link testo esame

- Esercizio c1 - Soluzione
- Esercizio c2
- Esercizio g1
- Esercizio g2

Esercizio c1

Un professore di Sistemi Operativi organizza una partita a rubabandiera ai giardini margherita come evento di fine corso. Partecipano alla gara due squadre da MAX studenti ognuna (gli studenti sono numerati da 0 a MAX-1, le squadre si chiamano A e B). Il professore puo' chiamare 1 studente (rubabandiera classico), 2 studenti (uno sulle spalle dell'altro), 3 studenti (seggiolino) o quattro studenti (aeroplano).

Il processo professore e' il seguente:

```
A=0
B=1
process prof:
    rb.nuovapartita()
    while True:
        punteggio = rb.chiama(listarandom(MAX)) # genera una lista di 1, 2, 3 o 4 elementi
        print(punteggio[A], ":", punteggio[B])
        if max(punteggio) == 10:
            break
```

I processi studente/studentessa sono rappresentati dal seguente codice

```
process studente(squadra, numero):
    while True:
        if rb.pronto(squadra, numero):
            break
        rb.allabandiera(squara, numero)
```

Regole (del modello): il prof inizia la partita con *nuovapartita* ponendo il risultato iniziale sullo 0 a 0.

Durante la partita pensa la lista dei numeri ma sblocca gli studenti solo se tutti sono in *pronto*, altrimenti aspetta.

Vince un punto la squadra che per prima arriva alla bandiera (tutti i componenti chiamati dal prof sono *allabandiera*).

La procedure entry *chiama* restituisce il punteggio attuale. La procedure entry *pronto* restituisce 0 normalmente, un valore diverso da zero indica il termine della partita.

Soluzione

```
#define MAX     // numero massimo di studenti per squadra
// A, B sono le 2 squadre //
#define A 0
#define B 1

monitor rubabandiera{
    int punteggio[2];   // array punteggi di ogni squadra A, B
    int pronti[2];      // studenti pronti per squadra
    int numeri;         // quanti numeri chiamati per rubare la bandiera
    int arrivati[2];    // studenti arrivati alla bandiera per squadra
    int chiamati[];     // quali numeri sono stati chiamati per rubare la bandiera
    condition ok2run;       // posso andare a prendere la bandiera?

    void nuovapartita(){
        punteggio[A] = 0;
        punteggio[B] = 0;
    }
    int chiama(int chiamati[]){ // chiamati = 1|2|3|4 studenti per squadra
        this.chiamati = chiamati;
        numeri = len(chiamati);
        if (pronti[A] == MAX-1 && pronti[B] == MAX-1)
            arrivati[A] = 0;
            arrivati[B] = 0;
            ok2run.signal();
        return punteggio;   // a quanto siamo arrivati in questo momento
    }
    int pronto(int squadra, int numero) {
        pronti[squadra]++;
        if (pronti[squadra]>=MAX)
            return 1;   //errore ho più studenti pronti di quanti ce ne siano nella squadra
        if (pronti[squadra] < numeri)
            ok2run.wait();      // devo aspettare che anche i miei compagni siano pronti
        return 0; // ok, aggiunto ai pronti
    }
    void allabandiera(int squadra, int numero){
        arrivati[squadra]++;
        if (arrivati[squadra] == numeri)    // sono arrivati tutti?
            pronti[A] = 0;
            pronti[B] = 0;
            punteggio[squadra]++;
    }
}
```

Esercizio c2

sia dato un servizio di message passing asincrono distratto.

Questo servizio si comporta come un servizio di message passing asincrono ma talvolta dimentica la destinazione.

E' però possibile indicare un processo come "ufficio messaggi smarriti" al quale verranno recapitati tutti i messaggi per i quali il servizio distratto ha dimenticato la destinazione.

```
Void dmsgsend(pid_t dest, msg_t msg); //si comporta come amsgsend ma può dimenticare la dest
msg_t dmsgrecv();       //si comporta come amsgrecv(*)
void dset_lost_n_found(pid_t pid); //indica il processo per i messaggi smarriti.
```

Usando il servizio "distratto" e un processo "ufficio messaggi smarriti", implementare un servizio di message passing standard (senza la selezione del mittente in ricezione, la amsgrecv riceve da qualsiasi mittente).

```
void dmsgsend(pid_t dest, msg_t msg);
msg_t dmsgrecv();
void dset_lost_n_found(pid_t pid);

void amsgsend(msg_t msg, pid_t dest){
    dmsgsend(dest, <msg, getpid()>);
}

msg_t amsgrecv(pid_t ANY){
    <msg, pid> = dmsgrecv(ANY);
    if msg == NULL {
        dset_lost_n_found(pid);
    }
    return msg;
}
```

Esercizio g1

Trovare una stringa di riferimenti infinita (che usi un numero finito di pagine), se esiste, per la quale l'algoritmo LIFO (per il quale la pagina vittima è l'ultima caricata) e MIN si comportino esattamente nello stesso modo.

Trovare una stringa di riferimenti infinita (che usi un numero finito di pagine), se esiste, per la quale l'algoritmo LIFO (per il quale la pagina vittima è l'ultima caricata) e LRU si comportino esattamente nello stesso modo.

Esercizio g2

Rispondere alle domande seguenti: - a) Un i-node di un file system tipo ext2 per un errore viene riportato un numero di link maggiore del reale. Cosa può succedere se si continua ad usare il file system? (perché?) E se il numero di link errato fosse al contrario inferiore al reale cosa potrebbe succedere? (perché?) - b) Perché è necessario usare spinlock in sistemi multiprocessore per implementare kernel di tipo simmetrico (SMP)? - c) Perché nei sistemi reali l'algoritmo di rimpiazzamento second chance (orologio) viene preferito a LRU sebbene il primo non sia a stack e il secondo sì? - d) Perché revocare un'autorizzazione espressa come capability è più difficile che revocare lo stesso diritto quando espresso come access control list?

- a.
- b.
- c.
- d. Perchè nella capability l'autorizzazione è memorizzata nei processi, mentre nell'access control list sono memorizzate in apposite strutture dati.