Sistemi Operativi e Laboratorio, Prova del 4/4/2019 – versione A

Nome e cognome: _____ Matricola:_____ corso:____ aula: __fila:__ posto:___

Esercizio 1 (5 punti)

Un aeroporto dispone di un'unica pista di decollo e di un'area di preparazione al decollo che può ospitare al massimo `n` aeromobili. Quando un aeromobile fermo al gate è pronto per la partenza richiede preliminarmente l'autorizzazione all'accesso dell'area di preparazione al decollo (se questa è piena l'aeromobile resta in attesa al gate) e, successivamente, quando la pista è disponibile, ottiene l'accesso alla pista da parte della torre di controllo. Al termine del decollo l'aeromobile comunica alla torre di controllo il completamento del decollo (e l'implicito rilascio della pista). L'intera procedura avviene quindi secondo la seguente sequenza:

```
richiestaPreparazioneDecollo(idAeromobile)
<decollo>
decolloAvvenuto()
```

Ipotizzando una gestione interamente automatica dell'aeroporto e degli aeromobili, ogni aeromobile è associato ad un thread che ne gestisce l'intera procedura di decollo (denotiamo tali thread $A_1,\ldots,A_m$).

Si chiede di completare le procedure `richiestaPreparazioneDecollo(idAeromobile)` e `decolloAvvenuto()` assumendo che i thread $A_1,\ldots,A_m$ utilizzino le seguenti variabili condivise per coordinare le varie fasi della partenza:

- `mutex` – per gestire la mutua esclusione tra aeromobili;
- `attesaAlGate` – variabile di condizione sulla quale si sospendono gli aeromobili che non ottengono l'accesso all'area di preparazione al decollo;
- `attesaInPreparazione` – variabile di condizione sulla quale si sospendono gli aeromobili in preparazione al decollo per attendere la disponibilità della pista;

Lo stato del sistema è inoltre rappresentato dalle seguenti variabili:

- `readyGate` (inizializzato a `0`): è il numero di aerei fermi al gate in attesa di accedere all'area di preparazione al decollo;
- `attesaDecollo` (inizializzata a `0`): è il numero di aerei nell'area di preparazione al decollo;
- `pistaLibera` (inizializzata a `True`): è `True` se la pista è libera, è `False` altrimenti.

Si chiede di completare la seguente soluzione assumendo che la semantica delle variabili di condizione sia di tipo MESA.

```
richiestaPreparazioneDecollo(Ai) {

        _____________________________
        readyGate++;
        while (attesaDecollo == n)

                _____________________________
        readyGate--;
        attesaDecollo++;
        while (! pistaLibera)

                _____________________________
        attesaDecollo--;
        pistaLibera=FALSE;

        if (readyGate>0) _____________________________

        _____________________________
}
decolloAvvenuto() {

        _____________________________
        pistaLibera=TRUE;

        if(attesaDecollo>0) _____________________________

        _____________________________
}
```

Soluzione

```
richiestaPreparazioneDecollo(Ai) {
        mutex.Acquire();
        readyGate++;
        while (attesaDecollo == n)
```

```
            attesaAlGate.wait(mutex);
        readyGate--;
        attesaDecollo++;
        while (! pistaLibera)
            attesaInPreparazione.wait(mutex);
        attesaDecollo--;
        pistaLibera=FALSE;
        if (readyGate>0) attesaAlGate.signal();
        mutex.Release();
}
decolloAvvenuto() {
        mutex.Acquire();
        pistaLibera=TRUE;
        if(attesaDecollo>0) attesaInPreparazione.signal();
        mutex.Release();
}
```

Esercizio 2 (5 punti)

Dire quali operazioni svolge all'inizio della sua computazione l'handler di gestione dell'interruzione da timer (max. 5 righe).

_____

_____

_____

_____

_____

Si consideri un processore che dispone dei seguenti registri:
- i registri speciali PC (program counter) e PS (program status) e lo stack pointer SP
- un banco di registri riservato allo stato utente, che comprende i registri generali R1, R2,
- un ulteriore banco di registri riservato allo stato supervisore, che comprende i registri generali R'1, R'2 e lo stack pointer dello stack del nucleo SP'.

