DEFINIZIONE:

Sia d un dato condiviso dai processi P1 e P2

Siano o1 e o2 operazioni su d eseguite da P1 e P2.

Siano f1 e f2 funzioni tali che, se n ha valore Vn;

•f1(Vd) è il valore assunto da n eseguendo o1;

•f2(Vd) è il valore assunto da n eseguendo o2;

Le operazioni o1 e o2 danno luogo ad una race condition sul dato condiviso d se, eseguendo o1 e o2 con ndavente un valore iniziale Vd, accade che d assume un valore V’d diverso da f1(f2(Vd)) e da f2(f1(Vd)).

Si ha la race condition quando l’array può assumere, per esempio, il valore .. che è diverso dai due ammissibili.

Non si ha la race condition quando si assumono gli stessi valori.

Assumiamo un array di int A inizializzato con [10,20].

* Un thread esegue l’operazione A[1]=A[2]+5.
* L’altro thread A[2]=A[1]+55.

Si argomenti in modo formale se possono verificarsi race condition su A.

Funzione che formalizza l’operazione del primo thread:

f1([m,n]) = [n+5,n];

Funzione che formalizza l’operazione del secondo thread:

f2([m,n]) = [m,m+55]

Nota: in particolare, vale che:

f1([10,20]) = [20+5,20] = [25,20] e

f1([10,65]) = [65+5, 65] = [70,65]

Nota: in particolare, vale che:

f2([10,20]) = [10,10+55] = [10,65] e

f2([25,20]) = [25, 25+55] = [25,80]

Valori ammissibili su A, cioè valori che otterremmo in caso di esecuzione sequenziale delle due operazioni:

f2(f1([x,y])) = f2([y+5,y]) = [y+5, y+60]

e

f1(f2([x,y]) = f1([x,x+55]) = [x+60, x+55]

nel nostro esempio:

f2(f1([10,20])) = f2([20+5, 20]) = [20+5,20+60]

f1(f2([10,20])) = f1([10,10+55]) = [10+60, 10+55]

possiamo avere race condition perché l’array può assumere per esempio il valore [25,20] oppure [10,65] che non sono ammissibili

Assumiamo che due thread condividano una variabile double X, con valore iniziale X=100.

* Il primo thread esegue l’istruzione X=50;
* Il secondo thread esegue l’istruzione if (X>70) { X=200; } else { X=0; }

Discutere, formalmente, se è possibile che si verifichino race condition sulla variabile X.

Funzione che formalizza l’operazione del primo thread:

f1(x) = 50

Funzione che formalizza l’operazione del secondo thread:

f2(x) = 200 se x>70

f2(x) = 0 se x<=70

Pertanto, componendo le funzioni come segue, spieghiamo il comportamento dei thread quando la loro attività viene sequenzializzata in ordine arbitrario:

f1(f2(x)) = f1(200) = 200 se x>70

f1(f2(x)) = f1(0) = 50 se x<=70

e

f2(f1(x)) = f2(50)) = 0

In particolare, avendo tenendo conto che il valore iniziali di X è 100 vale che:

f1(f2(100) = f1(200) = 50

e

f2(f1(100) = f2(50) = 0

Possiamo pertanto dire che avremo race condition se e solo se X può acquisire valori diversi da 50 e 0.

Una possibile esecuzione è la seguente: il secondo thread testa la guardia (X>70), che risulta vera, perde il processore, il secondo thread esegue l’assegnamento, il primo thread riacquisisce il processore e assegna il valore 200 alla variabile X, che non verrà modificato in seguito. Essendo 200 diverso da 50 e da 0, concludiamo che questa esecuzione da luogo a race condition.

Si assuma la variabile X condivisa da due thread che invocano il seguente metodo M:

* public void M( ) { if ( X >= 0 ){ X = - 5; } else { X = 12; }

Discutere formalmente se si possono verificare race condition su X.

L’effetto sulla variabile X dell’esecuzione di ognuno dei due thread può essere formalizzato con la funzione f: R—> R definita, per parti, come segue:

f(y) = 12, se y < 0.

f(y) = -5, se y >= 0.

Pertanto, il comportamento dei due thread che eseguono sequenzialmente può essere formalizzato applicando due volte f:

f(f(y)) = f(12) = -5, se y < 0.

f(f(y)) = f(-5) = 12, se y >= 0.

Pertanto, se il valore iniziale di X è non negativo, per non avere race condition la variabile X deve assumere valore finale 12.

Altrimenti, se il valore iniziale di X è negativo, per non avere race condition la variabile X deve assumere valore finale -5.

Assumiamo che inizialmente X valga 10. Un thread potrebbe testare la guardia e perdere il processore prima di eseguire il ramo “then”.

L’altro thread potrebbe eseguire tutto il metodo. Il primo thread riprenderebbe in seguito l’esecuzione ed eseguirebbe il ramo “then”.

Alla fine, X varrebbe -5, ma l’unico valore ammissibile partendo da X=10 è 12, pertanto in questo caso avremmo race condition.

Supponiamo di avere un array A di interi condiviso da due thread che eseguono:

public void m(){ A[0] = A[1]+10; A[1] = A[0]+20}.

Spiegare formalmente se possono esserci race condition:

Funzione che formalizza l’operazione del primo thread:

f1([x,y]) = [y+10,y];

Funzione che formalizza l’operazione del secondo thread:

f2([x,y]) = [x,x+20];

Valori ammissibili su A, cioè valori che otterremmo in caso di esecuzione sequenziale delle due operazioni:

f2(f1([x,y])) = f2([y+10,y]) = [y+10, y+30]

e

f1(f2([x,y]) = f1([x,x+20) = [x+30, x+20]

ipotizzando x e y con valore x=2 e y=3

Funzione che formalizza l’operazione del primo thread:

f1([2,3]) = [3+10,3] = [13,3]

f1([2,22]) = [22+10,22] = [32,22]

Funzione che formalizza l’operazione del secondo thread:

f2([x,y]) = [2,2+20] = [2,22]

f2([x,y]) = [13,13+20] = [13,33]

Valori ammissibili su A, cioè valori che otterremmo in caso di esecuzione sequenziale delle due operazioni:

f2(f1([x,y])) = f2([y+10,y]) = [3+10, 3+30] = [13,33]

e

f1(f2([x,y]) = f1([x,x+20) = [2+30,2+20] = [32,22]

possiamo avere race condition perché l’array può assumere per esempio il valore [13,3] oppure [2,22] che non sono ammissibili