Architettura dei Calcolatori e Sistemi Operativi

Prova di lunedì 22 gennaio 2018

Cognome_____ **Nome** _____

Matricola_____ **Firma**_____

Istruzioni

- Si scriva solo negli spazi previsti nel testo della prova e non si separino i fogli.
- Per la minuta si utilizzino le pagine bianche inserite in fondo al fascicolo distribuito con il testo della prova. I fogli di minuta se staccati vanno consegnati intestandoli con nome e cognome.
- È vietato portare con sé libri, eserciziari e appunti, nonché cellulari e altri dispositivi mobili di calcolo o comunicazione. Chiunque fosse trovato in possesso di documentazione relativa al corso – anche se non strettamente attinente alle domande proposte – vedrà annullata la propria prova.
- Non è possibile lasciare l'aula conservando il tema della prova in corso.
- Tempo a disposizione 2 h : 00 m

Valore indicativo di domande ed esercizi, voti parziali e voto finale:

esercizio 1 (4 punti) _____

esercizio 2 (6 punti) _____

esercizio 3 (4 punti) _____

esercizio 4 (2 punti) _____

voto finale: (16 punti) _____

esercizio n. 1 – programmazione concorrente

Si consideri il programma C seguente (gli "`#include`" e le inizializzazioni dei *mutex* sono omessi, come anche il prefisso `pthread_` delle funzioni di libreria NPTL):

```
pthread_mutex_t power, ground
sem_t chord, wire
int global = 0
```

```
void * plug (void * arg) {
   mutex_lock (&power)
   sem_wait (&chord)
   mutex_unlock (&power)                          /* statement A */
   sem_post (&chord)
   mutex_lock (&ground)
   global = 1                                     /* statement B */
   sem_post (&wire)
   mutex_unlock (&ground)
   return (void * 2)
} /* end plug */
```

```
void * socket (void * arg) {
   mutex_lock (&power)
   mutex_lock (&ground)
   global = 3
   sem_post (&chord)
   mutex_unlock (&power)                          /* statement C */
   sem_wait (&chord)
   mutex_unlock (&ground)
   global = 4
   sem_wait (&wire)
   return (void * 5)
} /* end socket */
```

```
void main ( ) {
   pthread_t th_1, th_2, th_3
   sem_init (&chord, 0, 0)
   sem_init (&wire, 0, 0)
   create (&th_1, NULL, plug, NULL)
   create (&th_2, NULL, socket, NULL)
   join (th_2, &global)                           /* statement D */
   join (th_1, &global)
   return
} /* end main */
```

Si completi la tabella qui sotto **indicando lo stato di esistenza del *thread*** nell'istante di tempo specificato da ciascuna condizione, così: se il *thread* **esiste**, si scriva ESISTE; se **non esiste**, si scriva NON ESISTE; e se può essere **esistente** o **inesistente**, si scriva PUÒ ESISTERE. Ogni casella della tabella va riempita in uno dei tre modi (non va lasciata vuota).

Si badi bene alla colonna "condizione": con "subito dopo statement X" si chiede lo stato che il *thread* assume tra lo statement X e lo statement immediatamente successivo del *thread* indicato.

condizione	***thread***	
	th_1 – `plug`	**th_2 – `socket`**
subito dopo stat. **A**	Esiste	Esiste
subito dopo stat. **B**	Esiste	Esiste
subito dopo stat. **C**	Esiste	Esiste
subito dopo stat. **D**	Può esistere	Non esiste

Si completi la tabella qui sotto, **indicando i valori delle variabili globali** (sempre esistenti) nell'istante di tempo specificato da ciascuna condizione. Il **valore** della variabile va indicato così:

- intero, carattere, stringa, quando la variabile ha un valore definito; oppure X quando è indefinita
- se la variabile può avere due o più valori, li si riporti tutti quanti
- il semaforo può avere valore positivo o nullo (non valore negativo)

Si badi bene alla colonna "condizione": con "subito dopo statement X" si chiede il valore (o i valori) che la variabile ha tra lo statement X e lo statement immediatamente successivo del *thread* indicato.

condizione	**variabili globali**	
	chord	*global*
subito dopo stat. **A**	0	3
subito dopo stat. **B**	0	1 - 4
subito dopo stat. **C**	1 - 0	3
subito dopo stat. **D**	0	5