Il sistema operativo realizza i thread a livello kernel, e i thread costituiscono l'unica unità di schedulazione del sistema. Come meccanismi di mutua esclusione e di sincronizzazione, il sistema offre ai thread i semafori.
Al tempo t sono presenti solo tre thread: il thread T1, in stato di esecuzione, il thread T2 in stato di pronto e il thread T3 in stato di attesa sul semaforo SEM. Nessun altro thread è presente nel sistema. Al tempo *t* il thread T1 esegue una istruzione SVC per invocare la chiamata di sistema P(SEM). Al momento dell'esecuzione dell'istruzione SVC i registri del processore, i descrittori di T1, T2 e T3 e lo stack del nucleo hanno i contenuti mostrati in tabella. Lo stack pointer del nucleo ha il valore 1016.
Il vettore di interruzione associato all'interruzione generata da SVC è 0425 e la parola di stato del nucleo è 275E.
Si chiede:

a) il contenuto dei descrittori, dei registri generali e speciali, dello stack del nucleo e lo stato del processore durante la fase di estrazione della prima istruzione della funzione di servizio;
b) il contenuto dei descrittori, dei registri generali e speciali, dello stack del nucleo e lo stato del processore durante la fase di estrazione dell'istruzione IRET con la quale termina la chiamata di sistema;
c) il contenuto dei descrittori, dei registri generali e speciali, dello stack del nucleo e lo stato del processore durante la fase di estrazione dell'istruzione eseguita subito dopo la IRET.

Descrittore T1	
stato	esec.
PC	2E31
PS	16F2
SP	1873
R1	1234
R2	16CC

Descrittore T2	
stato	pronto
PC	B12C
PS	B6F2
SP	BFF5
R1	B5CC
R2	B000

Descrittore T3	
stato	attesa
PC	CC11
PS	C6F2
SP	CFF7
R1	C1C1
R2	C2C2

Processore: registri speciali						
PC	2F00		PS	16F2	stato	utente

Stack del nucleo	
……	……
1016	
1014	
1012	
1010	
100E	
100C	

Registri generali	
SP	AFFF
R1	AA2A
R2	A2CE
Registri stato superv.	
SP'	1016
R1'	0000
R2'	0001

Soluzione

a) contenuto dei descrittori, dei registri generali e speciali, dello stack del nucleo e lo stato del processore durante la fase di estrazione della prima istruzione della funzione di servizio:

Sistemi Operativi e Laboratorio, Prova del 4/4/2019 – versione A

Descrittore T1	
stato	
PC	
PS	
SP	
R1	
R2	

Descrittore T2	
stato	
PC	
PS	
SP	
R1	
R2	

Descrittore T3	
stato	
PC	
PS	
SP	
R1	
R2	

Processore: registri speciali					
PC		PS		stato	

Stack del nucleo	
……	……..
1016	
1014	
1012	
1010	
100E	
100C	

Registri generali	
SP	
R1	
R2	
Registri stato superv.	
SP’	
R1’	
R2’	

b) contenuto dei descrittori, dei registri generali e speciali, dello stack del nucleo e lo stato del processore durante la fase di estrazione dell’istruzione IRET con la quale termina la chiamata di sistema;

Descrittore T1	
stato	
PC	
PS	
SP	
R1	
R2	

Descrittore T2	
stato	
PC	
PS	
SP	
R1	
R2	

Descrittore T3	
stato	
PC	
PS	
SP	
R1	
R2	

Processore: registri speciali					
PC		PS		stato	

Stack del nucleo	
……	……..
1016	
1014	
1012	
1010	
100E	
100C	

Registri generali	
SP	
R1	
R2	
Registri stato superv.	
SP’	
R1’	
R2’	

c) Contenuto dei descrittori, dei registri generali e speciali, dello stack del nucleo e lo stato del processore durante la fase di estrazione dell’istruzione eseguita subito dopo la IRET.

