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 venerdì 24 gennaio 2020

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 (5 punti) _____

esercizio 3 (6 punti) _____

esercizio 4 (1 punti) _____

voto finale: (16 punti) _____

CON SOLUZIONI (in corsivo)

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 lonely
sem_t lazy, busy
int global = 0
```

```
void * bobby (void * arg) {
   mutex_lock (&lonely)
   sem_wait (&busy)
   global = 1                                   /* statement A */
   mutex_unlock (&lonely)
   sem_wait (&lazy)
   sem_post (&busy)                             /* statement B */
   sem_post (&lazy)
   return NULL
} /* end bobby */
```

```
void * tommmy (void * arg) {
   sem_post (&busy)
   mutex_lock (&lonely)
   global = 2                                   /* statement C */
   sem_wait (&lazy)
   sem_wait (&busy)
   sem_post (&lazy)
   mutex_unlock (&lonely)
   return (void * 3)
} /* end tommy */
```

```
void main ( ) {
   pthread_t th_1, th_2
   sem_init (&lazy, 0, 1)
   sem_init (&busy, 0, 0)
   create (&th_1, NULL, bobby, NULL)
   create (&th_2, NULL, tommy, NULL)
   join (th_2, &global)                         /* statement D */
   join (th_1, NULL)
   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 – bobby	**th_2 – tommy**
subito dopo stat. **A**	*ESISTE*	*ESISTE*
subito dopo stat. **B**	*ESISTE*	*ESISTE*
subito dopo stat. **C**	*PUÒ ESISTERE*	*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 supponga che il mutex valga 1 se occupato, e valga 0 se libero

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**		
	lonely	*lazy*	*busy*
subito dopo stat. **A**	*1*	*1*	*0*
subito dopo stat. **B**	*0 / 1*	*0*	*1*
subito dopo stat. **C**	*1*	*0 / 1*	*0 / 1*
subito dopo stat. **D**	*0 / 1*	*1*	*0*

Il sistema può andare in stallo (*deadlock*), con uno o più *thread* che si bloccano, in (almeno) **due casi diversi**. Si chiede di precisare il comportamento dei thread (in due casi), indicando gli statement dove avvengono i blocchi e i possibili valori della variabile *global*:

caso	**th_1 – bobby**	**th_2 – tommy**	*global*
1	***wait lazy***	***wait busy***	*2*
2	***wait busy***	**-**	*2 / 3*
3			

esercizio n. 2 – processi e nucleo

prima parte – gestione dei processi

```
// programma prova.c
main ( ) {
   pid1 = fork ( )              // P crea Q
   nanosleep (4)
   if (pid1 == 0) {             // codice eseguito da Q
      read (stdin, msg, 24)
      exit (-1)
   } else {
      pid2 = fork ( )           // P crea R
      if (pid2 == 0) {          // codice eseguito da R
         execl ("/acso/prog_x", "prog_x", NULL)
         exit (-1)
      } else {
         write (stdout, msg, 5)
         pid = wait (&status)    // P aspetta la terminazione di uno dei due figli
      } // end_if pid2
   } // end_if pid1
   exit (0)
} // prova
```

```
// programma prog_x.c
// dichiarazione e inizializzazione dei mutex presenti nel codice
// dichiarazione dei semafori presenti nel codice
```

void * **me** (void * arg) {		void * **you** (void * arg) {
mutex_lock (&far)		**mutex_lock** (&far)
sem_wait (&glance)		**sem_post** (&glance)
mutex_lock (&near)		**mutex_unlock** (&far)
mutex_unlock (&near)		**mutex_lock** (&near)
sem_post (&glance)		**sem_wait** (&glance)
mutex_unlock (&far)		**mutex_unlock** (&near)
return NULL		**return** NULL
} // me		} // you

```
main ( ) {  // codice eseguito da R
    pthread_t th_1, th_2
    sem_init (&glance, 0, 1)
    create (&th_1, NULL, me, NULL)
    create (&th_2, NULL, you, NULL)
    join (th_1, NULL)
    join (th_2, NULL)
    exit (1)
} // main
```

Un processo ***P*** esegue il programma ***prova*** e crea due un figli ***Q*** e ***R***; il figlio ***R*** esegue una mutazione di codice (programma ***prog_x***). La mutazione di codice va a buon fine e vengono creati i thread ***th_1*** e ***th_2***.

