Esercizio n. 1 – modello thread e parallelismo

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

```
#define NSEM 7
sem_t sem_start[NSEM];
sem_t sem_stop[NSEM];
int ret_status[NSEM];
```

```
void * P (void * arg)
{
   int base = (int) arg;
   int alfa;
   int status;
   int index;
   sem_wait (&sem_start[base]);
   for(alfa = 0; alfa <= 1; alfa++)
   {
      int off = (base == 0) ? 3 : 1;
      index = base + 1 + off * alfa;
      sem_post(&sem_start[index]);
      if (alfa == 1)
      {
         sem_wait(&sem_stop[index]);
         status = ret_status[index];
      } /* statement A */
   }
   index = base + 1;
   sem_wait (&sem_stop[index]);
   status = ret_status[index];                    /* statement B */
   ret_status[base] = alfa;
   sem_post(&sem_stop[base]);
   return NULL;
} /* end P */
```

```
void * N (void * arg)
{
   int tid = (int) arg;
   sem_wait (&sem_start[tid]);
   ret_status[tid] = (tid % 2) ? 4 : 3;
   sem_post(&sem_stop[tid]);                     /* statement C */
   return NULL;
} /* end N */
```

```
int main ( )
{
   pthread_t th_P0, th_F1, th_F2;
   pthread_t th_N1, th_N2, th_N3, th_N4;
   int i;
```

```
   for(i = 0; i < NSEM; i++)
   {
      sem_init(&sem_start[i], 0, 0);
      sem_init(&sem_stop[i], 0, 0);
   }

   pthread_create (&th_P0, NULL, P, (void *) 0);      /* P0 */
   pthread_create (&th_F1, NULL, P, (void *) 1);      /* F1 */
   pthread_create (&th_F2, NULL, P, (void *) 4);      /* F2 */

   pthread_create (&th_N1, NULL, N, (void *) 2);      /* N1 */
   pthread_create (&th_N2, NULL, N, (void *) 3);      /* N2 */
   pthread_create (&th_N3, NULL, N, (void *) 5);      /* N3 */
   pthread_create (&th_N4, NULL, N, (void *) 6);      /* N4 */

   sem_post(&sem_start[0]);
   sem_wait(&sem_stop[0]);

   pthread_join (th_P0, NULL);
   pthread_join (th_F1, NULL);
   pthread_join (th_F2, NULL);

   pthread_join (th_N1, NULL);
   pthread_join (th_N2, NULL);
   pthread_join (th_N3, NULL);
   pthread_join (th_N4, NULL);

   return 0;
} /* end main */
```

1. Disegnare un grafo di sincronizzazione fra I thread (P0, F1, F2, N1, N2, N3, N4). Il grafo di sincronizzazione è composto da:
    - *cerchi*, rappresentanti i thread;
    - *frecce uni-direzionali*, da un thread T1 ad un thread T2, quando T1 esegue un post su almeno uno dei semafori su cui T2 è in attesa;
    - *frecce bi-direzionali*, tra un thread T1 e un thread T2, quando T1 esegue un post su almeno uno dei semafori su cui T2 è in attesa e, viceversa, T2 esegue un post su almeno uno dei semafori su cui T1 è in attesa.
2. Riempire la tabella relativa ai thread creati, indicando l'argomento passato a ciascun thread e il valore presente nell'array *ret_status* in corrispondenza di tali indici.

Thread	**P0**	**F1**	**F2**	**N1**	**N2**	**N3**	**N4**
arg							
ret_status							

3. Riempire la tabella dei valori delle variabili segnando il valore di ciascuna variabile in corrispondenza degli statement indicati. Il valore di una variabile o semaforo può essere indicato come:
    - intero, carattere, stringa, quando la variabile ha un valore definito; se la variabile può assumere più valori, riportare tutti i possibili valori.
    - *NE*: se la variabile non esiste
    - *UND*: se la variabile esiste ma non è stata ancora inizializzata.