Descrittore T1	
stato	
PC	
PS	
SP	
R1	
R2	

Descrittore T2	
stato	
PC	
PS	
SP	
R1	
R2	

Descrittore T3	
stato	
PC	
PS	
SP	
R1	
R2	

Processore: registri speciali					
PC		PS		stato	

Stack del nucleo	
……	……..
1016	
1014	
1012	
1010	
100E	
100C	

Registri generali	
SP	
R1	
R2	
Registri stato superv.	
SP’	
R1’	
R2’	

Soluzione

a) Contenuto dei descrittori, dei registri generali e speciali, dello stack del nucleo e lo stato del processore durante la fase di estrazione della prima istruzione della funzione di servizio:

Descrittore T1	
stato	esec.
PC	2E31
PS	16F2
SP	1873
R1	1234
R2	16CC

Descrittore T2	
stato	pronto
PC	B12C
PS	B6F2
SP	BFF5
R1	B5CC
R2	B000

Descrittore T3	
stato	attesa
PC	CC11
PS	C6F2
SP	CFF7
R1	C1C1
R2	C2C2

Processore: registri speciali					
PC	0425	PS	275E	stato	supervisore

Stack del nucleo	
……	……..
1016	2F00
1014	16F2
1012	
1010	
100E	
100C	

Registri generali	
SP	AFFF
R1	AA2A
R2	A2CE
Registri stato superv.	
SP’	1012
R1’	0000
R2’	0001

b) Contenuto dei descrittori, dei registri generali e speciali, dello stack del nucleo e lo stato del processore durante la fase di estrazione dell’istruzione IRET con la quale termina la chiamata di sistema;

Descrittore T1	
stato	attesa
PC	2F00
PS	16F2
SP	AFFF
R1	AA2A

Descrittore T2	
stato	Esecuz.
PC	B12C
PS	B6F2
SP	BFF5
R1	B5CC

Descrittore T3	
stato	attesa
PC	CC11
PS	C6F2
SP	CFF7
R1	C1C1

Stack del nucleo	
……	……..
1016	B12C
1014	B6F2
1012	
1010	

Registri generali	
SP	BFF5
R1	B5CC
R2	B000
Registri stato superv.	

R2	A2CE		R2	B000		R2	C2C2
Processore: registri speciali							
PC	0425+?		PS	275E		stato	superviso-re

100E	
100C	

SP'	1012
R1'	?
R2'	?

c) Contenuto dei descrittori, dei registri generali e speciali, dello stack del nucleo e lo stato del processore durante la fase di estrazione dell'istruzione eseguita subito dopo la IRET.

Descrittore T1	
stato	attesa
PC	2F00
PS	16F2
SP	AFFF
R1	AA2A
R2	A2CE

Descrittore T2	
stato	Esecuz.
PC	B12C
PS	B6F2
SP	BFF5
R1	B5CC
R2	B000

Descrittore T3	
stato	attesa
PC	CC11
PS	C6F2
SP	CFF7
R1	C1C1
R2	C2C2

Processore: registri speciali					
PC	B12C	PS	B6F2	stato	utente

Stack del nucleo	
……	……
1016	
1014	
1012	
1010	
100E	
100C	

Registri generali	
SP	BFF5
R1	B5CC
R2	B000
Registri stato superv.	
SP'	1016
R1'	?
R2'	?

Esercizio 3 (5 punti)

Un sistema con 5 processi (A, B, C, D, E) e risorse dei tipi R1, R2, R3, R4, rispettivamente di molteplicità [8, 6, 7, 5], utilizza l'algoritmo del banchiere per evitare lo stallo. Il sistema ha raggiunto lo stato (sicuro) mostrato nelle tabelle seguenti.

Assegnazione attuale				
	R1	R2	R3	R4
A			1	1
B	2	3		
C	2	2		
D			2	
E	2		3	2

Esigenza residua				
	R1	R2	R3	R4
A	6	5	1	4
B	5	1	6	3
C	2	1	0	2
D	3	3	3	2
E	2	2	0	1

Molteplicità			
R1	R2	R3	R4
8	6	7	5

Disponibilità			
2	1	1	2

Si considerino ora i seguenti casi (**in alternativa**):

a) Il processo A richiede un'istanza della risorsa R3
b) Il processo B richiede un'istanza della risorsa R3

In ognuno dei due casi, dire se la risorsa viene assegnata dal sistema operativo e quale è lo stato del processo interessato dopo che l'algoritmo del banchiere ha analizzato la richiesta di risorse.

