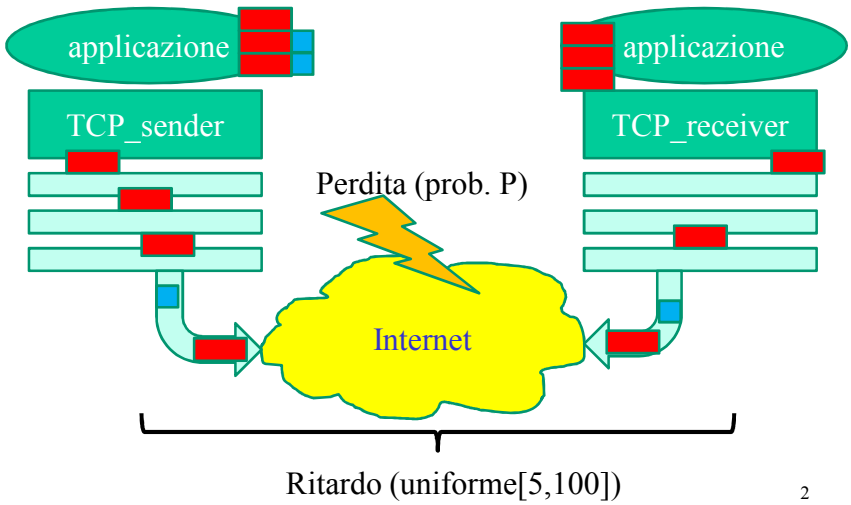
Esempio: modello trasmissione TCP

- Simuliamo un sistema con un sender e un receiver al livello trasporto usando il protocollo TCP, e immaginiamo tutta la rete sottostante come un black-box model caratterizzato da un tempo medio di trasmissione end-to-end e una probabilità di perdita dei dati *p*
- Vediamo i 2 approcci per la simulazione:
  - orientata ai processi
  - ad eventi

1

Esempio: modello trasmissione TCP



2

Interazione tra processi

- Simuliamo un sistema con un sender e un receiver al livello trasporto usando il protocollo TCP, e immaginiamo tutta la rete sottostante come un black-box model caratterizzato da un tempo medio di trasmissione end-to-end e una probabilità di perdita dei dati *p*
- Vediamo i 2 approcci per la simulazione:
  - orientata ai processi
  - ad eventi

3

Introduzione alla Simulazione

- Simulazione per **Interazione tra Processi**
  - es. linguaggi SIMULA, C-SIM, ecc.
  - implementazione di processi pseudo-paralleli
  - reentrant-code (un programma eseguibile da più istanze di processi "diversi", con stato locale), threads...
  - processi eseguiti in MODEL-time (non real time) che può essere "accelerato o rallentato".

4

Introduzione alla Simulazione

- Simulazione per **Interazione tra Processi**
  - Vita di un processo nel modello
    - fase "attiva" di esecuzione (richiede 0 model-time)
      - aggiornamento e gestione stato locale e globale
    - fase "passiva" di esecuzione (in >0 model-time discreto)
      - es. attesa per risorsa o servizio ricevuto
  - ordine di esecuzione dei processi governato dal "calendar" (lista di puntatori a processi ordinati per model-time di "risveglio") e gestito attraverso un processo-scheduler (kernel)

5

Introduzione alla Simulazione

- Simulazione per **Interazione tra Processi**
  - Possibili stati dei processi
    - Passive: il processo è definito, ma non ha nessuna entry relativa nel calendar, cioè non si sa se e quando si "risveglia"
    - Planned: il processo è definito, e ha una entry relativa nel calendar, cioè si sa quando si "risveglia"
    - Active: il processo è eseguito (la sua entry è in testa al calendar)
    - Terminated: il processo è terminato (non può essere riavviato) e i suoi dati sono ancora mantenuti in memoria
  - le transizioni di stato avvengono attraverso l'esecuzione del processo stesso o da parte di terzi