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, e tenendo conto che il processo ***R*** **non** ha ancora eseguito la **`execl`**. 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**	**P**	**Q**	**R**	*th_1*	*th_2*	
evento oppure *processo-chiamata*	*PID*	**1**	**2**	*3*	*4*	*5*	*6*	
	TGID	**1**	**2**	*3*	*4*	*4*	*4*	
Q – nanosleep (4)	**0**	**pronto**	**A write su stdout**	**A nanosleep**	**esec**	*NE*	*NE*	
5 interrupt da **std_out**, tutti i 5 caratteri richiesti trasferiti	1	*pronto*	*exec*	*A nano*	*pronto*	*NE*	*NE*	
interrupt da RT_clock e scadenza quanto di tempo	2	*pronto*	*pronto*	*A nano*	*exec*	*NE*	*NE*	
R – execl	3	*pronto*	*pronto*	*A nano*	*exec*	*NE*	*NE*	
R – create th_1	4	*pronto*	*pronto*	*A nano*	*exec*	*pronto*	*NE*	
interrupt da RT_clock e scadenza timer di nanosleep	5	pronto	pronto	esec	pronto	pronto	NE	
Q – read	6	*pronto*	*esec*	*A read*	*pronto*	*pronto*	*NE*	
P – wait	7	*pronto*	*A wait*	*A read*	*pronto*	*esec*	*NE*	
th_1 – lock (&far)	8	*pronto*	*A wait*	*A read*	*pronto*	*esec*	*NE*	
interrupt da RT_clock e scadenza quanto di tempo	9	*pronto*	*A wait*	*A read*	*esec*	*pronto*	*NE*	
R – create th_2	10	*pronto*	*A wait*	*A read*	*esec*	*pronto*	*pronto*	
24 interrupt da std_in, tutti i 24 caratteri richiesti trasferiti	11	pronto	A wait	esec	pronto	pronto	pronto	
Q – exit	12	*pronto*	*exec*	*NE*	*pronto*	*pronto*	*pronto*	
P – exit	13	*pronto*	*NE*	*NE*	*pronto*	*esec*	*pronto*	
interrupt da RT_clock e scadenza quanto di tempo	14	*pronto*	*NE*	*NE*	*pronto*	*pronto*	*esec*	

seconda parte – scheduling dei processi

Si consideri uno scheduler CFS con **3 task** caratterizzato da queste condizioni iniziali (**da completare**):

CONDIZIONI INIZIALI (da completare)							
RUNQUEUE	**NRT**	**PER**	**RQL**	**CURR**	**VMIN**		
	3	6	*4*	t1	100		
TASK	**ID**	**LOAD**	**LC**	**Q**	**VRTC**	**SUM**	**VRT**
CURRENT	t1	1	*0,25*	*1,5*	*1*	30	100,50
RB	t2	2	*0,50*	*3*	*0,5*	20	101
	t3	1	*0,25*	*1,5*	*1*	10	101,50

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

`Events of task t1: WAIT at 0,5; WAKEUP after 2,5;`

`Events of task t2: CLONE at 2;`

Simulare l'evoluzione del sistema per **4 eventi** riempiendo le seguenti tabelle (per indicare le condizioni di rescheduling della wakeup e della clone utilizzare le tabelle finali):

EVENTO		**TIME**	**TYPE**	**CONTEXT**	**RESCHED**		
		0,5	*wait*	*t1*	*true*		
RUNQUEUE	**NRT**	**PER**	**RQL**	**CURR**	**VMIN**		
	2	*6*	*3*	*t2*	*101*		
TASK	**ID**	**LOAD**	**LC**	**Q**	**VRTC**	**SUM**	**VRT**
CURRENT	*t2*	*2*	*0,67*	*4*	*0,5*	*20*	*101*
RB	*t3*	*1*	*0,33*	*2*	*1*	*10*	*101,5*
WAITING	*t1*	*1*				*30,5*	*101*