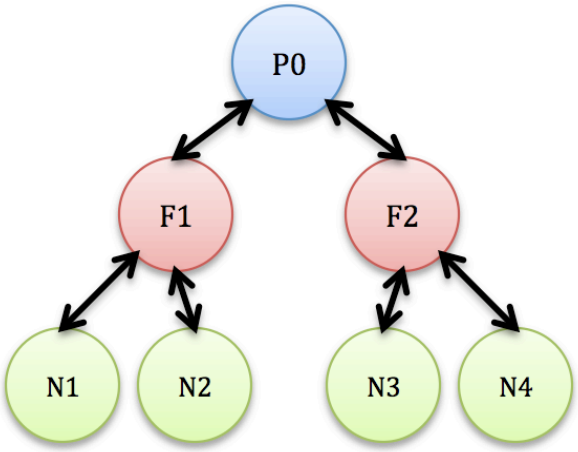
    Prestare attenzione all'etichetta **'Condizione'**. In particolare, con **'Subito dopo statement X'** si intende il valore (o i valori) assumibili dalle variabili tra lo statement X e lo statement immediatamente successivo *all'interno del thread indicato nella condizione stessa*.

Thread	**Condizione**	**Variabile**		
		alfa	**status**	**sem_stop[6]**
P0	Subito dopo statement A in F1 (prima iterazione)			
P0	Subito dopo statement A in F2 (prima iterazione)			
P0	Subito dopo statement B in P0			
F1	Subito dopo statement C in N2			
F1	Subito dopo statement B in F1			
F2	Subito dopo statement C in N3			
F2	Subito dopo statement B in F2			

SOLUZIONE

1)

Il grafo connette con delle freccie bi-direzionali *P0* a (*F1*, *F2*), *F2* a (*N1*, *N2*) ed *F3* a (*N3*, *N4*). La struttura di sincronizzazione in pratica è tale per cui *P0*, *F1* ed *F2* fanno partire due thread ciascuno tramite un post sul semaforo *sem_start*, aspettando la loro terminazione in ordine inverso (attesa su *sem_stop*). Quindi, *P0* fa partire *F1* ed *F2* e poi attende che termini prima *F2* seguito da *F1*. *F1* fa partire *N1* ed *N2* aspettando che termini prima *N2* e poi *N1*. Infine *F2* fa partire *N3* ed *N4* aspettando che termini prima *N4* poi *N3*.



2)

Thread	P0	F1	F2	N1	N2	N3	N4
arg	0	1	4	2	3	5	6
ret_status	2	2	2	3	4	4	3

3)

Thread	Condizione	Variabile		
		alfa	status	sem_stop[6]
P0	Subito dopo statement A in F1 (prima iterazione)	0 / 1 / 2	UND / 2	0 / 1
P0	Subito dopo statement A in F2 (prima iterazione)	1	UND	0
P0	Subito dopo statement B in P0	2	2	0
F1	Subito dopo statement C in N2	1 / 2 / NE	UND / 3 / 4 / NE	0 / 1
F1	Subito dopo statement B in F1	2	3	0 / 1
F2	Subito dopo statement C in N3	0 / 1 / 2 / NE	UND / 3 / 4 / NE	0 / 1
F2	Subito dopo statement B in F2	2	4	0

Esercizio n. 2 – modello thread e parallelismo (Compitino 2014)