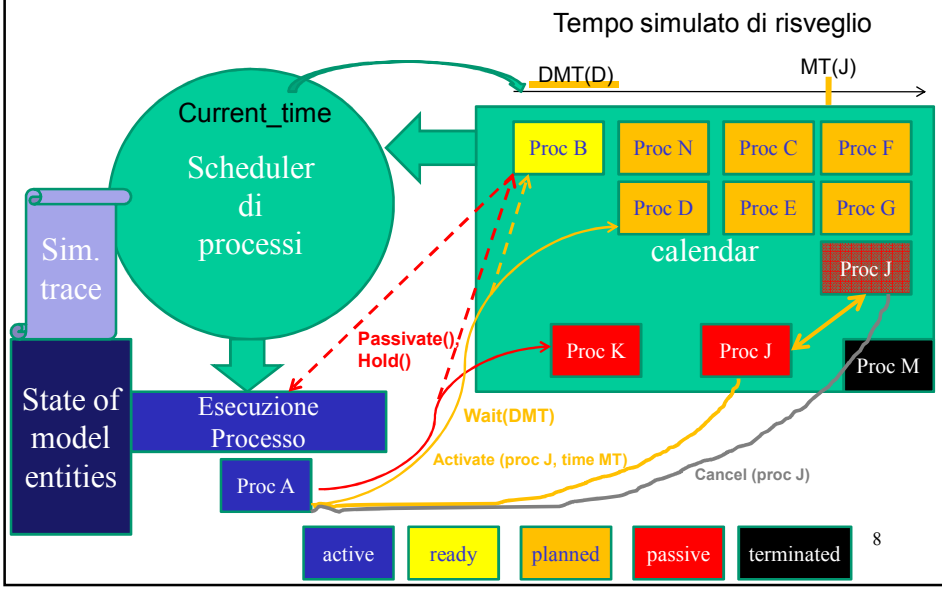
6

Introduzione alla Simulazione

- Simulazione per **Interazione tra Processi**
  - Primitive di transizione di stato dei processi
    - passivate(),hold(): l'attuale processo ATTIVO diventa PASSIVO e viene rimosso dal calendar. Il primo processo in testa al calendar diventa il processo attivo.
    - wait(DMT): il processo x ATTIVO diventa PLANNED al Model-Time attuale + DeltaModelTime (DMT). Il primo processo in testa al calendar diventa il processo attivo.
    - activate(Y,MT): il processo x ATTIVO cambia lo stato del processo Y da PASSIVO a PLANNED(tempo MT). Il processo x rimane ATTIVO.
    - cancel(Y): come activate, ma x ATTIVO cambia Y da PLANNED a PASSIVO.

7

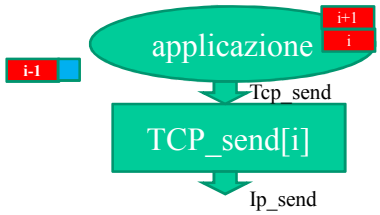
Scheduler e calendar



8

TCP (processi)

```
Tcp_send[i] \\ TCP session process
{
While(true)
{
sent[i] := false
 while(sent=false)
 {
  new IP_Send(Packet, i)
  activate(IP_Send(Packet, i), now)
  new Timer[i]
  activate(Timer[i](t), now)
  new Tcp_receive()
  activate(TCP_receive[i], now)
  passivate() \\ waiting ack or timeout
  \\ svegliato o da timeout o da ricezione ack da livello IP
  if (timeout == false) \\ se svegliato da ack
   {
   activate(Tx_buffer_send(i+1), now)
   sent[i] :=true
   }
 }
passivate() \\ waiting next (i+1) packet to send
}
}
```

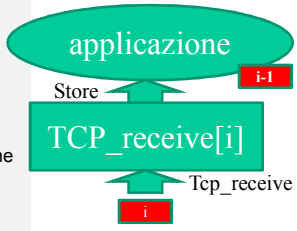



```
IP_Send[i]
{
 received[i] := false
 if RND()>p   \\ pacchetto arrivato
  {
   received[i] := true
   delay[i] := uniform(5,100)
   Wait(delay[i])
   activate(TCP_receive[i], now)
  }
Terminate()
}
```

TCP (processi)

```
Tcp_receive[i]
{
 passivate()
 // svegliato da passaggio pacchetto da livello IP a Tcp o Timeout//

 if timeout[i] == false \\ se svegliato da IP sottostante
 {
    if (OK(Packet, i)) \\ se pacchetto ricevuto ok
       {
       activate(IP_Send(Ack, i), now)   \\ invia ack
       activate(Store(Packet, i), now)    \\ bufferizza pacchetto per applicazione
       }
 terminate()
}
```




```
Timer[i](t)
{
 timeout[i] := false
 Wait(t)            \\ t := TIMEOUT
 if (ack_received[i] == false)
  {
  timeout[i] := true
  activate(Tcp_receive[i], now)
  activate(TCP_send[i], now)
  }
 terminate()
}
```

