2022

Esercizi misti ACSO

IN PREPARAZIONE ALL'ESAME

Esercizio 1 - esame del 29 agosto 2017

Si consideri il programma C seguente (gli "#include" e le inizializzazioni dei mutex sono omessi):

```
pthread_mutex_t stop
sem_t min, max
int global = 0
```

```
void * put (void * arg) {
   pthread_mutex_lock (&stop)
   global = 1                                   /* statement A */
   sem_wait (&min)
   pthread_mutex_unlock (&stop)
   sem_wait (&min)
   sem_post (&max)                              /* statement B */
   return NULL
} /* end put */
```

```
void * get (void * arg) {
   if (arg == 2) global = 2                     /* statement C */
   pthread_mutex_lock (&stop)
   sem_post (&min)
   sem_wait (&max)
   pthread_mutex_unlock (&stop)
   return arg
} /* end get */
```

```
void main ( ) {
   pthread_t th_1, th_2, th_3
   sem_init (&min, 0, 0)
   sem_init (&max, 0, 1)
   pthread_create (&th_1, NULL, put, NULL)
   pthread_create (&th_2, NULL, get, 2)
   pthread_create (&th_3, NULL, get, 3)
   pthread_join (th_1, NULL)
   pthread_join (th_2, NULL)                    /* statement D */
   pthread_join (th_3, &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 – put	**th_2 – get**	**th_3 – get**
subito dopo stat. **A**			
subito dopo stat. **C** in **th_3**			
subito dopo stat. **D**			

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)
- una variabile mutex assume valore 0 per mutex libero e valore 1 per mutex occupato

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.

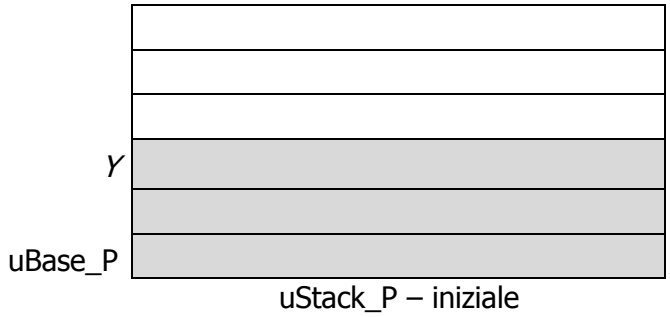
condizione	**variabili globali**		
	min	*max*	*global*
subito dopo stat. **A**			
subito dopo stat. **B**			
subito dopo stat. **C** in **th_3**			

Il sistema può andare in stallo (*deadlock*), con uno o più *thread* che si bloccano, in **tre 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:

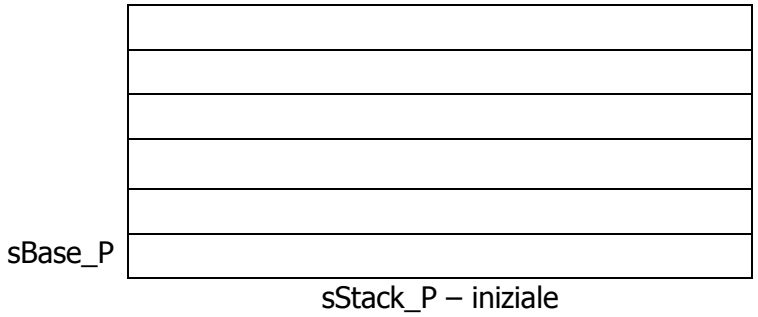
caso	**th_1 – put**	**th_2 – get**	**th_3 – get**
1			
2			
3			

Esercizio 2 – nucleo - esame del 12 febbraio 2019

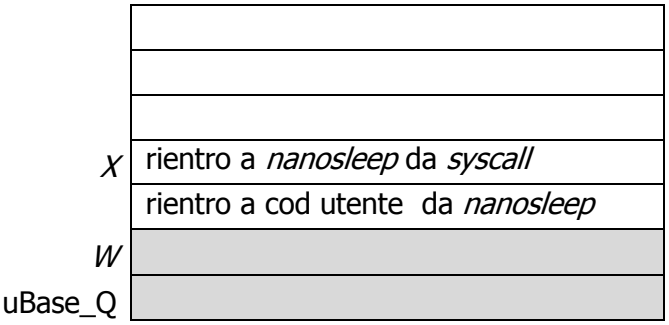
Sono dati due processi P e Q, dove P è il processo padre di Q. Non ci sono altri processi utente nel sistema. Lo stato iniziale delle pile di sistema e utente dei due processi è riportato qui sotto.



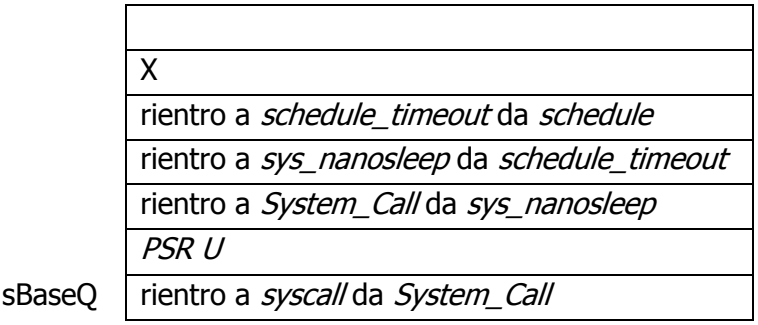
uStack_P – iniziale



sStack_P – iniziale



uStack_Q – iniziale



sStack_Q – iniziale

Si considerino due eventi, **Evento 1** e **Evento 2**, che si verificano in successione.

Evento 1: Interrupt da Real Time Clock con *time_out* scaduto. Il risveglio del processo in attesa di *time_out* non comporta l'attivazione di *resched ( )*.

Evento 2: il processo correntemente in esecuzione invoca **exit ( )**

Per gli eventi indicati si compili la **tabella di invocazione dei moduli** della pagina successiva, riempiendola in successione con le invocazioni relative a **Evento 1** seguite da quelle relative a **Evento2**.

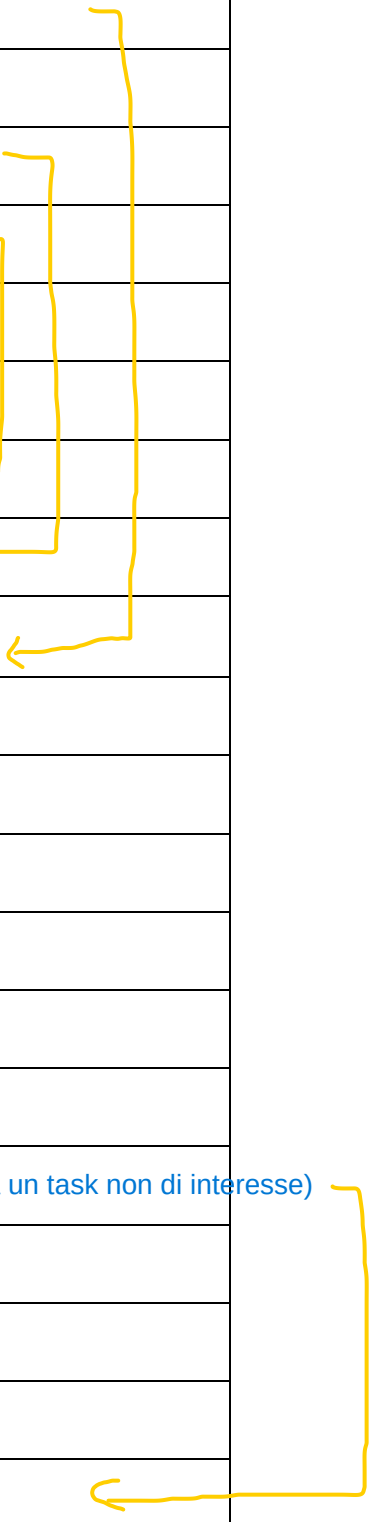
Per la compilazione si segua lo schema usuale, **mostrando** le invocazioni di tutti i **moduli** (e eventuali relativi ritorni) e precisando processo e modo. La simulazione delle invocazioni dei moduli deve arrivare fino al *context switch* del processo che esegue la *exit*, ma non oltre.

NOTAZIONE da usare per i moduli: > (invocazione), nome_modulo (esecuzione), < (ritorno)

Evento 1 ed Evento 2

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

Tabella di invocazione dei moduli		
processo	**modo**	**modulo**
P	U -> S	> R_int_clock
P	S	> task_tick <
P	S	> Controlla_timer
P	S	> wake_up_process
P	S	> enqeue_task <
P	S	> check_preempt_curr <
P	S	wake_up_process <
P	S	Controlla_timer <
P	S -> U	IRET (R_int_clock <)
P	U	Codice utente di P
P	U	> exit
P	U	> syscall
P	U -> S	SYSCALL (> system_call)
P	S	> sys_exit_group
P	S	> sys_exit
P	S	> wake_up_process (Risveglia un task non di interesse)
P	S	> check_preempt_curr
P	S	> resched <
P	S	check_preempt_curr <
P	S	wake_up_process <
P	S	> schedule
P	S	> pick_next_task <
P -> Q	S	CONTEXT_SWITCH (schedule)



Esercizio 2 – scheduler - esame del 13 settembre 2019

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	10	100
RB	t2	2	0.5	3	0.5	20	100.8
	t3	1	0.25	1.5	1	30	101

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