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

```
pthread_mutex_t you, me;
sem_t plus, minus, zero;
int global = 0;
```

```
void * one (void * arg) {
   pthread_mutex_lock (&you);
   sem_post (&plus);                          /* statement A */
   pthread_mutex_unlock (&you);
   pthread_mutex_lock (&me);
   sem_wait (&minus);
   sem_post (&zero);
   global = 1;
   pthread_mutex_unlock (&me);
   return NULL;
} /* end one */
```

```
void * two (void * arg) {
   sem_wait (&plus);                          /* statement B */
   pthread_mutex_lock (&you);
   global = 2;
   sem_wait (&zero);
   pthread_mutex_unlock (&you);
   return NULL;
} /* end two */
```

```
void * three (void * arg) {
   sem_wait (&zero);
   pthread_mutex_lock (&me);                  /* statement C */
   sem_post (&minus);
   global = 3;
   pthread_mutex_unlock (&me);
   return NULL;
} /* end three */
```

```
void main ( ) {
   pthread_t th_1, th_2, th_3;
   sem_init (&plus, 0, 0);
   sem_init (&minus, 0, 0);
   sem_init (&zero, 0, 1);
   pthread_create (&th_2, NULL, two, NULL);
   pthread_create (&th_1, NULL, one, NULL);
   pthread_create (&th_3, NULL, three, NULL);
   pthread_join (th_1, NULL);
   pthread_join (th_2, NULL);              /* statement D */
   pthread_join (th_3, 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	**th_2**	**th_3**
subito dopo stat. **A**			
subito dopo stat. **B**			
subito dopo stat. **C**			
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:

4. intero, carattere, stringa, quando la variabile ha un valore definito; se la variabile può assumere più valori, riportare tutti I possibili valori.
5. se la variabile può avere due o più valori, li si riporti tutti quanti
6. 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	
	zero	*global*
subito dopo stat. **A**		
subito dopo stat. **B**		
subito dopo stat. **C**		
subito dopo stat. **D**		

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, con il valore (o i valori) della variabile `global`:

caso	th_1	th_2	th_3	*global*
1				
2				

SOLUZIONE

Condizione	**thread**		
	th_1	**th_2**	**th_3**
subito dopo stat. **A**	*ESISTE*	*ESISTE*	*PUÒ ESISTERE*
subito dopo stat. **B**	*PUÒ ESISTERE*	*ESISTE*	*PUÒ ESISTERE*
subito dopo stat. **C**	*ESISTE*	*ESISTE*	*ESISTE*
subito dopo stat. **D**	*NON ESISTE*	*NON ESISTE*	*PUÒ ESISTERE*

condizione	**variabili globali**	
	zero	*global*
subito dopo stat. **A**	*0 / 1*	*0 / 3*
subito dopo stat. **B**	*0 / 1*	*0 / 1 / 3*
subito dopo stat. **C**	*0*	*0 / 2*
subito dopo stat. **D**	*0*	*1 / 2*

caso	**th_1**	**th_2**	**th_3**	*global*
1	`wait minus`	`wait zero` [+]	`lock me`	*0* [*] */ 2*
2	`wait minus`	–	`wait zero`	*2*

[*] *Asintoticamente il valore 0 viene sovrascritto dal valore 2.*

[+] *Asintoticamente il thread 2 si blocca su* `wait zero`*; questa però è solo una conseguenza dello stallo, il quale esiste indipendentemente dal blocco del thread 2.*

Esercizio n. 3 – thread e parallelismo (appello 26/2/2015)

Si consideri il programma C seguente (gli "#include" sono omessi):

```
pthread_mutex_t GATE = PTHREAD_MUTEX_INITIALIZER;
sem_t GO;
int global = 0;

void * FIRST (void * arg) {
   sem_wait (&GO);                              /* istruzione A */
   global = 1;
   pthread_mutex_lock (&GATE);
   sem_post (&GO);
   pthread_mutex_unlock (&GATE);
   return NULL;
} /* end FIRST */

void * LAST (void * arg) {
   if (global == 0) {
      global = 2;
      pthread_mutex_lock (&GATE);               /* istruzione B */
      sem_wait (&GO);
      pthread_mutex_unlock (&GATE);
      sem_post (&GO);
   } else {
      global = 3;
      sem_wait (&GO);                           /* istruzione C */
   } /* if */
   return NULL;
} /* end LAST */

void main ( ) {
   pthread_t TH_1, TH_2, TH_3;
   sem_init (&GO, 0, 1);
   pthread_create (&TH_1, NULL, FIRST, NULL);
   pthread_create (&TH_2, NULL, LAST, NULL);
   pthread_join (TH_1, NULL);
   pthread_create (&TH_3, NULL, FIRST, NULL);
   pthread_join (TH_3, NULL);                   /* istruzione D */
   pthread_join (TH_2, 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 la variabile o il parametro assume tra lo statement X e lo statement immediatamente successivo del thread indicato.

condizione	**stato del thread**		
	TH_1	**TH_2**	**TH_3**
subito dopo istr. **A** in **TH_1**			
subito dopo istr. **B**			
subito dopo istr. **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:

7. intero, carattere, stringa, quando la variabile ha un valore definito; se la variabile può assumere più valori, riportare tutti I possibili valori.
8. se la variabile può avere due o più valori, li si riporti tutti quanti
9. 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	**variabile globale**	
	GO	*global*
subito dopo istr. **A** in **TH_1**		
subito dopo istr. **B**		
subito dopo istr. **C**		
subito dopo istr. **D**		

Il sistema va in stallo (deadlock), in **uno o più casi**. Qui **si indichino** le primitive dove si bloccano i thread (uno solo o più di uno), precisando il valore (o i valori) della variabile `global` (il numero di righe nella tabella non è significativo):

caso	**TH_1**	**TH_2**	**TH_3**	*global*
1				
2				
3				
4				

SOLUZIONE

condizione	**stato del thread**		
	TH_1	**TH_2**	**TH_3**
subito dopo istr. **A** in **TH_1**	*ESISTE*	*PUÒ ESISTERE*	*NON ESISTE*
subito dopo istr. **B**	*PUÒ ESISTERE*	*ESISTE*	*PUÒ ESISTERE*
subito dopo istr. **D**	*NON ESISTE*	*PUÒ ESISTERE*	*NON ESISTE*

condizione	**variabile globale**	
	GO	*global*
subito dopo istr. **A** in **TH_1**	*0*	*0 / 2*
subito dopo istr. **B**	*0 / 1*	*1 / 2*
subito dopo istr. **C**	*0*	*1 / 3*
subito dopo istr. **D**	*0 / 1*	*1 / 2 / 3*

caso	**TH_1**	**TH_2**	**TH_3**	*global*
1	*Lock GATE*	*prima wait GO*	*non creato*	*1 / 2*
2	*terminato*	*prima wait GO*	*Lock GATE*	*1 / 2*
3	*terminato*	*terminato*	*Wait GO*	*3*

Esercizio n. 4 – thread e parallelismo (appello settembre 2014)

Si consideri il programma C seguente (gli "#include" sono omessi):

```
/* variabili globali*/
pthread_mutex_t gate = PTHREAD_MUTEX_INITIALIZER;
sem_t go;
int val = 0;
```

```
void * header (void * arg) {            /* funzione di thread*/

     printf ("Iniziato.\n");
     sem_post (&go);
     pthread_mutex_lock (&gate);
     sem_wait (&go);                            /* statement A */
     pthread_mutex_unlock (&gate);
     printf ("Finito.\n");
     return NULL;

} /* header */
```

```
void * trailer (void * id) {            /* funzione di thread*/

     int on = 0;                          /* variabile locale*/

     pthread_mutex_lock (&gate);            /* statement B */
     sem_wait (&go);
     pthread_mutex_unlock (&gate);
     on = 1;                                /* statement C */
     sem_post (&go);
     val = (int) id;
     return NULL;

} /* trailer */
```

```
int main (int argc, char * argv [ ]) {   /* thread principale*/

     pthread_t hr, tr1, tr2;
     sem_init (&go, 0, 0);
     pthread_create (&tr1, NULL, &trailer, (void *) 1);
     pthread_create (&hr, NULL, &header, NULL);
     pthread_create (&tr2, NULL, &trailer, (void *) 2);
     pthread_join (hr);
     pthread_join (tr1);                    /* statement D */
     pthread_join (tr2);

} /* 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 la variabile o il parametro assume tra lo statement X e lo statement immediatamente successivo del thread indicato.

condizione	**stato del thread**	
	TR1	**TR2**
subito dopo stat. **A**		
subito dopo stat. **C** in **TR2**		
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:

10. intero, carattere, stringa, quando la variabile ha un valore definito; se la variabile può assumere più valori, riportare tutti I possibili valori.
11. se la variabile può avere due o più valori, li si riporti tutti quanti
12. 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**	
	go	*val*
subito dopo stat. **A**		
subito dopo stat. **B** in **TR1**		
subito dopo stat. **C** in **TR2**		

Il sistema può andare in stallo (deadlock) in una o più situazioni. Qui **si indichino** gli statement dove si bloccano i thread incorsi in stallo (non necessariamente tutti) (numero di righe non significativo):

caso	HR	TR1	TR2
1			
2			
3			
4			
5			

SOLUZIONE

condizione	stato del thread	
	TR1	**TR2**
subito dopo stat. **A**	*PUÒ ESISTERE*	*PUÒ ESISTERE*
subito dopo stat. **C** in **TR2**	*PUÒ ESISTERE*	*ESISTE*
subito dopo stat. **D**	*NON ESISTE*	*PUÒ ESISTERE*

condizione	variabili globali	
	go	*val*
subito dopo stat. **A**	*0*	*0 / 1 / 2*
subito dopo stat. **B** in **TR1**	*0 / 1*	*0 / 2*
subito dopo stat. **C** in **TR2**	*0*	*0 / 1*

caso	HR	TR1	TR2
1	*terminato*	***sem_wait*** *(&go)*	***pthread_mutex_lock*** *(&gate)*
2	*terminato*	***pthread_mutex_lock*** *(&gate)*	***sem_wait*** *(&go)*
3	*terminato*	*terminato*	***sem_wait*** *(&go)*
4	*terminato*	***sem_wait*** *(&go)*	*terminato*

Esercizio n. 5 – modello thread e parallelismo (compitino 2013)

Si consideri il programma C seguente (gli "#include" sono omessi):