Soluzione

Stato raggiunto dopo l'ipotetica assegnazione di una istanza di R3 al processo A:

Assegnazione attuale				
	R1	R2	R3	R4
A				
B				
C				
D				
E				

Esigenza residua (dopo l'assegnazione attuale)				
	R1	R2	R3	R4
A				
B				
C				
D				
E				

Molteplicità			
R1	R2	R3	R4
8	6	7	5

Disponibilità			

a1) Il processo _____ può/non può* terminare; la disponibilità di {R1, R2, R3, R4} diviene _____
a2) Il processo _____ può/non può* terminare; la disponibilità di {R1, R2, R3, R4} diviene _____
a3) Il processo _____ può/non può* terminare; la disponibilità di {R1, R2, R3, R4} diviene _____
a4) Il processo _____ può/non può* terminare; la disponibilità di {R1, R2, R3, R4} diviene _____
a5) Il processo _____ può/non può* terminare; la disponibilità di {R1, R2, R3, R4} diviene _____

* Cancellare ciò che non si applica

Stato sicuro? _____

Di conseguenza: risorsa assegnata? _____ Stato di A? _____

Stato raggiunto dopo l'ipotetica assegnazione di un'istanza di R3 al processo B:

Sistemi Operativi e Laboratorio, Prova del 4/4/2019 – versione A

Assegnazione attuale				
	R1	R2	R3	R4
A				
B				
C				
D				
E				

Esigenza residua (dopo l'assegnazione attuale)				
	R1	R2	R3	R4
A				
B				
C				
D				
E				

Molteplicità			
R1	R2	R3	R4
8	6	7	5

Disponibilità			

b1) Il processo _____ può/non può* terminare; la disponibilità di {R1, R2, R3, R4} diviene _____
b2) Il processo _____ può/non può* terminare; la disponibilità di {R1, R2, R3, R4} diviene _____
b3) Il processo _____ può/non può* terminare; la disponibilità di {R1, R2, R3, R4} diviene _____
b4) Il processo _____ può/non può* terminare; la disponibilità di {R1, R2, R3, R4} diviene _____
b5) Il processo _____ può/non può* terminare; la disponibilità di {R1, R2, R3, R4} diviene _____
* Cancellare ciò che non si applica
Stato sicuro? _____
Di conseguenza: risorsa assegnata? _____ Stato di B? _____

Soluzione
Stato raggiunto dopo l'ipotetica assegnazione di un'istanza di R3 al processo A:

Assegnazione attuale				
	R1	R2	R3	R4
A			2	1
B	2	3		
C	2	2		
D			2	
E	2		3	2

Esigenza residua				
	R1	R2	R3	R4
A	6	5	0	4
B	5	1	6	3
C	2	1	0	2
D	3	3	3	2
E	2	2	0	1

Molteplicità			
R1	R2	R3	R4
8	6	7	5

Disponibilità			
2	1	0	2

a1) Il processo C può terminare; la disponibilità di {R1, R2, R3, R4} diviene {4,3,0,2}
a2) Il processo E può terminare ; la disponibilità di {R1, R2, R3, R4} diviene {6,3,3,4}
a3) Il processo D può terminare; la disponibilità di {R1, R2, R3, R4} diviene {6,3,5,4}
a4) i processi A e B non possono terminare

Stato sicuro? **NO**
Risorsa assegnata? **NO**
Stato di A? **ATTESA**

Stato raggiunto dopo l'ipotetica assegnazione di un'istanza di R3 al processo B:

Assegnazione attuale				
	R1	R2	R3	R4
A			1	1
B	2	3	1	
C	2	2		
D			2	
E	2		3	2

Esigenza residua (dopo l'assegnazione attuale)				
	R1	R2	R3	R4
A	6	5	1	4
B	5	1	5	3
C	2	1	0	2
D	3	3	3	2
E	2	2	0	1

Molteplicità			
R1	R2	R3	R4
7	6	7	3

Disponibilità			
2	1	0	2

a1) Il processo C può terminare; la disponibilità di {R1, R2, R3, R4} diviene {4,3,0,2}
a2) Il processo E può terminare ; la disponibilità di {R1, R2, R3, R4} diviene {6,3,3,4}
a3) Il processo D può terminare; la disponibilità di {R1, R2, R3, R4} diviene {6,3,5,4}
b4) Il processo B può terminare ; la disponibilità di {R1, R2, R3, R4} diviene {8,6,6,4}
b5) Il processo A può terminare ; la disponibilità di {R1, R2, R3, R4} diviene {8,6,7,5}