```
Events of task t1:  WAIT at 0.5; WAKEUP after 6;
Events of task t2:  WAIT at 0.5; WAKEUP after 1;
Events of task t3:  EXIT at 0.5
```

Simulare l'evoluzione del sistema per **4 eventi** riempiendo le seguenti tabelle.

Indicare la valutazione delle condizioni di preemption per l'evento di WAKEUP nell'apposito spazio alla fine dell'esercizio.

EVENTO		**TIME**	**TYPE**	**CONTEXT**	**RESCHED**	
		0.5	WAIT	T1	TRUE	T1 -> VRT = 100 + 0.5 * 1 = 100.5
RUNQUEUE	**NRT**	**PER**	**RQL**	**CURR**	**VMIN**	
	2	6	3	T2	100.5	
TASK	**ID**	**LOAD**	**LC**	**Q**	**VRTC**	**SUM**	**VRT**
CURRENT	T2	2	2/3	4	0.5	20	100.8
RB	T3	1	1/3	2	1	30	101
WAITING	T1	1				10.5	100.5

EVENTO		TIME	TYPE	CONTEXT	RESCHED		
		1	WAIT	T2	TRUE		
RUNQUEUE	NRT	PER	RQL	CURR	VMIN		
	1	6	1	T3	101		
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT	T3	1	1	6	1	30	101
RB							
WAITING	T1	1				10.5	100.5
	T2	2				20.5	101.05

T2 -> VRT = 100.8 + 0.5 * 0.5 = 101.05

EVENTO		TIME	TYPE	CONTEXT	RESCHED		
		1.5	EXIT	T3	TRUE		
RUNQUEUE	NRT	PER	RQL	CURR	VMIN		
	- - - -	- - - -	- - - -	IDLE	101.5		
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT	IDLE						
RB							
WAITING	T1	1				10.5	100.5
	T2	2				20.5	101.05

T3 -> VRT = 101 + 0.5 * 1 = 101.5

EVENTO		TIME	TYPE	CONTEXT	RESCHED		
		2	WAKE UP	IDLE	TRUE		
RUNQUEUE	NRT	PER	RQL	CURR	VMIN		
	1	6	2	T2	101.5		
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT	T2	2	1	6	0.5	20.5	101.05
RB							
WAITING	T1	1				10.5	100.5

T2 -> VRT = 101.05

WAKE UP => RESCHED è TRUE perchè c'è solo IDLE

Esercizio 2 – stato - esame del 7 luglio 2017

```
// programma prova.c
main ( ) {
   pid1 = fork ( )
   if (pid1 == 0) {          // codice eseguito dal figlio Q
      execl ("/acso/prog_x", "prog_x", NULL)
      exit (-1)
   } else {     // codice eseguito dal padre P
      write (fd, vet, 5)         // scrittura di 1 blocco su file precedentemente aperto
      pid1 = waitpid (pid1, ...)
      exit (0)
   } // if
}  // prova
```

```
// programma prog_X.c
int num
pthread_mutex_t DOOR= PTHREAD_MUTEX_INITIALIZER
sem_t CHECK
```

void * UNO (void * arg) {		void * DUE (void * arg) {
sem_wait (&CHECK)		if (num > 0) {
pthread_mutex_lock (&DOOR)		pthread_mutex_lock (&DOOR)
sem_wait (&CHECK)		sem_post (&CHECK)
pthread_mutex_unlock (&DOOR)		pthread_mutex_unlock (&DOOR)
return NULL		} else {
} // UNO		sem_post (&CHECK) }
		return NULL
		} // DUE