La global viene modificata dalla return solamente quando viene eseguita join

Il sistema può andare in stallo (*deadlock*), con uno o più *thread* che si bloccano, in **due casi diversi** (con *deadlock* si intende anche un blocco dovuto a un solo *thread* che non potrà mai proseguire). Si indichino gli statement dove avvengono i blocchi:

caso	**th_1 – `plug`**	**th_2 – `socket`**
1	sem_wait(&chord)	mutex_lock(&power)
2	sem_wait(&chord)	sem_wait(&wire)

esercizio n. 2 – processi e nucleo

prima parte – gestione dei processi

```
// programma prova.c
main ( ) {
   pid1 = fork ( )
   if (pid1 == 0) {          // codice eseguito dal figlio Q
      execl ("/acso/prog_x", "prog_x", NULL)
      exit (-1)
   } else {
      pid1 = wait (&status)
      write (stdout, msg, 25)
   } /* if */
   exit (0)
}  /* prova */
```

```
// programma prog_x.c
mutex_t ZERO = PTHREAD_MUTEX_INITIALIZER
sem_t RED, BLUE
```

void * LESS (void * arg) {		void * EQUAL (void * arg) {
sem_wait (&BLUE)		sem_post (&BLUE)
mutex_lock (&ZERO)		mutex_lock (&ZERO)
sem_wait (&BLUE)		mutex_unlock (&ZERO)
sem_post (&RED)		sem_wait (&RED)
mutex_unlock (&ZERO)		return NULL
return NULL		} /* EQUAL */
} /* LESS */		

```
main ( ) {  // codice eseguito da Q
    pthread_t TH_1, TH_2
    sem_init (&BLUE, 0, 1)
    sem_init (&RED, 0, 0)
    create (&TH_2, NULL, EQUAL, NULL)
    create (&TH_1, NULL, LESS, NULL)
    join (TH_2, NULL)
    join (TH_1, NULL)
    exit (1)
}  /* main */
```

```
// programma esempio.c
main ( ) {
    fd = open ("/acso/dati", O_RDWR)
    nanosleep (3)
    read (fd, vett, 512)
    exit (1)
}  /* main */
```

Un processo ***P*** esegue il programma ***prova*** e crea un figlio ***Q*** che esegue una mutazione di codice (programma ***prog_x***). La mutazione di codice va a buon fine e sono creati i thread ***TH1*** e ***TH2***. Un processo ***S*** esegue il programma ***esempio***.

Si simuli l'esecuzione dei processi completando tutte le righe presenti nella tabella così come risulta dal codice dato, dallo stato iniziale e dagli eventi indicati. Si completi la tabella riportando quanto segue:

- ⟨ *PID*, *TGID* ⟩ di ciascun processo che viene creato
- ⟨ *identificativo del processo-chiamata di sistema / libreria* ⟩ nella prima colonna, dove necessario e in funzione del codice proposto
- in ciascuna riga lo stato dei processi **al termine dell'evento o della chiamata associata alla riga stessa**; si noti che la prima riga della tabella **potrebbe essere solo parzialmente completata**

TABELLA DA COMPILARE (numero di colonne non significativo)