Stato sicuro? **SI**
Risorsa assegnata? **SI**
Stato di di B? **ESECUZIONE**

Esercizio 4 (4 punti)
Spiegare la differenza tra lock e spinlock. (max. 5 righe):

_____

_____

_____

_____

_____

In un sistema che implementa i thread a livello del nucleo, dire in quale stato del processore (utente/supervisore o entrambi) possono essere eseguite (o ha senso eseguire) le seguenti operazioni:

operazione	**Stato del processore: [utente/supervisore/entrambi]**
Abilitare le interruzioni	
Invocare l'istruzione SVC di invocazione di una chiamata di sistema	
Modificare il contenuto dello stack	
Modificare il vettore di interruzione	
Invocare l'istruzione TSL (test and set lock – read-modify-write)	
Eseguire il codice della thread_yield()	

Soluzione

operazione	**Stato del processore: [utente/supervisore/entrambi]**
Abilitare le interruzioni	Suvervisore
Invocare l'istruzione SVC di invocazione di una chiamata di sistema	Utente
Modificare il contenuto dello stack	Entrambi (nello stack del rispettivo livello)
Modificare il vettore di interruzione	Supervisore
Invocare l'istruzione TSL (test and set lock – read-modify-write)	Entrambi
Eseguire il codice della thread_yield()	Supervisore

Esercizio 5 (5 punti)
Si considerino due thread consumatori P1 e P2 e due thread produttori C1 e C2 che cooperano utilizzando un buffer condiviso a tre posizioni, utilizzando il seguente codice:

Codice consumatori	Codice produttori

while TRUE {
1. P(empty);
2. P(mutex);
<preleva un dato dal buffer>
3. V(mutex);
4. V(full);
<*usa dato*>
}
	while TRUE {
	<*prepara nuovo dato*>
	a. P(full);
	b. P(mutex);
	<deposita dato nel buffer>
	c. V(mutex);
	d. V(empty);
	}

Ipotizzando che il thread in esecuzione resti in tale stato finchè non si interrompe spontaneamente (il sistema operativo non forza il prerilascio del processore), si considerino i seguenti casi (da considerare **in alternativa**):
caso 1: Ad un certo istante di tempo lo stato del sistema è il seguente:

- P1 è sospeso sulla linea 1;
- P2 è sospeso sulla linea 2;
- C1 è **in esecuzione** e deve eseguire la linea c;

- C2 è pronto per eseguire la linea a;

Dire cosa come cambia lo stato del sistema a partire dal tempo t, per 5 linee di codice eseguite o fino all'istante in cui il processo C1 si sospende. In ogni riga mostrare lo stato risultante dopo l'esecuzione dell'istruzione corrispondente.

Stato C1	Linea eseguita	Empty (valore, coda)	Full (valore, coda)	Mutex (valore, coda)
Esecuzione	**prima di eseguire c**	**0, P1**	**1, -**	**0, P2**
	c			
	d			
	a			
	b			
	c			

caso 2: Ad un certo istante di tempo lo stato del sistema è il seguente:

- P1 è pronto per eseguire la linea 4;
- P2 **è in esecuzione** e deve eseguire la linea 1;
- C1 è sospeso sulla riga a;
- C2 è pronto per eseguire la linea a;

Mostrare lo stato del sistema al termine dell'esecuzione di ogni linea di codice numerata eseguita da P2, per 5 linee di codice eseguite o fino a che P2 si sospende. In ogni riga mostrare lo stato risultante dopo l'esecuzione dell'istruzione corrispondente.

Stato P2	Linea eseguita	Empty (valore, coda)	Full (valore, coda)	Mutex (valore, coda)
Esecuzione	**Prima di eseguire 1**	**2, -**	**0, C1**	**1, -**
	1			
	2			
	3			
	4			
	1			

Soluzione

caso 1:

Stato C1	Linea eseguita	Empty (valore, coda)	Full (valore, coda)	Mutex (valore, coda)
Esecuzione	**prima di eseguire c**	**0, P1**	**1, -**	**0, P2**
Esecuzione	**c**	0, P1	1, -	0, -
Esecuzione	**d**	0, -	1, -	0, -
Esecuzione	**a**	0, -	0, -	0, -
Sospeso	**b**	0, -	0, -	0, C1

caso 2:

Stato P2	Linea eseguita	Empty (valore, coda)	Full (valore, coda)	Mutex (valore, coda)
Esecuzione	**Prima di eseguire 1**	**2, -**	**0, C1**	**1, -**
Esecuzione	**1**	1, -	0, C1	1, -
Esecuzione	**2**	1, -	0, C1	0, -
Esecuzione	**3**	1, -	0, C1	1, -
Esecuzione	**4**	1, -	0, -	1, -
Esecuzione	**1**	0, -	0, -	1, -

Esercizio 6 (3 punti)

Spiegare in cosa consiste lo stato di Zombie in Unix e in che modo un processo può entrare in questo stato (max. 5 righe):

Sistemi Operativi e Laboratorio, Prova del 4/4/2019 – versione A

Si consideri il seguente frammento di codice:

```
…
a = fork();
if (a > 0) {
        b=fork();
        if (b <= 0) { printf("B"); }
        else {
            c= wait(&status);
            printf("E");
        }
else {
        printf("D");
        exit(1);
}
printf("F");
…
```

Il processo che esegue l'intero frammento è quello che esegue le due fork, cioè il padre.
Il processo generato con la prima fork è il primo figlio.
Il processo generato con la seconda fork è il secondo figlio.
Dire cosa stampano il padre e i due figli considerando i seguenti casi di successo e fallimento delle fork:

a) Se entrambe le fork hanno successo:
Il padre stampa: _____
Il primo figlio stampa: _____
Il secondo figlio stampa: _____

b) Se la prima fork ha successo e la seconda fallisce:
Il padre stampa: _____
Il primo figlio stampa: _____
Il secondo figlio stampa: _____

c) Se la prima fork fallisce:
Il padre stampa: _____
Il primo figlio stampa: _____
Il secondo figlio stampa: _____

Soluzione
a) Se entrambe le fork hanno successo:
Il padre stampa: EF
Il primo figlio stampa: D
Il secondo figlio stampa: BF

b) Se la prima fork ha successo e la seconda fallisce:
Il padre stampa: BF
Il primo figlio stampa: D
Il secondo figlio non esiste.

c) Se la prima fork fallisce:
Il padre stampa: D
Il primo figlio non esiste.
Il secondo figlio non esiste.

Esercizio 7 (3 punti)

In un sistema con risorse R1, R2, R3, R4, R5 e R6 (tutte di molteplicità 2), sono presenti i processi P2, P1 e P3 che inizialmente non possiedono risorse e successivamente avanzano senza interagire reciprocamente e alternandosi nello stato di esecuzione con velocità arbitrarie.

Nel corso della propria esistenza, ciascun processo esegue una propria sequenza di richieste, che possono intercalarsi in modo arbitrario con quelle degli altri processi. Dopo aver ottenuto e utilizzato le risorse che richiede, ogni processo termina rilasciando tutte le risorse ottenute.

Si consideri la sequenza di richieste sotto riportate:

Processo	Prima richiesta	Seconda richiesta	Terza richiesta	Quarta richiesta	Terminazione
P1	1 istanza di R3	1 istanza di R5	1 istanza di R4	1 istanza di R1	Rilascia tutto
P2	1 istanza di R6	1 istanza di R5	1 istanza di R2	1 istanza di R4	Rilascia tutto
P3	1 istanza di R2	2 istanze di R4	1 istanza di R1	1 istanza di R5	Rilascia tutto

Dire se i processi evitano la possibilità di stallo e motivare la risposta.

Soluzione

I processi evitano lo stallo [SI/NO]? _____

Motivazione [se evitano lo stallo spiegare perché. Se non evitano lo stallo mostrare una sequenza che li porta in stallo]:

_____

_____

_____

_____

Soluzione

I processi non evitano lo stallo perché violano il vincolo di assegnazione ordinata.