```
main ( ) {  // codice eseguito da Q
    pthread_t TH_1, TH_2
    sem_init (&CHECK, 0, 0)
    // viene letto da standard input il valore di num
    pthread_create (&TH_1, NULL, UNO, NULL)
    pthread_create (&TH_2, NULL, DUE, NULL)
    pthread_join (TH_2, NULL)
    pthread_join (TH_1, NULL)
    exit (1)
}  // main
```

Un processo ***P*** esegue il programma ***prova*** creando il figlio ***Q***, che esegue una mutazione di codice che va a buon fine. Nel codice mutato ***prog_X***, ***Q*** crea i thread ***TH_1*** e ***TH_2***. Si ipotizzi che il valore di ***num*** letto da ***Q*** sia maggiore di 0.

Si simuli l'esecuzione dei processi (fino a ***udt*** = **160**) così come risulta dal codice dato, dagli eventi indicati e facendo bene attenzione allo stato iniziale considerato per la simulazione. **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 del tempo indicato**; 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	TH1	TH2		
	PID	**1**	2	3	4	5		
evento/processo-chiamata	*TGID*	**1**	2	3	3	3		
	0	**Pronto**	**ESEC**	**NON ESISTE**	NE	NE		
P - fork	10	pronto	ESEC	pronto	NE	NE		
P - write	20	pronto	A write	ESEC	NE	NE		
Q - execl	30	pronto	A write	ESEC	NE	NE		
interrupt da RT_clock, scadenza quanto di tempo	40	pronto	A write	ESEC	NE	NE		
Interrupt da STDOUT, tutti i dati scritti	50	pronto	ESEC	Pronto	Non esiste	Non esiste		
interrupt da RT_clock, scadenza quanto di tempo	60	pronto	pronto	ESEC	NE	NE		
Q - pthread_create(TH1)	70	pronto	pronto	ESEC	pronto	NE		
interrupt da RT_clock, scadenza quanto di tempo	80	pronto	ESEC	pronto	pronto	NE		
P - wait	90	pronto	Attesa	Pronto	ESEC	Non esiste		
TH1 - sem_wait(&CHECK)	100	pronto	A wait	ESEC	A sem	NE		
Q - pthread_create(TH2)	110	pronto	A wait	ESEC	A sem	pronto		
Q - join(TH2)	120	pronto	A wait	A join	A sem	ESEC		
TH2 - mutex_lock(&DOOR)	130	pronto	A wait	A join	A sem	ESEC		
TH2 - sem_post(&CHECK)	140	pronto	Attesa	Attesa	ESEC	Pronto		
TH1 - mutex_lock(&DOOR)	150	pronto	A wait	A join	A lock	ESEC		
TH2 - mutex_unlock(&DOOR)	160	pronto	A wait	A join	ESEC	pronto		

esercizio n. 3 – memoria

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

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

situazione iniziale (esiste un solo processo P)

```
PROCESSO: P  ********************************************************
    VMA : C  000000400,  2 , R  ,  P  ,  M  , <X, 0>
          S  000000600,  2 , W  ,  P  ,  M  , <X, 2>
          P  7FFFFFFC,  3 , W  ,  P  ,  A  , <-1,0>
    PT: <c0 :1 R>  <c1 :-  ->  <s0 :- ->  <s1 :- ->  <p0 :2 W>
        <p1 :- ->  <p2 :-  ->
    process P - NPV of PC and SP:  c0, p0

____MEMORIA FISICA____(pagine libere: 5)_____________________________
      00 : <ZP>                 ||  01 : Pc0 / <X, 0>         ||
      02 : Pp0                  ||  03 : ----                 ||
      04 : ----                 ||  05 : ----                 ||
      06 : ----                 ||  07 : ----                 ||

SWAP FILE:     ----, ----, ----, ----, ----, ----,
LRU ACTIVE:    PP0, PC0
LRU INACTIVE:
```

evento 1: ***read*** **(Pc0, Ps0),** ***write*** **(Pp0, Pp1)**

PT del processo: **P**				
c0: :1 R	c1: :- -	s0: :3 R	s1: :- -	p0: :2 W
p1: :4 W	p2: :- -			

process P	NPV of **PC**: c0	NPV of **SP**: p1

MEMORIA FISICA	
00: <ZP>	01: Pc0 / <X, 0>
02: Pp0	03: Ps0 / <X, 2>
04: Pp1	05:
06:	07:

LRU ACTIVE: PP1, PS0, PP0, PC0

LRU INACTIVE: _____

evento 2: ***read*** **(Pc0, Ps1) –** ***write*** **(Pp1) – 4** ***kswapd***

PT del processo: **P**				
c0: :1 R	c1: :- -	s0: :3 R	s1: :5 R	p0: :2 W
p1: :4 W	p2: :- -			

MEMORIA FISICA	
00: <ZP>	01: Pc0 / <X, 0>
02: Pp0	03: Ps0 / <X, 2>
04: Pp1	05: Ps1 / <X, 3>
06:	07:

LRU ACTIVE: PS1, PP1, PC0

LRU INACTIVE: ps0, pp0

evento 3: ***mmap*** **(0x000000002000000, 2, W, S, M, "F", 0)** **VMA M0**

VMA del processo **P** (è da compilare solo la riga relativa alla VMA **M0**)							
AREA	NPV iniziale	dimensione	R/W	P/S	M/A	nome file	offset
M0	0x 0 0000 2000	2	W	S	M	F	0

evento 4: ***read*** **(Pc0, Pm00)**

PT del processo: **P**				
p1: :4 W	p2: :- -	m00: :6 W	m01: :- -	

MEMORIA FISICA	
00: <ZP>	01: Pc0 / <X, 0>
02: Pp0	03: Ps0 / <X, 2>
04: Pp1	05: Ps1 / <X, 3>
06: Pm00 / <F, 0>	07:

LRU ACTIVE: PM00, PS1, PP1, PC0

LRU INACTIVE: ps0, pp0

evento 5: ***clone*** **(R, c1)**

PT del processo: **P**				
t00: :2 W	t01: :- -	m00: :6 W	m01: :- -	

process R	NPV of **PC**: c1	NPV of **SP**: t00

MEMORIA FISICA	
00: <ZP>	01: PRc0 / <X, 0>
02: PRt00	03: - - - -
04: Prp1	05: PRs1 / <X, 3>
06: Pm00 / <F, 0>	07:

SWAP FILE	
s0: Pp0	s1:

LRU ACTIVE: PRT00, PRM00, PRS1, PRP1, PRC0

LRU INACTIVE:

evento 6: **4** ***kswapd*** ACTIVE: INACTIVE: prt00, prm00, prs1, prp1, prc0

evento 7: ***context_switch*** **(R),** ***read*** **(Rm01)**

MEMORIA FISICA	
00: <ZP>	01: PRm01 / <F, 1>
02: PRt00 (D)	03: PRc1 / <X, 1>
04: - - - -	05: PRs1 / <X, 3>
06: Pm00 / <F, 0>	07:

SWAP FILE	
s0: Pp0	s1: PRp1

ACTIVE: PRM01, PRC1

INACTIVE: prt00, prm00, prs1

Esercizio 3 – memoria - esame del 12 febbraio 2019

prima parte – memoria

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

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

Situazione iniziale (esistono due processi P e Q)