EVENTO		**TIME**	**TYPE**	**CONTEXT**	**RESCHED**		
		2,5	*clone*	*t2*	*false*		
RUNQUEUE	**NRT**	**PER**	**RQL**	**CURR**	**VMIN**		
	3	*6*	*5*	*t2*	*101,5*		
TASK	**ID**	**LOAD**	**LC**	**Q**	**VRTC**	**SUM**	**VRT**
CURRENT	*t2*	*2*	*0,4*	*2,4*	*0,5*	*22*	*102*
RB	*t3*	*1*	*0,2*	*1,20*	*1*	*10*	*101,5*
	t4	*2*	*0,4*	*2,4*	*0,5*	*0*	*102,7*
WAITING	*t1*	*1*				*30,5*	*101*

EVENTO		TIME	TYPE	CONTEXT	RESCHED		
		2,9	*Q_scade*	*t2*	*true*		
RUNQUEUE	**NRT**	**PER**	**RQL**	**CURR**	**VMIN**		
	3	*6*	*5*	*t3*	*101,5*		
TASK	**ID**	**LOAD**	**LC**	**Q**	**VRTC**	**SUM**	**VRT**
CURRENT	*t3*	*1*	*0,2*	*1,20*	*1*	*10*	*101,5*
RB	*t2*	*2*	*0,4*	*2,4*	*0,5*	*22,4*	*102,2*
	t4	*2*	*0,4*	*2,4*	*0,5*	*0*	*102,7*
WAITING	*t1*	*1*				*30,5*	*101*

EVENTO		TIME	TYPE	CONTEXT	RESCHED		
		3	*wakeup*	*t3*	*true*		
RUNQUEUE	**NRT**	**PER**	**RQL**	**CURR**	**VMIN**		
	4	*6*	*6*	*t1*	*101,6*		
TASK	**ID**	**LOAD**	**LC**	**Q**	**VRTC**	**SUM**	**VRT**
CURRENT	*t1*	*1*	*0,17*	*1*	*1*	*30,5*	*101*
RB	*t3*	*1*	*0,17*	*1*	*1*	*10,1*	*101,6*
	t2	*2*	*0,33*	*2*	*0,5*	*22,4*	*102,2*
	t4	*2*	*0,33*	*2*	*0,5*	*0*	*102,7*
WAITING							

Condizioni di rescheduling a **wake_up** del **task t1**:

wake_up: *101 + 1 x 0,17 = 101,17 < 101,6 ? → true*

Condizioni di rescheduling a **clone** del **task t2**:

clone: *102,7 + 1 x 0,4 = 103,1 < 102 ? → false*

esercizio n. 3 – memoria e file system

prima parte – gestione dello spazio di memoria

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

MAXFREE = **3** **MINFREE** = **2**

situazione iniziale (esiste un processo P)

```
PROCESSO: P  ********************************************************
    VMA : C  000000400,  2 , R  ,  P  ,  M  , <X,0>
          S  000000600,  1 , W  ,  P  ,  M  , <X,2>
          D  000000601,  2 , W  ,  P  ,  A  , <-1,0>
          P  7FFFFFFB,  4 , W  ,  P  ,  A  , <-1,0>

    PT: <c0 :- -> <c1 :1 R> <s0 :4 W> <d0 :- -> <d1 :- -> <p0 :s0 W>
        <p1 :5 W> <p2 :2 W> <p3 :- ->

process P - NPV of PC and SP:  c1, p2

____MEMORIA FISICA____(pagine libere: 3)_____________________________
      00 : <ZP>                ||  01 : Pc1 / <X,1>          ||
      02 : Pp2                 ||  03 : ----                 ||
      04 : Ps0                 ||  05 : Pp1                  ||
      06 : ----                ||  07 : ----                 ||

____STATO del TLB_____________________________________________________
      Pc1 :  01 -  0: 1:   ||   Pp2 : 02 -  1: 0:    ||
      Ps0 :  04 -  1: 0:   ||   Pp1 : 05 -  1: 0:    ||
            -----          ||         -----          ||
            -----          ||         -----          ||

SWAP FILE:     Pp0  , ----, ----, ----, ----, ----
LRU ACTIVE:    PC1
LRU INACTIVE:  pp2, pp1, ps0
```

evento 1 e 1 bis: *read* (Pd0), *write* (Pp3, Pd1)

Pagina Pd0 in ZP con COW. La pagina Pp3 è di growsdown, quindi modifica di VMA P e poi allocazione in 03. La pagina Pd1 attiva PFRA, liberate 04 e 05 e scrittura in swapfile, pagina Pd1 in 04. Le liste LRU sono aggiornate di conseguenza.