identificativo simbolico del processo		***IDLE***	***S***	***P***	Q	TH2	TH1		
evento oppure *processo-chiamata*	*PID*	**1**	**2**	3	4	5	6		
	TGID	**1**	**2**	3	4	4	4		
P – pid1 = fork	**0**	**pronto**	**A open**	**esec**	pronto	NE	NE		
P - wait (sys_wait)	1	pronto	A open	A wait	ESEC	NE	NE		
Q - execl (sys_execve)	2	pronto	A open	A wait	ESEC	NE	NE		
interrupt da *DMA_in*, tutti i blocchi richiesti trasferiti	3	pronto	ESEC	A wait	pronto	NE	NE		
S - nanosleep (sys_nano)	4	pronto	A nano	A wait	ESEC	NE	NE		
interrupt da *RT_clock* e scadenza *quanto di tempo*	5	pronto	A nano	A wait	ESEC	NE	NE		
Q: create(TH2) (sys_clone)	6	pronto	A nano	A wait	ESEC	pronto	NE		
interrupt da RT_clock e scadenza *timer* di *nanosleep*	7	pronto	ESEC	A wait	pronto	pronto	NE		
S: read (sys_read)	8	pronto	A read	A wait	pronto	esec	NE		
TH1: sem_wait	9	pronto	A read	A wait	pronto	ESEC	NE		
TH1: mutex_lock(ZERO)	10	pronto	A read	A wait	pronto	ESEC	NE		
interrupt da *RT_clock* e scadenza *quanto di tempo*	11	pronto	A read	A wait	ESEC	pronto	NE		
Q: create(TH1) (sys_clone)	12	pronto	A read	A wait	ESEC	pronto	pronto		
Q: join(TH2) (sys_wait)	13	pronto	A read	A wait	A join(TH2)	ESEC	pronto		
interrupt da RT_clock e scadenza quanto di tempo	14	pronto	A read	A wait	A join	pronto	esec		
TH1: sem_wait(BLUE)	15	pronto	A read	A wait	A join	ESEC	pronto		

seconda parte – scheduling dei processi

Si consideri uno scheduler CFS con **tre task** caratterizzato da queste condizioni iniziali (**da completare**):

CONDIZIONI INIZIALI (da completare)							
RUNQUEUE	**NRT**	**PER**	**RQL**	**CURR**	**VMIN**		
	3	6	2	t1	100		
TASK	**ID**	**LOAD**	**LC**	**Q**	**VRTC**	**SUM**	**VRT**
CURRENT	t1	1	0.25	1.5	1	50	101
RB	t2	2	0.5	3	0.5	60	102
	t3	1	0.25	1.5	1	70	103

Durante l'esecuzione dei task si verificano i seguenti eventi:

```
events of task t1:   EXIT after 2,2;
events of task t2:   WAIT after 2,5;   WAKEUP after 0,5;
```

Simulare l'evoluzione del sistema per **4 eventi** riempiendo le seguenti tabelle. Indicare le eventuali valutazioni delle condizioni di *preemption* nell'apposito spazio alla fine dell'esercizio.

EVENTO		**TIME**	**TYPE**	**CONTEXT**	**RESCHED**	
		1.5	s QdT	T1	TRUE	
RUNQUEUE	**NRT**	**PER**	**RQL**	**CURR**	**VMIN**	
	3	6	4	T2	102	
TASK	**ID**	**LOAD**	**LC**	**Q**	**VRTC**	**SUM**	**VRT**
CURRENT	T2	2	0.5	3	0.5	60	102
RB	T1	1	0.25	1.5	1	51.5	102.5
	T3	1	0.25	1.5	1	70	103
WAITING							

T1 -> VRT = 101 + 1.5 * 1 = 102.5

EVENTO		**TIME**	**TYPE**	**CONTEXT**	**RESCHED**	
		4	wait	T2	TRUE	
RUNQUEUE	**NRT**	**PER**	**RQL**	**CURR**	**VMIN**	
	2	6	2	T1	102.5	
TASK	**ID**	**LOAD**	**LC**	**Q**	**VRTC**	**SUM**	**VRT**
CURRENT	T1	1	0.5	3	1	51.5	102.5
RB	T3	1	0.5	3	1	70	103
WAITING	T2	2				62.5	103.25

T2 -> VRT = 102 + 2.5 * 0.5 = 103.25

EVENTO		TIME	TYPE	CONTEXT	RESCHED		
		4.7	exit	T1	TRUE		
RUNQUEUE	NRT	PER	RQL	CURR	VMIN		
	1	6	1	T3	103		
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT	T3	1	1	6	1	70	103
RB							
WAITING	T2	2				62.5	103.25

T1 -> VRT = 102.5 + 0.7 * 1 = 103.2

RESCHEDULE: 103.25 + 1 * 1 = 104.25 > 103.5 => FALSE

EVENTO		TIME	TYPE	CONTEXT	RESCHED		
		5.2	wake up	T3	FALSE		
RUNQUEUE	NRT	PER	RQL	CURR	VMIN		
	2	6	3	T3	103.5		
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT	T3	1	1/3	2	1	70.5	103.5
RB	T2	2	2/3	4	0.5	62.5	103.25
WAITING							