```
pthread_mutex_t time, space;
sem_t open, close;
int global = 0;
```

```
void * one (void * arg) {
   pthread_mutex_lock (&time);
   sem_post (&open);                          /* statement A */
   pthread_mutex_lock (&space);
   global = 1;
   pthread_mutex_unlock (&time);
   pthread_mutex_unlock (&space);
   return (void *) 1;
} /* end one */
```

```
void * two (void * arg) {
   pthread_mutex_lock (&time);
   global = 2;
   sem_wait (&open);                          /* statement B */
   pthread_mutex_unlock (&time);
   sem_wait (&close);
   return NULL;
} /* end two */
```

```
void * three (void * arg) {
   pthread_mutex_lock (&space);
   sem_wait (&close);
   global = 3;
   pthread_mutex_unlock (&space);
   sem_post (&close);                         /* statement C */
   return NULL;
} /* end two */
```

```
void main ( ) {
   pthread_t th_1, th_2, th_3;
   sem_init (&open, 0, 0);
   sem_init (&close, 0, 1);
   pthread_create (&th_2, NULL, two, NULL);
   pthread_create (&th_1, NULL, one, NULL);
   pthread_create (&th_3, NULL, three, NULL);
   pthread_join (th_3, NULL);
   pthread_join (th_1, &global);          /* statement D */
   pthread_join (th_2, 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	**th_2**	**th_3**
subito dopo stat. **A**			
subito dopo stat. **B**			
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:

13. intero, carattere, stringa, quando la variabile ha un valore definito; se la variabile può assumere più valori, riportare tutti I possibili valori.
14. se la variabile può avere due o più valori, li si riporti tutti quanti
15. 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**	
	open	*global*
subito dopo stat. **A**		
subito dopo stat. **B**		
subito dopo stat. **C**		
subito dopo stat. **D**		

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 processo che non potrà mai proseguire). Si indichino gli statement dove avvengono i blocchi, con il valore (o i valori) della variabile `global`:

caso	**th_1**	**th_2**	**th_3**	*global*
1				
2				

SOLUZIONE

condizione	**thread**		
	th_1	**th_2**	**th_3**
subito dopo stat. **A**	*ESISTE*	*ESISTE*	*PUÒ ESISTERE*
subito dopo stat. **B**	*PUÒ ESISTERE*	*ESISTE*	*PUÒ ESISTERE*
subito dopo stat. **D**	*NON ESISTE*	*PUÒ ESISTERE*	*NON ESISTE*

condizione	**variabili globali**	
	open	*global*
subito dopo stat. **A**	*1*	*0 / 3*
subito dopo stat. **B**	*0*	*1 / 2 / 3*
subito dopo stat. **C**	*0 / 1*	*1 / 2 / 3*
subito dopo stat. **D**	*0 / 1*	*1 / 2*

caso	**th_1**	**th_2**	**th_3**	*global*
1	`lock time`	`wait open`	-	*2 / 3*
2	-	-	`wait close`	*2*