```
PROCESSO: P ********************************************************
    VMA :    C  000000400,  2 , R  ,  P  ,  M  , <X,0>
             S  000000600,  2 , W  ,  P  ,  M  , <X,2>
             P  7FFFFFFFB,  4 , W  ,  P  ,  A  , <-1,0>
    PT: <c0 :1  R>  <c1 :- ->  <s0 :3 R>  <s1 :- ->  <p0 :2 R>
        <p1 :5  W>  <p2 :6 W>  <p3 :- ->
    process P - NPV of PC and SP: c0, p1
PROCESSO: Q ********************** (non interessa) **********************
```

```
____MEMORIA FISICA____(pagine libere: 2)_____________________________
      00 : <ZP>             ||  01 : Pc0 / Qc0 / <X,0>    ||
      02 : Pp0 / Qp0        ||  03 : Ps0 / Qs0 / <X,2>    ||
      04 : ----             ||  05 : Pp1                  ||
      06 : Pp2              ||  07 : ----                 ||
```

```
____STATO del TLB_____________________________________________________
      Pc0 : 01 -  0: 1:     ||   Pp0 : 02 -  1: 0:        ||
      Ps0 : 03 -  0: 0:     ||   Pp1 : 05 -  1: 0:        ||
      Pp2 : 06 -  1: 0:     ||         -----              ||
```

SWAP FILE: `Qp1 , ----, ----, ----, ----, ----, ----, ----,`

LRU ACTIVE: `PC0,`
LRU INACTIVE: `pp1, pp2, pp0, ps0, qs0, qp0, qc0,`

evento 1: write (Ps0)

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <X, 0>
02: Pp0 / Qp0	03: Qs0 / <X, 2>
04: Ps0	05: Pp1
06: Pp2	07:

SWAP FILE	
s0: Qp1	s1:
s2:	s3:

ACTIVE: PC0

INACTIVE: pp1, pp2, pp0, ps0, qs0, qp0, qc0

ACTIVE: PS1, PC0

INACTIVE: pp1, pp2, pp0, qp0, qc0

evento 2: read (Ps1)

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <X, 0>
02: Pp0 / Qp0	03: Ps1 / <X, 3>
04: - - - -	05: Pp1
06: Pp2	07:

SWAP FILE	
s0: Qp1	s1: Ps0
s2:	s3:

evento 3: clone (R, c1)

VMA del processo **P/R** (compilare solo la riga relativa alla nuova VMA creata)							
AREA	NPV iniziale	dimensione	R/W	P/S	M/A	nome file	offset
T0	0x 7FFF F77F E	2	W	P	A	-1	0

ACTIVE: PRT00, PRS1, PRC0

INACTIVE: prp1, prp2, prp0, qp0, qc0

process R - NPV of PC and SP: c1, t00

MEMORIA FISICA	
00: <ZP>	01: PRc0 / Qc0 / <X, 0>
02: PRp0 / Qp0	03: PRs1 / <X, 3>
04: PRt00	05: Prp1
06: PRp2	07:

SWAP FILE	
s0: Qp1	s1: Ps0
s2:	s3:

evento 4: context switch (R)

MEMORIA FISICA	
00: <ZP>	01: PRc0 / Qc0 / <X, 0>
02: PRc1 / <X, 1>	03: PRs1 / <X, 3>
04: PRt00 (D)	05: PRp1 (D)
06: - - - -	07:

SWAP FILE	
s0: Qp1	s1: Ps0
s2: PRp0 / Qp0	s3: PRp2

ACTIVE: PRC01, PRT00, PRS1, PRC0

INACTIVE: prp1, qc0

Esercizio 3 – fs - esame del 29 agosto 2019

È dato un sistema di memoria caratterizzato dai seguenti parametri generali: **MAXFREE** = **3**, **MINFREE** = **1**, **dimensione memoria disponibile = 6 pagine**.
Nel sistema è attivo solamente un processo P.

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: **alcune Tabelle sono PARZIALI – riempire in tal caso solamente le celle indicate**

Evento 1: il processo P esegue exec (3, 1, 4, 2, c2, "X")

(si consultino gli eventi exec e mmap sul fascicolo per il significato dei parametri della exec e delle colonne nella tabella delle VMA)

VMA del processo **P**							
AREA	NPV iniziale	dimensione	R/W	P/S	M/A	nome file	offset
C	0x 0000 0040 0	3	R	P	M	X	0
K	0x 0000 0060 0	1	R	P	M	X	3
S	0x 0000 0060 1	4	W	P	M	X	4
D	0x 0000 0060 5	2	W	P	A	-1	0
P	0x 7FFF FFFF C	3	W	P	A	-1	0

PT del processo: **P** **(parziale: compilare solo le pagine indicate)**				
c0: :- -	c2: :1 R	s0: :- -	p0: :2 W	p2: :- -

NPV del PC: c2 NPV dello SP: p0

MEMORIA FISICA	
00: <ZP>	01: Pc2 / <X, 2>
02: Pp0	03:
04:	05:

TLB							
NPV	NPF	D	A	NPV	NPF	D	A
Pc2	1	0	1	Pp0	2	1	1

Evento 2: clone (Q, c0)

(si ricorda che:

- il primo parametro è il nome del processo che viene creato e il secondo parametro è la pagina della thread function
- la pila di un thread è di 2 pagine (nel nostro modello)

VMA del processo **P/Q** **(inserire solo le VMA nuove o modificate)**							
AREA	NPV iniziale	dimensione	R/W	P/S	M/A	nome file	offset
T0	0x 7FFF F77F E	2	W	P	A	-1	0

PT del processo: **P** **(inserire solo pagine create o modificate dall'evento clone)**				
<t00 :3 W>	<t01 :- ->			

NPV del PC del processo Q: c0 NPV dello SP del processo Q: t00

MEMORIA FISICA	
00: <ZP>	01: PQc2 / <X, 2>
02: PQp0	03: PQt00
04:	05:

TLB							
NPV	NPF	D	A	NPV	NPF	D	A
PQc2	1	0	1	PQp0	2	1	1
PQt00	3	1	1				

Esercizio 3 – fs - esame del 4 luglio 2019

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

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

Si consideri la seguente **situazione iniziale**:

```
____MEMORIA FISICA____(pagine libere: 5)_____________________________
      00 : <ZP>                 ||  01 : Pc0 / <X, 0>        ||
      02 : Pp0                  ||  03 : ----                ||
      04 : ----                 ||  05 : ----                ||
      06 : ----                 ||  07 : ----                ||
```

Per ciascuno 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**. La pagina in cima alla pila è **Pp0**.

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.

eventi 1 e 2 – fd = *open* (F), *read* (fd, 12000)

MEMORIA FISICA	
00: <ZP>	01: Pc0 / <X, 0>
02: Pp0	03: <F, 0>
04: <F, 1>	05: <F, 2>
06:	07:

	f_pos	**f_count**	**numero pagine lette**	**numero pagine scritte**
file F	12000	1	3	0

0 ---- 4096 ---- 8192 ---- 12288 ---- 16384 ---- 20480 ---- 24576
0 1 2 3 4 5

eventi 3-5 – *fork* (Q), *lseek* (fd, –10000), *write* (fd, 10)

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <X, 0>
02: Qp0 (D)	03: Pp0
04: <F, 0> (D)	05: <F, 2>
06:	07:

	f_pos	f_count	numero pagine lette	numero pagine scritte
file F	2010	2	4	0

eventi 6-9 – fd1 = *open* (G), *write* (fd1, 7000), *close* (fd), *close* (fd1)

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <X, 0>
02: Qp0 (D)	03: Pp0
04: <G, 0>	05: <G, 1>
06:	07:

	f_pos	f_count	numero pagine lette	numero pagine scritte
file F	2010	1	4	1
file G	- - - -	0	2	2

eventi 10 e 11 – *context switch* (Q), *write* (fd, 100)

	f_pos	f_count	numero pagine lette	numero pagine scritte
file F	2110	1	5	1

Esercizio 4 – fs1 - esame del 12 febbraio 2019

Si riportano nel seguito gli elementi di interesse di alcune `struct` necessarie alla gestione, organizzazione e accesso di file e cataloghi.

`struct task_struct { ……` `  struct files_struct *files ……}`	Ogni istanza rappresenta un descrittore di processo.

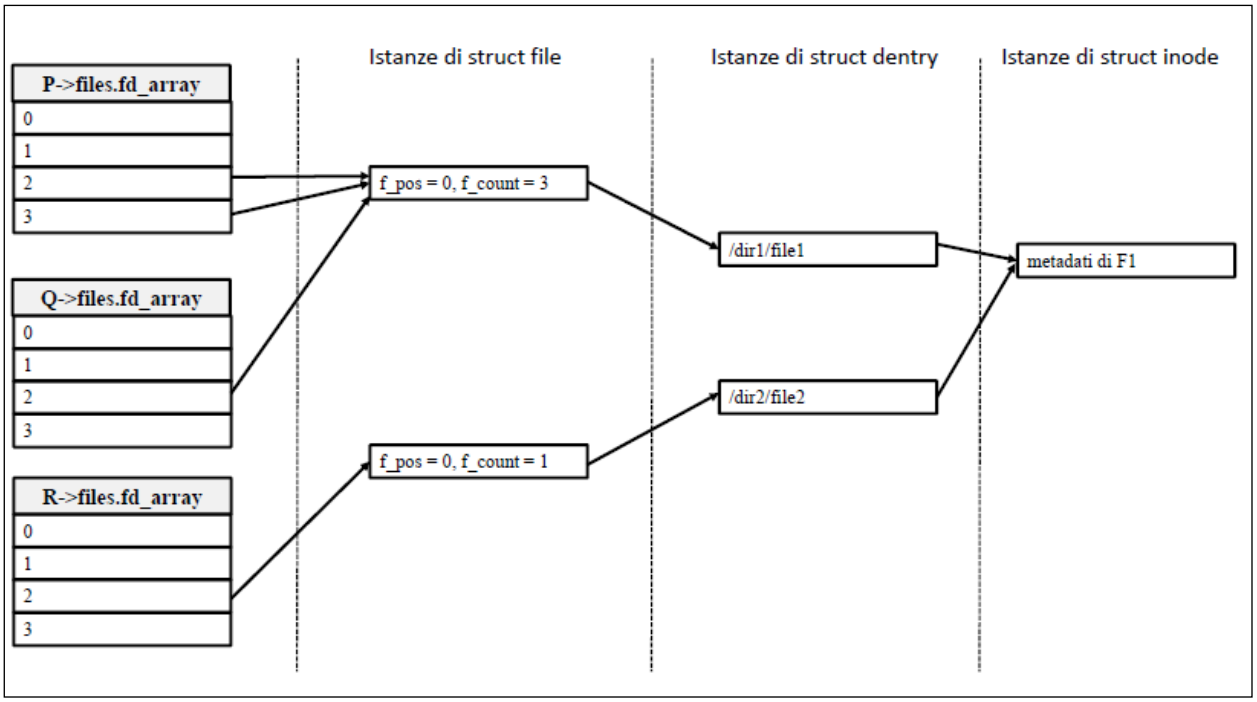
`struct files_struct { ……` `  struct file *fd_array [NR_OPEN_DEFAULT] ……}`	`fd_array[]` costituisce la tabella dei (descrittori) dei file aperti da un processo.

`struct file { ……`
 `  struct dentry *f_dentry`
 `  off_t f_pos  //posizione corrente`
 `  f_count  //contatore riferim. al file aperto ……}`
	Ogni istanza rappresenta un file aperto nel sistema.

`struct dentry { ……` `  struct inode *d_inode ……}`	Ogni istanza rappresenta un nodo (file o catalogo) nell'albero dei direttori del VFS.