PT del processo: **P**				
s0: *s1* *W*	d0: *0* *R*	d1: *4* *W*		
p0: s0 W	p1: *s2* *W*	p2: *2* *W*	p3: *3* *W*	p4: - -

MEMORIA FISICA	
00: *Pd0 /* <ZP>	01: *Pc1 / <X, 1>*
02: *Pp2*	03: *Pp3*
04: *Pd1*	05:
06:	07:

SWAP FILE	
s0: *Pp0*	s1: *Ps0*
s2: *Pp1*	s3:
s4:	s5:

LRU ACTIVE: *PD1, PP3, PD0, PC1* _____

LRU INACTIVE: *pp2*, _____

evento 2: *sbrk* (2)

VMA del processo **P** (compilare solo le righe relativa alle VMA **D** e **P**)							
AREA	NPV iniziale	dimensione	R/W	P/S	M/A	nome file	offset
D	*0000 0060 1*	*4*	*W*	*P*	*A*	*-1*	*0*
P	*7FFF FFFF A*	*5*	*W*	*P*	*A*	*-1*	*0*

evento 3: *read* (Pc1) – 4 *kswapd*

Nota bene: le pagine PD1, PP3, PD0 e PC1 hanno inizialmente bit A = 1.

LRU ACTIVE: *PC1* _____

LRU INACTIVE: *pd1, pp3, pd0, pp2*, _____

evento 4: *write* (Pd2, Pd3)

Pagina Pd2 in 05, Pd3 causa PFRA che libera le pagine 02 (PP2 in swapfile) e 03 (PP3 in swapfile). NB: la pagina Pd0 è in ZP, e non può essere liberata. Pd3 in 02. Aggiornamento liste LRU.

MEMORIA FISICA	
00: *Pd0 /* <ZP>	01: *Pc1 / <X, 1>*
02: *Pd3*	03: -----
04: *Pd1*	05: *Pd2*
06:	07:

SWAP FILE	
s0: *Pp0*	s1: *Ps0*
s2: *Pp1*	s3: *Pp2*
s4: *Pp3*	s5:

evento 5: *read* (Pp2)

Swapin di pagina Pp2 in 03, con COW abilitato.

PT del processo: **P**				
p0: s0 W	p1: *s2 W*	p2: *3 R*	p3: *s4 W*	p4: - -

process P – NPV of PC and SP: *c1, p2*

SWAP FILE	
s0: *Pp0*	s1: *Ps0*
s2: *Pp1*	s3: *Pp2*
s4: *Pp3*	s5:

LRU ACTIVE: *PP2, PD3, PD2, PC1* _____

LRU INACTIVE: *pd1, pd0,* _____

seconda parte – file system

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

MAXFREE = **2** **MINFREE** = **1**

Si consideri la seguente **situazione iniziale**:

```
____MEMORIA FISICA____(pagine libere: 3)_____________________________
      00 : <ZP>                ||  01 : Pc2 / <X,2>          ||
      02 : Pp0                 ||  03 : <G,2>                ||
      04 : Pm00                ||  05 : ----                 ||
      06 : ----                ||  07 : ----                 ||

____STATO del TLB_____________________________________________________
      Pc2  : 01 -  0: 1:   ||   Pp0 : 02 -  1: 1:    ||
      Pm00 : 04 -  1: 1:   ||         -----          ||
            -----          ||         -----          ||
```

Per ognuno dei seguenti eventi compilare le tabelle richieste con i dati relativi al contenuto della memoria fisica, delle variabili del FS relative al file F e al numero di accessi a disco effettuati in lettura e in scrittura.

È in esecuzione il processo **P**.

ATTENZIONE: il numero di pagine lette o scritte di un file è cumulativo, quindi è la somma delle pagine lette o scritte su quel file da tutti gli eventi precedenti oltre a quello considerato. Si ricorda che *close* scrive le pagine dirty di un file solo se *f_count* diventa = 0.

eventi 1 e 2: fd = *open* (“F”), *write* (fd, 6000)

MEMORIA FISICA	
00: <ZP>	01: *Pc2 / <X, 2>*
02: *Pp0*	03: *<G, 2>*
04: *Pm00*	05: *<F, 0> D*
06: *<F, 1> D*	07: ----

	f_pos	**f_count**	**numero pag. lette**	**numero pag. scritte**
file F	*6000*	*1*	*2*	*0*

evento 3: *close* (fd)