Una sequenza di richieste e assegnazioni che provoca lo stallo è:

P3 – 1 istanza di R2
P3 – 2 istanze di R4
P3 – 1 istanza di R1
P1 – 1 istanza di R3
P1 – 1 istanza di R5
P2 – 1 istanza di R6
P2 – 1 istanza di R5
P3 – 1 istanza di R5 e si sospende
P1 – 1 istanza di R4 e si sospende
P2 – 1 istanza di R2
P2 – 1 istanza di R4 e si sospende

Sistemi Operativi e Laboratorio, Prova del 4/4/2019 – versione A

Nome: _____ Cognome: _____ Matricola:_____ corso:____

Esercizio 8 (4 punti):
Dati i seguenti 3 files (myinc.h, myinc.c e myprog.c):

myinc.h

```
#if !defined(MYINC_H_)
#define MYINC_H_
#include<stdlib.h>
#include<stdio.h>
int G(int);
#endif
```

myinc.c

```
#include <myinc.h>
static int X=0;
static int F(int* Y) {return X+=*Y; }
int G(int Y) { return F(&Y); }
```

myprog.c

```
#include <myinc.h>
int main(int argc, char*argv[]) {
  if (argc!=2) return -1;
  int n=strtol(argv[1],NULL,10);
  int s=0;
  for(int i=1;i<=n;++i) s+=G(i);
  printf("s= %d\n", s);
  return 0;
}
```

completare il seguente mini **Makefile** per generare l'eseguibile **myprog**:

```
CC=gcc
AR=ar
SRC=$(PWD)
CFLAGS= -std=c99 -Wall -g
ARFLAGS=rvs
INCLUDE_DIR=$(SRC)/includes
LIB_DIR=$(SRC)/libs
TARGET=myprog
.PHONY= ……                                  # indicare i phony targets

%.o: %.c
        $(CC)  ……. -c -o $@ $<              # completare la regola

all:   ……                                  # completare la regola

myprog: ……                                 # inserire la dependency list
        $(CC) …… -o $@ $<  ……              # completare la regola

$(LIB_DIR)/libmyprog.a: ……...               # inserire la dependency list
       $(AR) $(ARFLAGS) $@ $<

clean:
      \rm -f *.o *~ $(LIB_DIR)/libmyprog.a

cleanall: clean
     \rm -f $(TARGET)
```

Dire se la funzione F è rientrante o meno motivando (brevemente) la risposta.

_____

Qual è l'output del programma se viene lanciato nel modo seguente ./myprog 4 ?

_____

Sistemi Operativi e Laboratorio, Prova del 4/4/2019 – versione A

Soluzione

```
CC=gcc
AR=ar
SRC=$(PWD)
CFLAGS= -std=c99 -Wall -g
ARFLAGS=rvs
INCLUDE_DIR=$(SRC)/includes
LIB_DIR=$(SRC)/libs
TARGET=myprog
.PHONY= all clean cleanall

%.o: %.c
        $(CC) $(CFLAGS) -I$(INCLUDE_DIR)  -c -o $@ $<

all: $(TARGET)

myprog: myprog.o $(LIB_DIR)/libmyprog.a
       $(CC) $(CFLAGS) -I$(INCLUDE_DIR) -o $@ $<  -L$(LIB_DIR) -lmyprog

$(LIB_DIR)/libmyprog.a: myinc.o $(INCLUDE_DIR)/myinc.h
      $(AR) $(ARFLAGS) $@ $<

clean:
     \rm -f *.o *~ $(LIB_DIR)/libmyprog.a

cleanall: clean
    \rm -f $(TARGET)
```

Le due funzioni non sono rientranti perché F legge e scrive la variabile globale X e G usa F che non è rientrante.

L'output di ./myprog 4 è: 20

Esercizio 9 (2 punti):

a) Dire quali sono le principali fasi della compilazione di un programma C descrivendone gli aspetti essenziali (massimo 5 righe) .

_____

_____

_____

_____

_____

b) Riempire la seguente tabella con il nome della directory di un sistema Linux-based corrispondente alla descrizione fornita.

Directory	Descrizione
/	E' la directory root
	Contiene i files di configurazione, il file dei gruppi e delle password.
	Contiene le librerie di sistema

	Contiene le directories degli utenti del sistema
	Contiene i programmi, i dati e le librerie ad uso degli utenti
	Contengono i files associati ai devices

Soluzione:

Directory	Descrizione
/	E' la directory root
/etc	Contiene i files di configurazione, il file dei gruppi e delle password.
/lib (o /lib64)	Contiene le librerie di sistema
/home	Contiene le directories degli utenti del sistema
/usr	Contiene i programmi, i dati e le librerie ad uso degli utenti
/dev	Contengono i files associati ai devices