`struct inode {` `  ……  }`	Ogni istanza rappresenta uno e un solo file fisicamente esistente nel volume.

La figura sottostante costituisce una rappresentazione dello stato del VFS raggiunto dopo l'esecuzione in sequenza di un certo numero di chiamate di sistema sotto riportate.



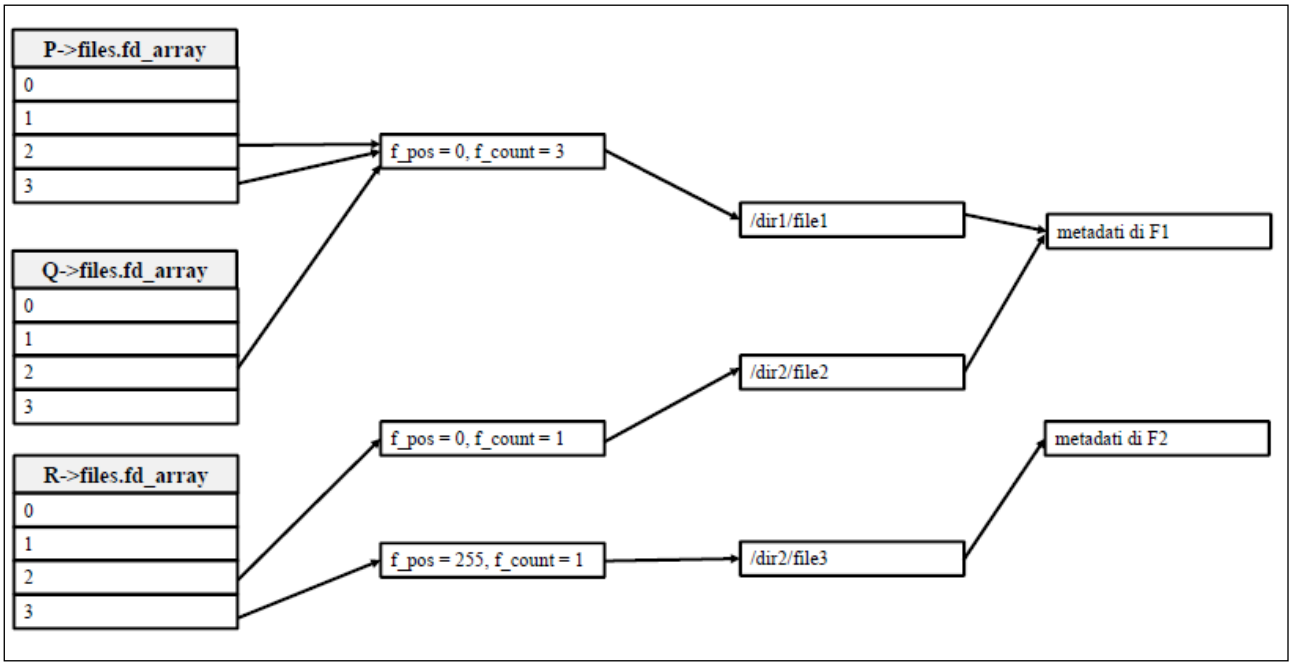
Chiamate di sistema eseguite nell'ordine indicato

- P: close (2)
- P: fd = open (/dir1/file1, …)
- P: pid = fork ( ) // il processo Q è stato creato da P
- P: fd1 = dup (fd)
- Il processo R è stato creato da altro processo non di interesse nell'esercizio
- R: link (/dir1/file1, /dir2/file2)
- R: close (2)
- R: fd = open (/dir2/file2)

Si supponga ora di partire dallo stato del VFS mostrato nella figura iniziale. Per ciascuno degli stati successivi, si risponda alle **domande** riportando la chiamata o la sottosequenza di chiamate che può avere generato la creazione di istanze di `struct` del VFS presentate nelle figure.

Le sole tipologie di chiamate da considerare sono: `open(), close(), read(), dup(), link()`

Domanda 1

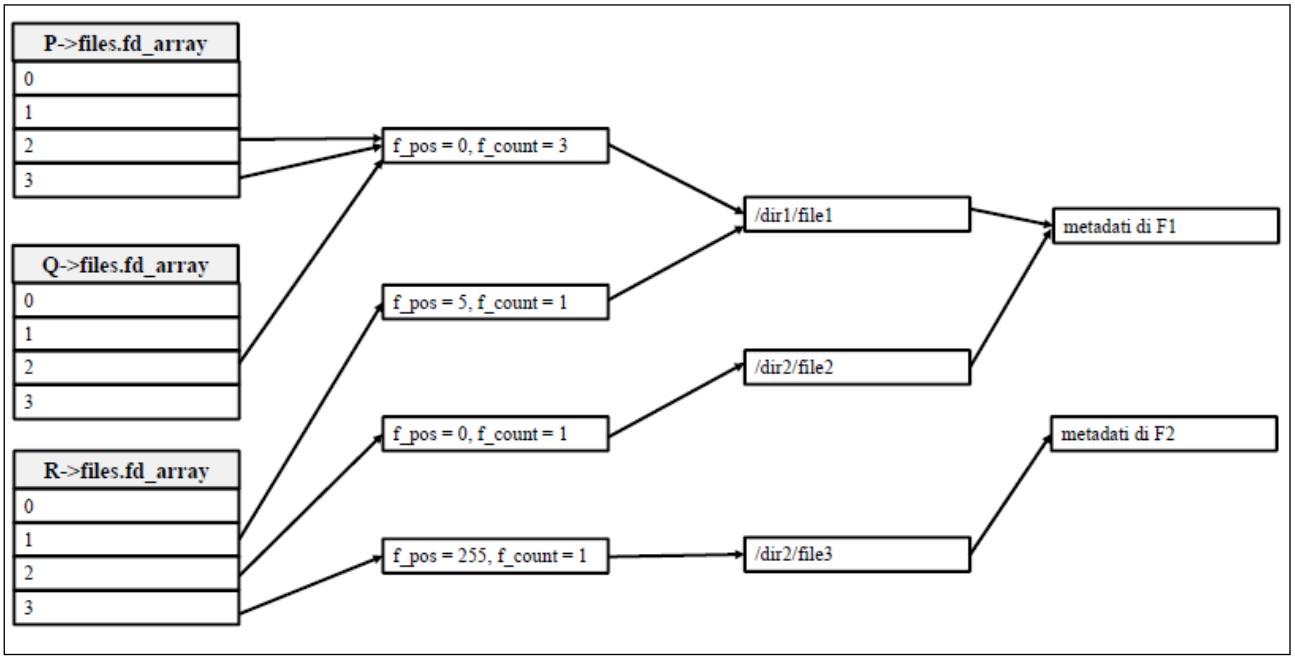


Chiamata/e di sistema

R: fd1 = open("/dir2/file3")

R: read(fd1, 255)

Domanda 2



Chiamata/e di sistema

R: close(1)

R: fd2 = open("/dir1/file1")

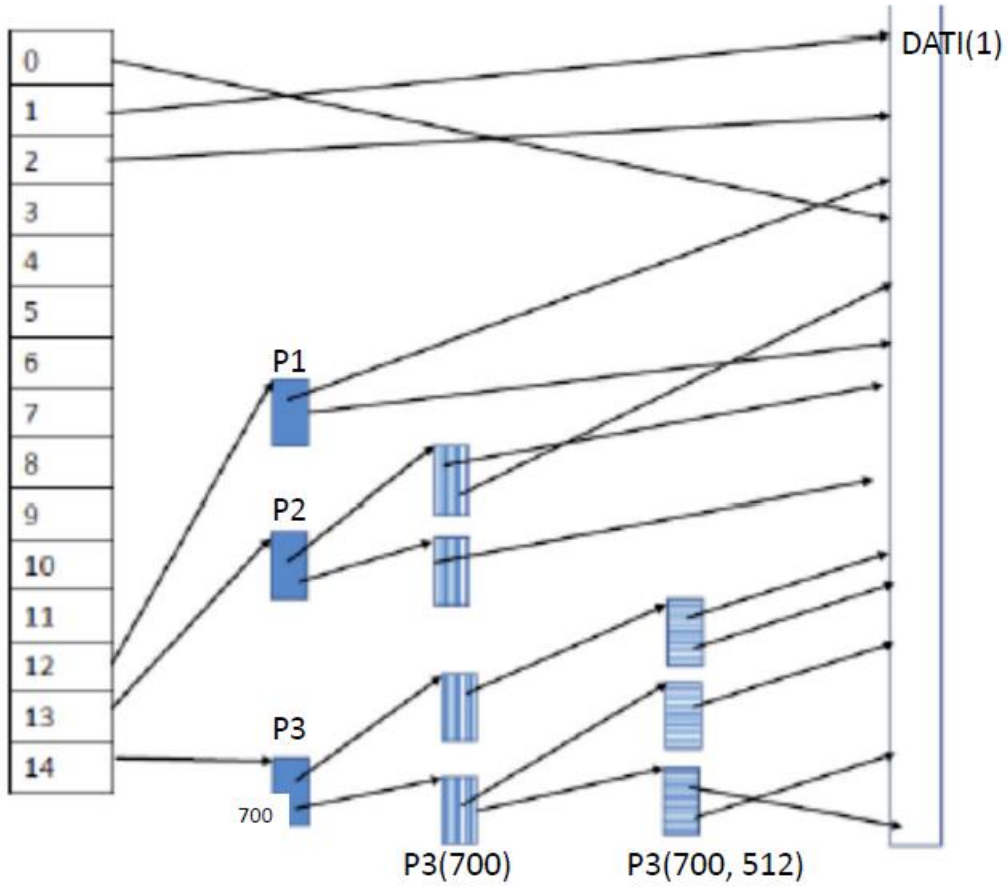
R: read(fd2, 5)

Esercizio 4 – fs2 - esame del 2 luglio 2018

Si consideri il FS ext2 con dimensione di blocco = dim. pagina = 4Kbyte.
Ogni puntatore occupa 4 byte.

Con riferimento alla figura seguente si consideri la seguente notazione:

- DATI(N) indica la pagina dati in posizione N; DATI(0) è la prima pagina dati del file
- P1, P2 e P3 sono i 3 blocchi contenenti puntatori di indirezione semplice
- Pi(j) indica un blocco di puntatori al secondo livello di indirezione raggiunto dal puntatore numero j del blocco i di indirezione semplice (esempio in figura: P3(700) è il blocco puntato dal puntatore in posizione 700 (il 701-esimo, perché i puntatori sono numerati da 0) del blocco P3
- pi(j,k) estende la stessa notazione ai blocchi di terzo livello – tripla indirezione – come P3(700,512) in figura, puntato dal puntatore in posizione 512 del blocco P3(700).



P1 => 12 - ... - 12 + 1024 - 1 => 12 - ... - 1035

P2(0) => 1036 - ... - 1036 + 1024 - 1 = 1035 - ... - 2059

P2(1) => 2060 - ... - 2060 + 1024 - 1 => 2060 - ... - 3083

Si supponga che un programma esegua in sequenza le seguenti operazioni su un file F:

