Prova 2021.07.21

Esercizio c.1

Per capire con quale squadra deve giocare la squadra i al turno k, mantengo in memoria un albero dove le foglie sono esattamente il numero di squadre, e il padre di due foglie contiene il vincitore. Mantenendo questa struttura aggiornata riesco sempre a trovare l'avversario, capire se è già arrivato fino a quel turno e nel caso aspettarlo.

1. dfs_init dovrebbe inizializzare l'albero, mettere nelle foglie il valore corrispondente della squadra (da 1 a N) e aggiornare l'array foglie
2. l'albero ha nelle foglie gli indici delle squadre (in ordine).
3. foglie è l'array dove `foglie[i]` è il nodo dell'albero della squadra `i`

```
class tree:
    int value = -1
    tree parent = null
    tree left = null
    tree right = null
    condition giocato

monitor Torneo:
    tree albero;
    tree foglie[2^N]

    supporto_numerazione_nodi = 0

    def dfs_init(tree nodo, int depth):
        if depth = 0:
            foglie[supporto_numerazione_nodi] = nodo
            nodo.value = supporto_numerazione_nodi++
        else:
            nodo.left = new tree;
            nodo.right = new tree;
            dfs(nodo.left, depth-1)
            dfs(nodo.right, depth-1)


    def inizializzazione():
        dfs_init(albero, N)


    def gioca(i, turno, forma):
        squadra = foglie[i]
        tree tmp = squadra
```

```
        avversario = NULL

        #trovo l'avversario, sicuramente le partite fino all'indice i escluso sono state go
        for (x in range(turno)):
           tmp = tmp.parent;

        if (tmp.left.value = i):
            avversario = tmp.right
        else:
            avversario = tmp.left

        if avversario.value = -1:
            #non ho ancora un avversario, lo aspetto
            avversario.giocato.wait()
        else:
            #se sono qua l'avversario già era arrivato e mi stava aspettando
            squadra.giocato.signal()


        if forma > valutaforma(avversario.value, turno):
            squadra.parent.value = i
            return True
        elif: forma == valutaforma(avversario.value, turno):
            x = random(0,2)
            if x == 0:
                squadra.parent.value = i
                return True
            else:
                squadra.parent.value = avversario.value
                return False
        else:
            squadra.parent.value = avversario.value
            return False
```

Esercizio c.2

Questo è troppo difficile!!! Nessuno capisce come farlo.

```
class wrongsem:
    int value = 0, count = 0
    semaphore mutex init 1;
    semaphore s init 0;

    void wV():
        mutex.P()
        if value == 0 && count > 0:
```

```
            s.V()
        else:
            value++
        mutex.V()   #<- va nell'else

    void wP()
        mutex.P()
        if value == 0:
            count++
            mutex.V()
            s.P()
            mutex.P()  #<-
            count--
        else:
            value--
        mutex.V()
```

Esercizio g.1

A

```
1 2 3 4 5 3 3 3 1 5
1 2 3 4 5 6 7 8 9 10
```

- NF = 3
  - 1|2|3 -> 4|2|3 -> 4|5|3 -> 4|5|1
- NF = 4
  - 1|2|3|4 -> 5|2|3|4 -> 1|2|3|4 -> 1|5|3|4

B

Non è a stack poichè esiste una stringa tale per cui aumentando il numero di frame aumentano i page fault, tale stringa è la stringa dell'esercizio.

Esercizio g.2

a) Perché il nome del file non è memorizzato all'interno dell'i-node nei file system tipo UNIX (e.g. ext2/3/4)?

Perchè tramite hardlink potrei avere due nomi che puntano allo stesso i-node.

b) Un bug di tipo buffer overflow consente ad un attaccante di spedire più dati di quelli che il buffer di ricezione può contenere, tracimando così nelle aree di memoria di altre variabili. Come è stato possibile in tanti casi che venisse spedito codice macchina e che il programma vittima dell'attacco lo eseguisse?

Sovrascrivendo l'indirizzo di ritorno posso ritornare in un indirizzo dello stack che ho scritto io e così facendo eseguo codice arbitrario. E' stato possibile perchè non

c'erano misure di sicurezza come uno stack non eseguibile attraverso il canarino (che proteggeva lo stack, se esso viene sovrascritto il programma termina con errore).

c) come fa l'allocazione gerarchica ad evitare che si possano verificare casi di deadlock?

Evita che si verifichi la condizione di attesa circolare. Assumiamo per assurdo che P usi la risorsa r1 e sia in attesa di r2 usata da Q. L'attesa circolare si verifica se Q (in una serie di richieste) sia in attesa di r1, ma ciò conduce all'assurdo perchè o r1 ha priorità minore r2 quindi Q dovrebbe rilasciare r2, o viceversa e P dovrebbe rilasciare r1.

d) Per rendere uno scheduler round-robin più pronto a servire i processi interattivi si decide di dimezzare la durata del time slice. Quali effetti collaterali può avere la scelta?

Se il time slice è troppo breve allora l'alto numero di content switch potrebbe pesare troppo sull'efficienza del sistema.