Esame di Informatica II – Modulo Sistemi Operativi, 19/07/2016

Nome: _____ Cognome: _____ Matricola: _____

Domanda 1:

Scheduler a breve, medio e lungo termine. Descrivere brevemente i compiti, gli obiettivi e gli aspetti architetturali più critici di questi tre moduli di un sistema operativo.

Nome: _____ Cognome: _____ Matricola: _____

Domanda 2:

Considerare un sistema la cui memoria è composta da 4 frame, ed in cui è in esecuzione un singolo processo che accede, in sequenza, alle seguenti pagine di memoria:

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

1) Simulare il comportamento delle seguenti politiche di rimpiazzamento delle pagine in memoria:

- First In First Out (FIFO)
- Least Recently Used (LRU)
- Rimpiazzamento ottimale

In particolare, riportare per ogni politica

- lo stato di allocazione dei frame al momento di ogni richiesta di accesso
- le richieste di accesso che causano page fault
- il numero totale di page fault

riempiendo le tabelle alla pagina seguente.

2) Immaginando di basarsi unicamente sui risultati di questa simulazione, quale politica potrebbe essere la più opportuna per un'implementazione **reale** del sistema? Perchè?

FIFO

Accesso	**1**	**2**	**3**	**4**	**1**	**2**	**5**	**1**	**2**	**3**	**4**	**5**
Frame 1												
Frame 2												
Frame 3												
Frame 4												
Page Fault?												

Numero totale di Page Fault: _____

LRU

Accesso	**1**	**2**	**3**	**4**	**1**	**2**	**5**	**1**	**2**	**3**	**4**	**5**
Frame 1												
Frame 2												
Frame 3												
Frame 4												
Page Fault?												

Numero totale di Page Fault: _____

Rimpiazzamento ottimale

Accesso	**1**	**2**	**3**	**4**	**1**	**2**	**5**	**1**	**2**	**3**	**4**	**5**
Frame 1												
Frame 2												
Frame 3												
Frame 4												
Page Fault?												

Numero totale di Page Fault: _____

Nome: _____ Cognome: _____ Matricola: _____

Domanda 3:

Data la seguente porzione di codice, che rappresenta una parte dell'implementazione della classe CircularBuffer, fornire un'implementazione dei metodi:
- insert: consente di inserire (in modo circolare) un nuovo elemento all'interno del circular buffer, se non è pieno;
- remove: consente di rimuovere l'elemento più vecchio dal circular buffer, se non è vuoto.

L'implementazione fornita dovrà garantire che le operazioni effettuate sulla struttura dati buffer avvengano in mutua esclusione.
Inoltre, un ipotetico thread Consumer deve rimanere in attesa se il buffer è vuoto fino a che non viene effettuata una insert.
Un ipotetico thread Producer, invece, deve rimanere in attesa se il buffer è pieno fino a che non viene effettuata una remove.
Come meccanismo di sincronizzazione, utilizzare i Lock e le Condition del package java.util.concurrent.locks.

```
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;


public class CircularBuffer<T> {


        private static final int BUFFER_SIZE = 5;

        private int count;    // numero di elementi nel buffer
        private int in;       // indice della prima posizione libera del buffer
        private int out;      // indice del prox elem. da rimuovere dal buffer
        private List<T> buffer;

        // Condition variables
        final Lock lock = new ReentrantLock();
        final Condition notFull = lock.newCondition();
        final Condition notEmpty = lock.newCondition();

        public BoundedBuffer() {
                // buffer is initially empty
                count = 0;
                in = 0;
                out = 0;
                buffer = new ArrayList<T>(BUFFER_SIZE);
        }
```

```
        @Override
        public void insert(T item) throws InterruptedException {
        // TODO: implementazione dell'operazione inserimento thread safe









        }

        @Override
        public T remove() throws InterruptedException {
        // TODO: implementazione dell'operazione rimozione thread safe









        }
}
```