Sistemi Operativi L-A
Compito di Martedì 15 Gennaio 2008
CdS in Ingegneria Informatica - Prof. Paolo Bellavista

Compito A - Parte di Programmazione di Sistema (16 punti)

Si scriva un programma C che, utilizzando le System Call del sistema operativo UNIX, abbia un'interfaccia del tipo:

```
giocoDeiDadi mosse vincitore
```

dove **mosse** è un nome assoluto di file esistente nel file system mentre **vincitore** è il nome relativo del file che deve essere creato dal programma concorrente nel direttorio corrente. Dopo aver effettuato gli opportuni controlli sui parametri di invocazione, il processo iniziale **P0** (Player 1) deve generare altri 2 processi **P1** (Player2) e **P2** (Coordinatore del gioco), entrambi figli di **P0**.

Il file **mosse** contiene 30 righe che rappresentano ciascuna la somma dei valori ottenuti tramite il lancio di una coppia di dadi secondo il seguente formato: ad esempio, "4+1\n 6+5\n 3+5\n…". Le prime 10 righe rappresentano i risultati dei 10 lanci che, turno dopo turno, saranno giocati da **P0**; il secondo slot di 10 righe rappresenta i 10 lanci che saranno effettuati da **P1**; infine le ultime 10 righe costituiscono i 10 lanci di **P2**.

Ad ogni turno, ciascuno dei tre processi effettua il suo lancio (secondo lo schema indicato sopra) e si aggiudica il turno il giocatore che ha ottenuto la somma più alta. Si preveda che **P0** e **P1** comunichino a **P2** la somma ottenuta (secondo il formato di rappresentazione che si ritiene più opportuno); non è invece necessario che il coordinatore **P2** comunichi agli altri la sua somma. Ricevute le informazioni, **P2** aggiudica il turno a uno dei processi e gli comunica/segnala chi ha vinto quel turno perché ha effettuato la somma più alta. Ogni processo autonomamente mantiene un proprio contatore con il numero dei propri turni vinti. Al termine dei 10 turni, il processo che si è aggiudicato il maggior numero di turni deve scrivere sul file **vincitore** "Sono il processo PIDx e ho vinto", mentre gli altri due processi dovranno scrivere sullo stesso file "Sono il processo PIDy e sono stato battuto"; non ci sono obblighi sull'ordine con cui le righe sono scritte su **vincitore**.

In ogni istante, deve essere possibile per l'utente forzare l'uscita dal gioco di **P0** premendo la combinazione di tasti <CTRL-C> o di **P1** tramite la combinazione di tasti <CTRL-Z>. In entrambi i casi la partita deve essere considerata terminata, avrà vinto il processo che in quel momento ha il numero più alto di prese, e i tre processi devono scrivere su **vincitore** l'esito della partita come nel caso normale di terminazione di tutte le mosse.

Si facciano le scelte di sincronizzazione dei processi ritenute più opportune, cercando di sequenzializzare il meno possibile le varie operazioni richieste.

Compito A - Parte di Programmazione
Java Thread (8 punti)

Si scriva un programma concorrente che utilizzi i Java Thread per simulare una stazione di rilevazione di condizioni meteorologiche.

Il sistema di rilevazione riceve dati da tre tipologie di sensori differenti (*thread Sensore.java*): di pressione, di temperatura e di umidità. Quando sono disponibili dati da TUTTI E TRE i sensori, un esperto meteorologo (*thread Meteorologo.java*) preleva un campione per ciascuna tipologia di dato ed elabora un bollettino (stampa a video dei tre valori rilevati).

Nella stazione meteorologica, ad ogni istante può essere presente al più un solo campione per ogni tipologia di sensore: un sensore che tenti di pubblicare dati di una tipologia già presente all'interno della stazione, deve attendere la diramazione del bollettino prima di poter inviare il proprio campione. Inoltre, per evitare che i sensori pubblichino dati obsoleti, la stazione meteorologica scarta tutti i campioni per cui la differenza tra l'istante di richiesta di pubblicazione e l'effettiva acquisizione del dato sia maggiore di una data soglia *T_MAX*.

I file allegati *Launcher.txt*, *Sensore.txt* e *Meteorologo.txt* contengono le implementazioni complete, rispettivamente, del programma principale, della classe *Sensore* e della classe *Meteorologo*. Inoltre, il file *Stazione.txt* contiene una bozza della classe corrispondente da completare: i candidati implementino i metodi mancanti *generaBollettino()* e *inviaCampione()*, secondo le specifiche di sincronizzazione illustrate in precedenza.

N.B.: per reperire gli istanti di tempo da utilizzare nel meccanismo di filtro dei dati obsoleti, si utilizzi il seguente codice:

```
long current_millisecs = new java.util.Date().getTime();
```

Il valore rappresenta l'istante di invocazione, con precisione del millisecondo.