Threads

- Unita’ di esecuzione dei programmi
- “processi a memoria condivisa”
  - in un sistema con multitasking basato sui processi un singolo processo puo’ ospitare piu’ threads
  - scheduling richiede overhead minore
  - ITC piu’ semplice di IPC perche’ stesso spazio indirizzi

Threads in Java

- Ogni programma Java ha almeno un Thread
  - esegue la funzione main
- Un Thread e’ un’istanza della classe java.lang.Thread
- Un Thread viene creato su un parametro di tipo Runnable
  - interfaccia java.lang.Runnable
  - definisce il metodo public void run()
  - il metodo run e’ il “main” del nuovo thread

Creazione di Threads

```
public class EsempioThreads implements Runnable
{
        java.util.Random r = new java.util.Random(System.currentTimeMillis());
        public static void main(String[] argv)
        {
                int i = 0;
                for(i=0;i<argv.length;i++)
                {
                        (new Thread(new EsempioThread(), argv[i])).start();
                }
                System.out.println(“Main thread exit”);
        }
        public void run()
        {
                <fai qualcosa di utile>
        }
}
```

Schedulazione di Threads

- La macchina virtuale non specifica se la schedulazione e' time sharing o cooperativa
  - non si puo' contare ne' sulla pre-emption ne' sul completamento di un'attivita'
- al momento in cui si effettua la schedulazione viene garantito il rispetto delle priorita'
  - verra' attivato un thread tra quelli con la priorita' piu' alta
- non viene garantita la fairness della politica di selezione del thread da attivare

Sincronizzazione

- problemi di programmazione concorrente
  - race
  - deadlock
- Java permette sincronizzazione tramite monitors (concurrent Pascal, anyone?)
- ogni oggetto e’ potenzialmente un monitor

Sincronizzazione (Cont.)

- mutua esclusione

```
synchronized(monitor)
{
    <regione critica>
}
```

- metodi mutualmente esclusivi
  - public synchronized void metodoMutex()
  - uso implicito dell'oggetto come monitor
    - metodi statici non possono essere synchronized

Sincronizzazione (Cont.)

- Java permette di mettere esplicitamente un thread in attesa
  - monitor.wait()
  - monitor.notify()
  - monitor.notifyAll()
- Non realizza direttamente le condition variables POSIX

Daemon Threads vs User Threads

- Java permette di definire Threads di tipo Daemon e di tipo user (dafault)
- Un thread di tipo demone e’ sottomesso al Thread che lo ha creato e viene terminato automaticamente se termina il thread creatore
- Un thread user ha una vita “propria”

Gruppi di Threads

- Java permette di definire gruppi di threads
- un gruppo di thread puo’ essere usato
  - per tenere traccia del numero di thread attivi
  - rimuovere threads “in blocco”
  - cambiare la priorita’ di threads “in blocco”
  - controllare l’accesso a threads