T3 -> VRT = 103 + 0.5 * 1 = 103.5

T2 -> VRT = 103.25

Valutazione della necessità di rischedulazione:

Tempo dell'evento considerato: 5.2

Tipo di evento: wake up

Calcolo: RESCHEDULE: 103.25 + 1 * 1 = 104.25 > 103.5 => FALSE

esercizio n. 3 – memoria

È dato un sistema di memoria caratterizzato dai seguenti parametri generali:

MAXFREE = 3 **MINFREE = 2**

Si consideri la seguente **situazione iniziale**:

```
========== situazione iniziale ==========

PROCESSO: P ***************************************************
    PT:  <c0: 1 R>  <s0: s1 R>  <s1: - ->  <d0: s2 R>  <d1: - ->
         <p0: 2 R>  <p1: 6  W>  <p2: 3 W>  <p3: - ->

    process P - NPV of PC and SP: c0, p2

PROCESSO: Q ***************************************************
    PT:  <c0: 1 R>  <s0: s1 R>  <s1: - ->  <d0: s2 R>  <d1: - ->
         <p0: 2 R>  <p1: s0 W>  <p2: - ->

    process Q - NPV of PC and SP: c0, p1

____MEMORIA FISICA____(pagine libere: 3)_____________________________
      00 : <ZP>                ||  01 : Pc0 / Qc0 / <X,0>   ||
      02 : Pp0 / Qp0           ||  03 : Pp2                 ||
      04 : ----                ||  05 : ----                ||
      06 : Pp1                 ||  07 : ----                ||

____STATO del TLB_____________________________________________________
      Pc0 : 01 -  0: 1:   ||   Pp0 : 02 -  1: 0:   ||
      Pp2 : 03 -  1: 0:   ||         -----         ||
      Pp1 : 06 -  1: 0:   ||         -----         ||

SWAP FILE:  Qp1, Ps0 / Qs0, Pd0 / Qd0, ----, ----, ----, ----

LRU ACTIVE:    PC0

LRU INACTIVE:  pp2, pp1, pp0, qp0, qc0
```

Si rappresenti l'effetto dei seguenti eventi consecutivi sulle strutture dati della memoria compilando esclusivamente le tabelle fornite per ciascun evento (l'assenza di una tabella significa che non è richiesta la compilazione della corrispondente struttura dati).

ATTENZIONE: **le Tabelle sono PARZIALI – riempire solamente le celle indicate**

evento 1: write (Pp0)

PT del processo: **P**				
s0: <:s1 R>	d0: <:s2 R>	p0: <:4 W>	p1: <:6 W>	p2: <:3 W>
PT del processo: **Q**				
s0: <:s1 R>	d0: <s2 R>	p0: <:2 R >	p1: <:s0 W>	p2: <:- ->

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <X, 0>
02: Qp0 (D)	03: Pp2
04: Pp0	05: - - - -
06: Pp1	07: - - - -

SWAP FILE	
s0: Qp1	s1: Ps0 / Qs0
s2: Pd0 / Qd0	s3:
s4:	s5:
s6:	s7:

Active: PP0, PC0 **Inactive:** pp2, pp1, pp0, qp0, qc0

evento 2: read (Ps0, Pd0)

PT del processo: **P**				
s0: <:2 R>	d0: <:4 R>	p0: <:s4 W>	p1: <:6 W>	p2: <:3 W>
PT del processo: **Q**				
s0: <:s1 R>	d0: <:s2 R>	p0: <:s3 R>	p1: <:s0 W>	p2: <:- ->

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <X, 0>
02: Ps0 / Qs0	03: Pp2
04: Pd0 / Qd0	05: - - - -
06: Pp1	07: - - - -

SWAP FILE	
s0: Qp1	s1: Ps0 / Qs0
s2: Pd0 / Qd0	s3: Qp0
s4: Pp0	s5:
s6:	s7:

Active: PD0, PS0, PC0 **Inactive:** pp2, pp1, qc0, qs0, qd0

evento 3: write (Ps0)