10

Introduzione alla Simulazione

- Simulazione per **scheduling di eventi**
  - non ci sono processi, ma esecuzione sequenziale di "gestori di eventi"
  - Si avanza il tempo simulato al tempo T dell'evento successivo (di solito il termine o l'inizio di un'attività) e si eseguono tutti gli eventi di T
  - Simulatore = Lista ordinata di eventi e scheduler di eventi
  - Il termine di un'attività coincide con la nuova allocazione di risorse rilasciate tra le entità in attesa e con lo scheduling di nuove attività causalmente determinate

11

Introduzione alla Simulazione

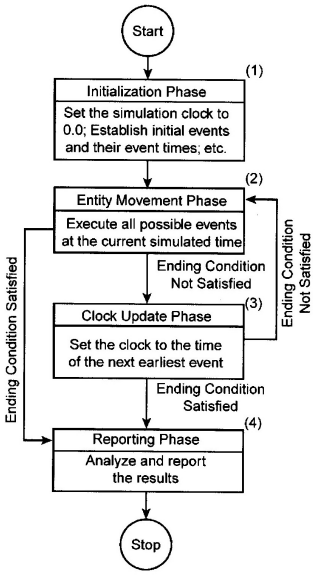
- **scheduler** di eventi
  - mantiene struttura di lista ordinata per tempo simulato (multi-linked) di eventi futuri (schedulati, cancellati, rinviati, bloccati).
  - Gestisce l'avanzamento del tempo simulato
    - event-driven: clock=(tempo del prossimo evento il lista)
    - unit-time: clock= clock+Delta... Sono accaduti eventi? Struttura dati Evento=(time, puntatore al codice della routine di evento)
  - La routine di evento aggiorna le var. di stato e la lista di eventi (inserisce, cancella, o rinvia eventi)

12

Introduzione alla Simulazione

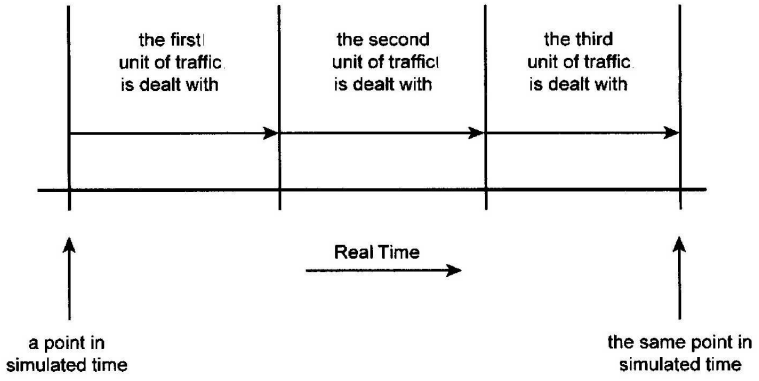
Flusso di esecuzione di un RUN di simulazione DES:

Ciclo tra EMP e CUP



Introduzione alla Simulazione

(Simulated) Model-time e (Real) WallClock-Time



14

Introduzione alla Simulazione

– Entità: possono essere in 5 stati possibili
  - Attivo
    – solo un'entità alla volta (quella in esecuzione al wall clock time attuale)
  - Ready
    – durante una EMP ci possono essere più entità pronte a "muoversi" al tempo T, che diventeranno ATTIVE una alla volta
  - Time-delayed
    – entità che diventeranno di nuovo READY a un tempo futuro già determinato (per le quali la prossima attività è un evento nel futuro)