MEMORIA FISICA	
00: <ZP>	01: *Pc2 / <X, 2>*
02: *Pp0*	03: *<G, 2>*
04: *Pm00*	05: *<F, 0>*
06: *<F, 1>*	07: ----

	f_pos	**f_count**	**numero pag. lette**	**numero pag. scritte**
file F	--	*0*	*2*	*2*

eventi 4 e 5: *fork* (“Q”), *context switch* (“Q”)

Viene invocato PFRA che libera le pagine fisiche 03 e 05, entrambe di page cache, poi viene allocata la pagina fisica 03 a PP0. Le pagine di P vengono messe a D anche a causa dello svuotamento del TLB.

MEMORIA FISICA	
00: <ZP>	01: *Pc2 /Qc2 / <X, 2>*
02: *Qp0 D*	03: *Pp0 D*
04: *Pm00 / Qm00 D*	05: *----*
06: *<F, 1>*	07: *----*

eventi 6 e 7: fd = *open* (“F”), *read* (7000)

MEMORIA FISICA	
00: <ZP>	01: *Pc2 / Qc2 / <X, 2>*
02: *Qp0 D*	03: *Pp0 D*
04: *Pm00 / Qm00 D*	05: *<F, 0>*
06: *<F, 1>*	07: *----*

	f_pos	f_count	numero pag. lette	numero pag. scritte
file F	*7000*	*1*	*3*	*2*

eventi 8 e 9: fd1 = *open* (“H”), *write* (fd1, 3000)

Viene invocato PFRA che libera le pagine fisiche 05 e 06, entrambe di page cache, poi viene allocata la pagina fisica 05 a <H, 0>).

MEMORIA FISICA	
00: <ZP>	01: *Pc2 / Qc2 / <X, 2>*
02: *Qp0 D*	03: *Pp0 D*
04: *Pm00 / Qm00 D*	05: *<H, 0> D*
06: *----*	07: *----*

	f_pos	f_count	numero pag. lette	numero pag. scritte
file F	*7000*	*1*	*3*	*2*
file H	*3000*	*1*	*1*	*0*

eventi 10 e 11: *close* (fd), *close* (fd1)

	f_pos	f_count	numero pag. lette	numero pag. scritte

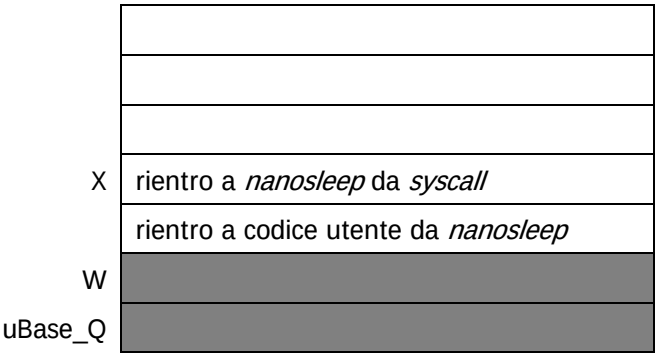
file F	----	*0*	*3*	*2*
file H	----	*0*	*1*	*1*

esercizio n. 4 – moduli del SO

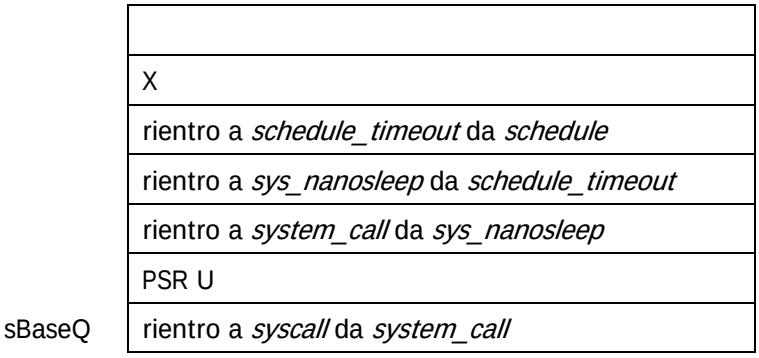
Sono dati due processi **P** e **Q**, dove P è il processo padre di Q. Non ci sono altri processi utente nel sistema. Lo stato iniziale delle pile di sistema e utente dei due processi è riportato qui sotto.