PT del processo: **P**				
s0: <:3 W>	d0: <:4 R>	p0: <:s4 W>	p1: <:s5 W>	p2: <s6 W>
PT del processo: **Q**				
s0: <:2 R>	d0: <:4 R>	p0: <:s3 R>	p1: <:s0 W>	p2: <:- ->

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <X, 0>
02: Qs0	03: Ps0
04: Pd0 / Qd0	05: - - - -
06: - - - -	07: - - - -

SWAP FILE	
s0: Qp1	s1: Ps0 / Qs0
s2: Pd0 / Qd0	s3: Qp0
s4: Pp0	s5: Pp1
s6: Pp2	s7:

Active: PS0, PD0, PC0 **Inactive:** qc0, qs0, qd0

Indicare il numero di letture/scritture di pagine sullo swap file dopo ognuno dei precedenti eventi (valori non cumulativi):

EVENTO	num. PAG. LETTE da swapfile	num PAG. SCRITTE su swapfile
evento 1	0	0
evento 2	2	2
evento 3	0	2

esercizio n. 4 – domande su argomenti vari

prima parte – struttura Tabella delle Pagine

Date le VMA di un processo P sotto riportate, definire:

1. la decomposizione degli indirizzi virtuali dell'NPV iniziale di ogni area secondo la notazione:

   PGD:PUD:PMD:PT

2. il numero di pagine necessarie in ogni livello della gerarchia e il numero totale di pagine necessarie a rappresentare la Tabella delle Pagine (TP) del processo
3. il numero di pagine virtuali occupate dal processo
4. il rapporto tra l'occupazione della TP e la dimensione virtuale del processo in pagine
5. la dimensione virtuale massima del processo in pagine, senza dover modificare la dimensione della TP
6. il rapporto relativo

VMA del processo **P**							
area	**NPV iniziale**	**dimensione**	**R/W**	**P/S**	**M/A**	**nome file**	**offset**
C	*0000 0040 0*	*3*	*R*	*P*	*M*	*X*	*0*
K	*0000 0060 0*	*1*	*R*	*P*	*M*	*X*	*3*
S	*0000 0060 1*	*4*	*W*	*P*	*M*	*X*	*4*
D	*0000 0060 5*	*2*	*W*	*P*	*A*	*-1*	*0*
M0	*0000 1000 0*	*1*	*W*	*S*	*M*	*G*	*2*
M1	*0000 3000 0*	*3*	*W*	*P*	*M*	*F*	*2*
M2	*0000 4000 0*	*2*	*W*	*P*	*A*	*-1*	*0*
P	*7FFF FFFF C*	*3*	*W*	*P*	*A*	*-1*	*0*

1. Decomposizione degli indirizzi virtuali

area	**NPV iniziale**	**PGD**	**PUD**	**PMD**	**PT**
C	0000 0040 0	0	0	2	0
K	0000 0060 0	0	0	3	0
S	0000 0060 1	0	0	3	1
D	0000 0060 5	0	0	3	5
M0	0000 1000 0	0	0	128	0
M1	0000 3000 0	0	0	384	0
M2	0000 4000 0	0	1	0	0
P	7FFF FFFF C	255	511	511	508