1. open (la open non legge il file, ma solo l'i-node),
2. lseek(FP) – si posiziona all'inizio della **pagina** di numero FP, cioè DATI(FP)
3. read(NUM) – NUM è il numero di **pagine** lette

Indicare il numero totale di blocchi dati e di blocchi puntatore trasferiti dal disco in memoria nei casi seguenti. Il primo caso è già compilato come esempio.

FP=11, NUM=2	**FP=1035, NUM=3**	**FP=2059, NUM=2**	**FP=2058, NUM=3**
DATI(11)			
P1			
DATI(12)			
Numero Totale di trasferimenti da disco nei diversi casi			
3	6	5	6

P1

DATI(1035)

P2

P2(0)

DATI(1036)

DATI(1037)

P2

P2(0)

DATI(2059)

P2(1)

DATI(2060)

P2

P2(0)

DATI(2058)

DATI(2059)

P2(1)

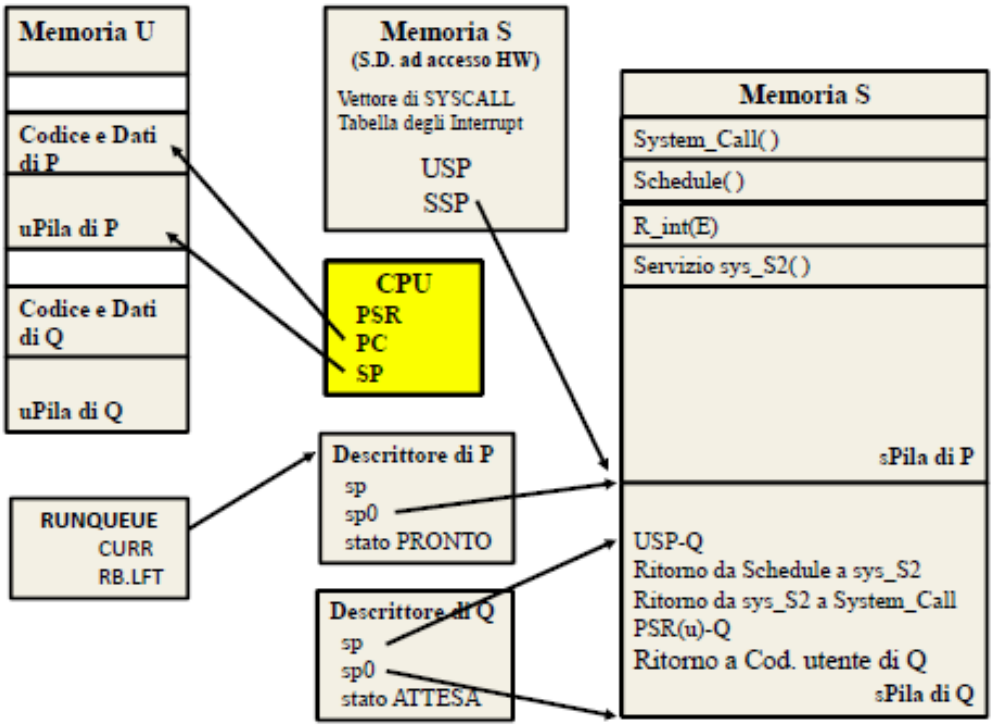
DATI(2060)

Esercizio 4 – nucleo - esame del 22 gennaio 2019

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

- il processo P è in esecuzione in modo U
- il processo Q è in stato di ATTESA dell'evento **E**. La coda RB è da considerarsi vuota (si ricorda che IDLE non appartiene alla 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, dove gli indirizzi rappresentati simbolicamente (X, W, .. A, B ..) sono indirizzi di parola.

Processo P		
PC	X	// è all'interno del codice utente
SP	W	// referenzia la uPila
SSP	Z	// base di sPila
USP	n.s.	// valore non significativo
Descrittore di P.stato	PRONTO	
Processo Q		
USP-Q	A	
Descrittore di Q.sp	B	
Descrittore di Q.sp0	C	
Descrittore di Q.stato	ATTESA	
RUNQUEUE		
CURR	P	
RB.LFT	NULL	

Si consideri il seguente **evento**:

si verifica l'interruzione associata all'evento E. La routine di risposta all'interrupt – *R_int(E)* – risveglia il processo Q che risulta avere un diritto di esecuzione maggiore di quello di P. All'interno di *R_int(E)* viene quindi invocato *schedule( )* per il *context switch*. Si supponga che l'invocazione di *schedule* in avvenga all'indirizzo Y+9, dove Y è l'indirizzo iniziale della routine di risposta all'interrupt.

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	Z- 4
s_Pila di P a (Z - 4)	USP (W)
s_Pila di P a (Z - 3)	Y+ 10
s_Pila di P a (Z - 2)	PSR (U)
s_Pila di P a (Z - 1)	X+ 1
SSP	Z
USP	W
Descrittore di P.sp	Z- 4
Descrittore di P.sp0	Z
Descrittore di P.stato	PRONTO

// 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	B + 1
SSP	C
USP	A
Descrittore di Q.sp	
Descrittore di Q.sp0	C
Descrittore di Q.stato	PRONTO
RUNQUEUE	
CURR	Q
RB.LFT	P