15

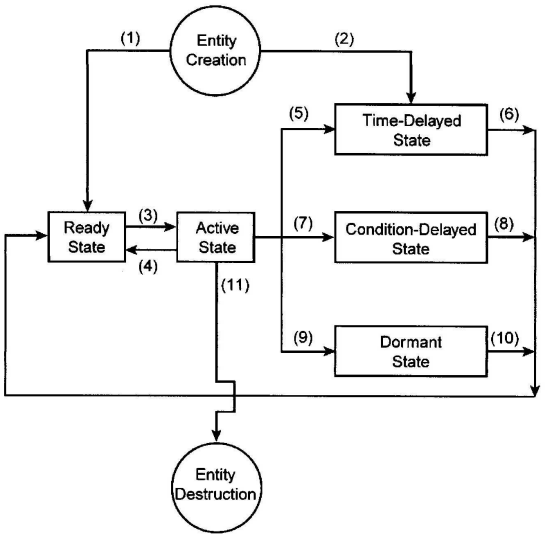
Introduzione alla Simulazione

– Entità: possono essere in 5 stati possibili
  - Condition-delayed
    – entità che diventeranno di nuovo READY a un tempo futuro non determinato, che dipende dal verificarsi di una certa condizione (es. il rilascio di una risorsa)
  - Dormienti
    – sono entità per le quali non è prevista una condizione di risveglio automatico, ma che possono essere rese attive dalla logica prevista dal creatore del modello, a seconda dei casi.

16

Introduzione alla Simulazione

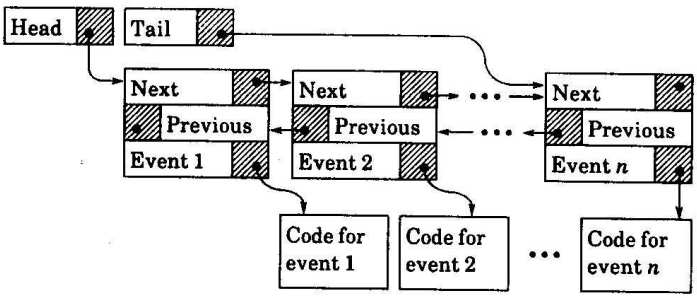
Ciclo dei possibili stati di un entità in un RUN di simulazione DES



17

Introduzione alla Simulazione

Es. di entry di una lista di gestione eventi



18

Introduzione alla Simulazione

Es. di lista di gestione eventi futuri (FEL)

t

t+Δ1

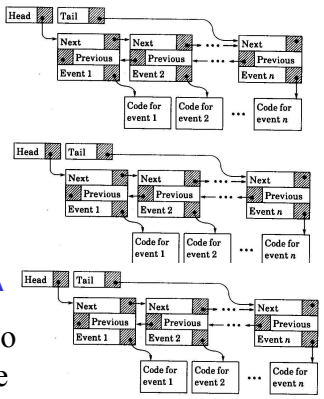
t+Δ2

approccio
event driven
Δ1 != Δ2

t

t+Δ

t+2Δ



approccio
unit-time
Δ costante

19

Introduzione alla Simulazione

- Strutture dati per il controllo delle entità
- tipicamente per ogni stato delle entità esiste una lista relativa
  - lista (singolo elemento) Active Entity (AEL)
    - tale entità viene eseguita fino a che non cambia stato o si elimina dal sistema
  - Current Event List (CEL)
    - contiene tutte le entità ATTIVE (pronte all'esecuzione)
    - contiene le possibili entità "clonate" dalla entità attiva
    - possibili criteri di priorità per diventare ATTIVE

20

Introduzione alla Simulazione

- Future Event List (FEL) (vedi figura Jain)
  - lista di eventi del tempo T1, T2...
  - contiene le entità in stato TIME-DELAYED
  - ordinata per tempo simulato di "movimento" dell'entità
  - quando termina la EMP, il CUP avanza al tempo simulato della prima entry della FEL, poi tutta la lista attuale di FEL viene spostata nella CEL (diventano entità READY), e la EMP riparte a "eseguire" le entità...
- Delay List (condition-delayed entities)
  - related waiting (se conosco le cause o eventi che possono liberare) o polled waiting (controllo ciclico condizioni)
  - entità "liberate" vanno aggiunte alla CEL (ready entity)

21

Introduzione alla Simulazione

- User Managed List
  - lista di entità in stato "dormiente"
  - la gestione e la semantica è a carico del creatore del modello

22

Introduzione alla Simulazione

– Elementi di controllo
  - sono elementi che agiscono da trigger di eventi e rilevano combinazioni di condizioni di stato
  - es. se (coda piena) e Tsimulazione>300 allora...