2. Numero pagine necessarie:

   # pag PGD: 1

   # pag PUD: 2

   # pag PMD: 3

   # pag PT: 6

   # pag totali: 12

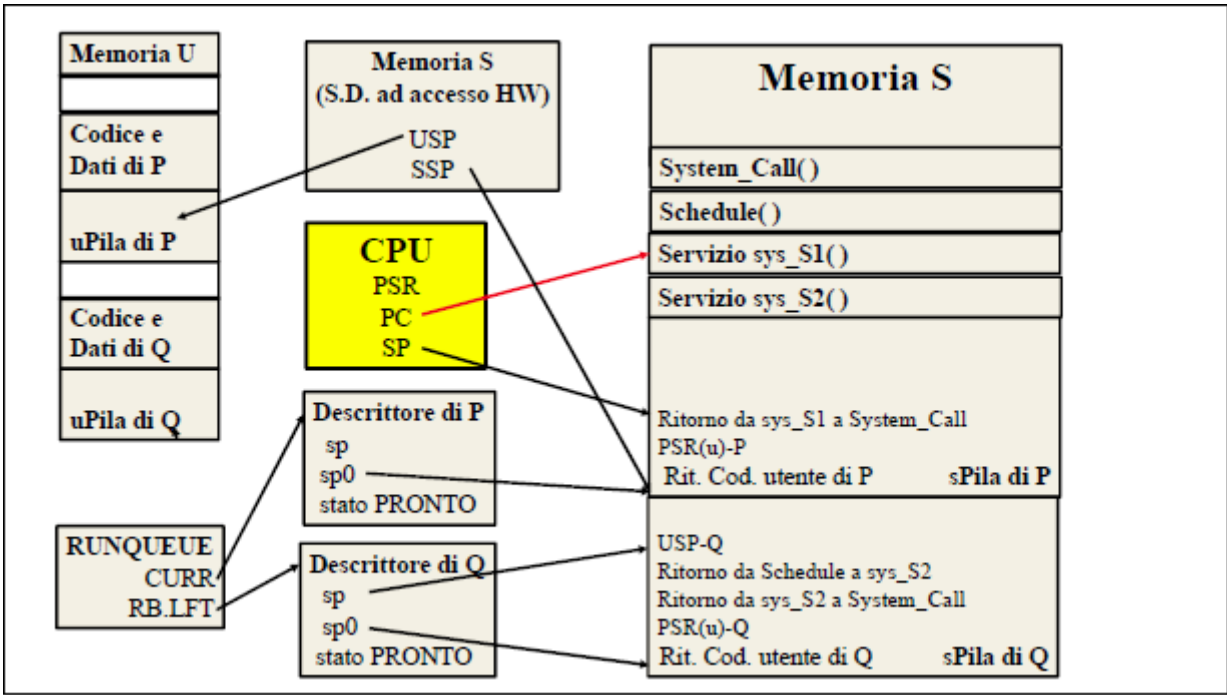
3. Numero pagine virtuali occupate dal processo: 19
4. Rapporto di occupazione: 12 / 19 = 63.16%
5. Dimensione massima del processo in pagine virtuali: 19 + 509 + 509 + 511 + 505 + 509 + 510 = 3072
6. Rapporto di occupazione con dimensione massima: 12 / 3072 = 0.390%

seconda parte – pila e strutture dati HW

Si considerino due processi P e Q. La situazione iniziale considerata è la seguente:

- il processo P è in esecuzione in modo S e sta eseguendo il servizio di sistema *sys_S1*
- il processo Q è l'unico processo della coda RB

La figura sotto riportata e i valori nella tabella successiva descrivono compiutamente, ai fini dell'esercizio, il contesto di P e il contesto di Q.



I valori della situazione iniziale di interesse sono i seguenti:

processo P	
PC	X
SP	Y
SSP	Z
USP	W
descrittore di P.stato	PRONTO
processo Q	
USP-Q	A
descrittore di Q.sp	B
descrittore di Q.sp0	C
descrittore di Q.stato	PRONTO
RUNQUEUE	
CURR	P
RB.LFT	Q

// è all'interno di *sys_S1*

Si consideri il seguente **evento**:

il servizio di sistema *sys_S1* sospende il processo P ponendolo in attesa e invoca *schedule ( )* per il *context switch*. Si supponga che l'invocazione di *schedule* in *sys_S1* avvenga all'indirizzo X + 5.

Domanda 1 – **salvataggio** del contesto di **P** eseguito durante il *context switch*

Completare la tabella seguente con i valori assunti dagli elementi subito dopo il salvataggio del contesto di P, ma prima della commutazione della pila di sistema.

processo P	
PC	
SP	
S_pila di P a (Y – 1)	
S_pila di P a (Y – 2)	
SSP	
USP	
descrittore di P.sp	
descrittore di P.sp0	
descrittore di P.stato	ATTESA

// non di interesse

Domanda 2 – **caricamento** del contesto di **Q** eseguito durante il *context switch*

Completare la tabella seguente con i valori assunti dagli elementi subito prima del ritorno da *schedule* al servizio di sistema *sys_S2*.

processo Q	
PC	
SP	
SSP	
USP	
descrittore di Q.sp	
descrittore di Q.sp0	
descrittore di Q.stato	
RUNQUEUE	
CURR	
RB.LFT	

// subito prima del ritorno da *schedule*

// non di interesse