// subito prima del ritorno da *schedule*

// non di interesse

Esercizio 4 – tabella delle pagine - esame del 9 settembre 2019

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 dell'area D del processo in pagine, senza dover modificare la dimensione della TP
6. la dimensione virtuale massima dell'area M0 del processo in pagine, senza dover modificare la dimensione della TP

VMA del processo **P**							
AREA	NPV iniziale	dimensione	R/W	P/S	M/A	nome file	offset
C	*0000 0040 0*	*16*	*R*	*P*	*M*	*X*	*0*
K	*0000 0060 0*	*2*	*R*	*P*	*M*	*X*	*16*
S	*0000 0060 2*	*120*	*W*	*P*	*M*	*X*	*18*
D	*0000 0067 A*	*2*	*W*	*P*	*A*	*-1*	*0*
M0	*0000 3000 0*	*1*	*W*	*S*	*M*	*G*	*2*
P	*7FFF FFFF C*	*3*	*W*	*P*	*A*	*-1*	*0*

1. Decomposizione degli indirizzi virtuali

		PGD :	PUD :	PMD :	PT
C	0000 0040 0	0	0	2	0
K	0000 0060 0	0	0	3	0
S	0000 0060 2	0	0	3	2
D	0000 0067 A	0	0	3	122
M0	0000 3000 0	0	0	384	0
P	7FFF FFFF C	255	511	511	508

2. Numero pagine necessarie

pag PGD: 1 # pag PUD: 2

pag PMD: 2 # pag PT: 4

pag totali: 9

3. Numero pagine virtuali occupate dal processo: 144

4. Rapporto di occupazione: 6.25%

5. Dimensione massima in pagine virtuali dell'area D: 390

6. Dimensione massima in pagine virtuali dell'area M0: 512

0x 0000 0067 A => 0000 0000 0000 0000 0000 0000 0110 0111 1010 => 0 0 3 122

0x 0000 3000 0 => 0000 0000 0000 0000 0011 0000 0000 0000 0000 0000 => 0 0 384 0

esercizio n. 4 – tabella delle pagine - esame del 2 Febbraio 2021

Date le VMA di un processo 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	2	R	P	M	X	0
K	0000 0060 0	1	R	P	M	X	3
D	0000 0060 1	16	W	P	A	-1	0
M0	0000 AB00 0	4	W	S	M	F	5
T0	7FFF F77F D	3	W	P	A	-1	0
P	7FFF FFFF C	3	W	P	A	-1	0

Decomposizione degli indirizzi virtuali

		PGD :	**PUD :**	**PMD :**	**PT**
C	0000 0040 0	0	0	2	0
K	0000 0060 0	0	0	3	0
D	0000 0060 1	0	0	3	1
M0	0000 AB00 0	0	2	336	0
T0	7FFF F77F D	255	511	238	509
P	7FFF FFFF C	255	511	511	508

Numero di pagine necessarie

0x 0000 AB00 0 => 0000 0000 0000 0000 1010 1011 0000 0000 0000 =>
=> 0 2 336 0

# pag PGD	1
# pag PUD	2
# pag PMD	3
# pag PT	5
# pag totali	11

0x 7FFF F77F D => 0111 1111 1111 1111 1111 0111 0111 1111 1101 =>

Numero di pagine virtuali occupate dal processo	29

Rapporto di occupazione	37.93%

Dimensione massima del processo in pagine virtuali	5 * 512 = 2560

Rapporto di occupazione con dimensione massima	0.43%