23

Introduzione alla Simulazione

– es. Problemi di implementazione e gestione
  - risorsa rilascia e tenta di acquisire subito la risorsa
    – 1) allocare la risorsa prima di rendere l'entità un contendente?
    – 2) rendere l'entità un contendente prima di allocare la risorsa?
    – 3) risorsa ri-catturata immediatamente?
  - Il primo della coda viene sempre ritardato
    – A è in coda prima di B per la risorsa R. A chiede 2 R mentre B chiede 1 R... una sola R diventa disponibile: cosa fare?

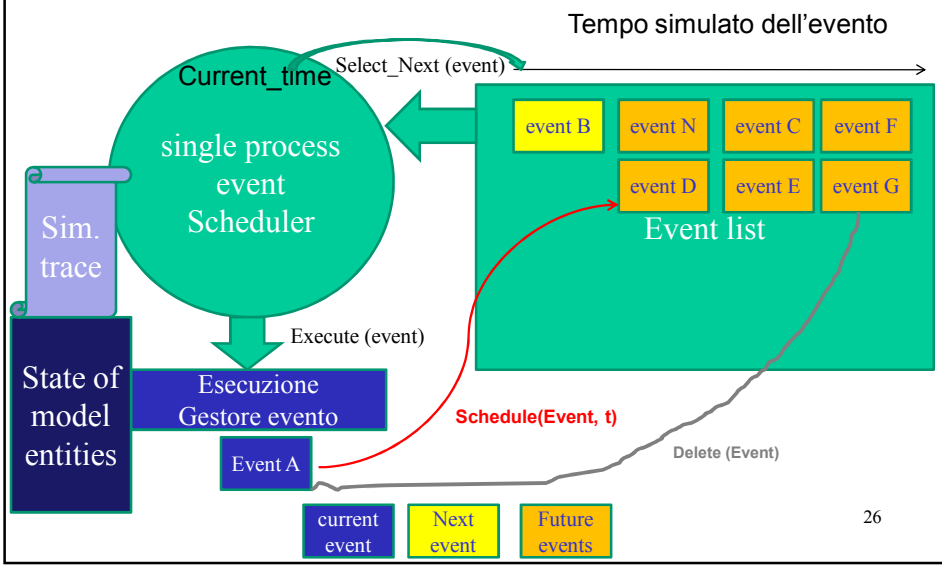
24

Introduzione alla Simulazione

- es. Problemi di implementazione e gestione
  - Yielding control
    - la Active entity vuole dare controllo ad altre Ready entities, poi vuole ri-acquisire il controllo, prima che il CUP avanzi il tempo simulato: come si fa?
    - es. apro porta per 10 entità e poi riblocco la porta...
  - Condizioni di attesa che coinvolgono il clock
    - es. wait until buffer empty OR clock = 1000 (non >=)
    - tipicamente si clonano le entità (dummy entity) e poi si gestiscono in coppia (la prima che si libera elimina l'altra)
  - Condizioni di attesa che coinvolgono lo stato delle risorse
    - es. A wait finchè R viene rilasciata da chi la detiene ora (B)...
    - occorre garantire che la entità A sia sbloccata PRIMA dell'eventuale cattura della risorsa R da parte di altre entità che erano in attesa

25

Scheduler and event list



26

TCP (eventi)

```
Tcp_send[i]
{
  schedule(IP_Send_packet(Packet, i), NOW)
  schedule(Timer_expired(i), NOW+TIMEOUT))
}

Tcp_timeout
{
  schedule(Tcp_send[i]...)
}

Timer_expired(i)
{
Tcp_timeout(i)
}

Tcp_receive \\pacchetto arrivato al lato ricevente TCP
{
   schedule(IP_Send_ack(i), NOW)   \\ invia ack
   schedule(Store(Packet, i), NOW)   \\ bufferizza pacchetto per applicazione
}

Tcp_receive _ack \\ ack arrivato al lato trasmittente TCP
{
   delete(Timer_expired[i])
   schedule(TCP_send(i+1), NOW)
}
```

```
IP_Send_packet[i]
{
 received[i] := false
 if RND()>p    \\ pacchetto arrivato
  {
   received[i] := true
   delay[i] := uniform(5,100)
   schedule((TCP_receive[i]), NOW+delay[i])
  }
}

IP_Send_ack[i]
{
 received_ack[i] := false
 if RND()>p    \\ ack arrivato
  {
   received_ack[i] := true
   delay_ack[i] := uniform(5,100)
   schedule((TCP_receive_ack[i]),
NOW+delay[i])
  }
}
```

27