uStack_P – iniziale

sStack_P – iniziale



uStack_Q – iniziale

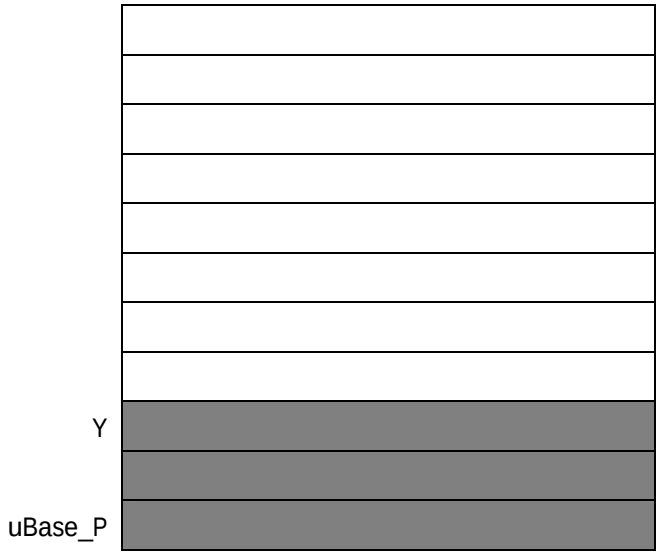


sStack_Q – iniziale

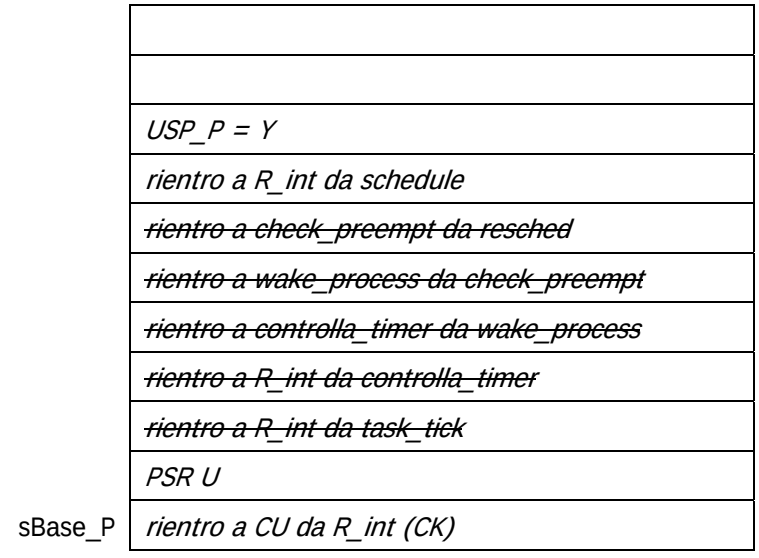
Si consideri il seguente **evento**:

Interrupt da Real Time Clock con *time_out* scaduto. Il risveglio del processo in attesa di *time_out* **comporta** l'attivazione di *resched ( )*.

Si mostrino le invocazioni di tutti i **moduli** (e eventuali relativi ritorni) fino alla gestione completa dell'evento. Si mostri inoltre lo stato finale delle pile del processo **P** al termine della gestione dell'evento.



uStack_P – finale



sStack_P – finale

invocazione moduli (num. di righe vuote non signif.)

tabella di invocazione dei moduli		
processo	**modo**	**modulo**
P	*U – S*	*> R_int (CK)*
P	*S*	*> task_tick <*
P	*S*	*> controlla_timer*
P	*S*	*> wakeup_process*
P	*S*	*> check_preeempt_curr*
P	*S*	*> resched (TNR = 1) <*
P	*S*	*check_preeempt_curr <*
P	*S*	*wakeup_process <*
P	*S*	*controlla_timer <*
P	*S*	*R_int (CK)*
P	*S*	*> schedule*
P	*S*	*> pick_next_task <*
P – Q	*S*	*schedule: context switch*
Q	*S*	*schedule <*
Q	*S*	*schedule_timeout <*
Q	*S*	*sys_nanosleep <*
Q	*S – U*	*system_call: SYSRET*
Q	*U*	*syscall <*
Q	*U*	*nanosleep <*
Q	*U*	*codice utente*

spazio libero per continuazione o brutta copia