28/02/2011

Ricevimento: mercoledì 14.30 - 16.30 o su appuntamento

Esame:

A) Prova lab di programmazione di sistema 20%. (fine aprile)

B) Progetto executive real-time 20%. (fine maggio) a coppie

C) Prova scritta sistemi in tempo reale 20%. (inizio giugno)

D) Prova scritta / pratica di concorrenza 30%. (appelli)

E) Orale (breve) 10%. (dopo D)

---

Lettori / Scrittori : weak reader preference monitor

```
type lettori_scrittori = monitor;
var  num_lettori : integer;
     occupato: boolean;
     arrivo: condition;

procedure entry Inizio_lettura();
begin
    if (occupato) arrivo.wait;
    num_lettori ++;
end;

procedure entry Fine_lettura();
begin
    num_lettori --;
    if (num_lettori = 0) arrivo.signal;
end;

procedure entry Inizio_scrittura();
begin
    if (num_lettori > 0) or (occupato) arrivo.wai
    occupato = true;
end;

procedure entry Fine_scrittura();
begin
    occupato = False;
    arrivo.signal;
end;

begin num_lettori = 0; occupato = False; end;

end;
```

Inefficiente! Il lettore non risveglia tutti gli altri!

Soluzione: due code distinte e scelta random! Oppure:

```
type  lettori_scrittori = monitor;
var   num_lettori_attesa: integer;
      num_lettori : integer;
      occupato: boolean;
      ok_prossimo, attesa_lettori : condition;

procedure entry Inizio_lettura();
begin
    num_lettori_attesa ++;
    if (occupato && num_lettori_attesa = 1) ok_prossimo.wait;
    else if(occupato) attesa_lettori.wait;
    num_lettori_attesa --;
    num_lettori ++;
    attesa_lettori.signal;
end;
procedure entry Fine_lettura();
begin
    num_lettori --;
    if (num_lettori = 0) ok_prossimo.signal;
end;
procedure entry Inizio_scrittura();
begin
    if (num_lettori > 0) or (occupato) ok_prossimo.wait;
    occupato = true;
end;
```

due code: una per gli scrittori e il primo lettore, un'altra per i lettori.

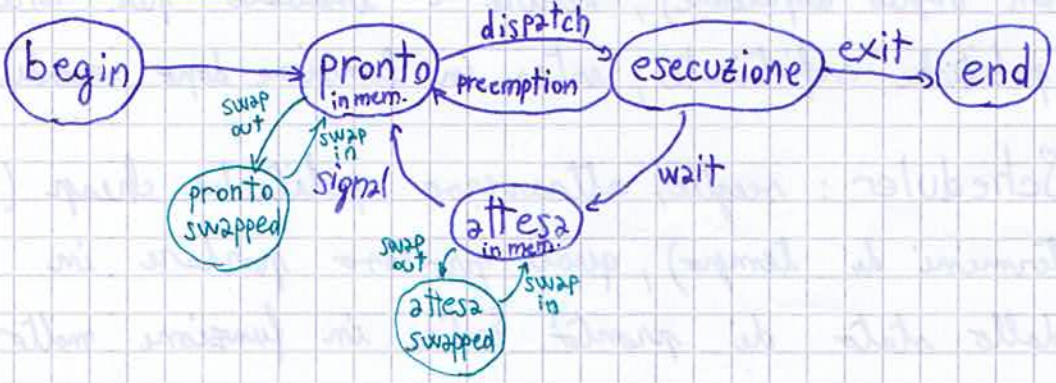
```
procedure entry Fine_scrittura();
begin
    occupato = false;
    ok_prossimo.signal;
end;
begin
    num_lettori = 0;
    num_lettori_attesa = 0;
    occupato = false;
end;
end;
```

# SCHEDULER CPU

## Cosa è lo scheduling della CPU?

↳ La gestione della differenza tra il numero di CPU virtuali e il numero di CPU reali (di solito 1).
Servono dei meccanismi e politiche per capire a chi assegnare la/le CPU

## Concetto di stato di processi e thread




Siccome solitamente la memoria disponibile è inferiore alla dimensione dello spazio di indirizzamento, faccio lo swap in memoria di massa.

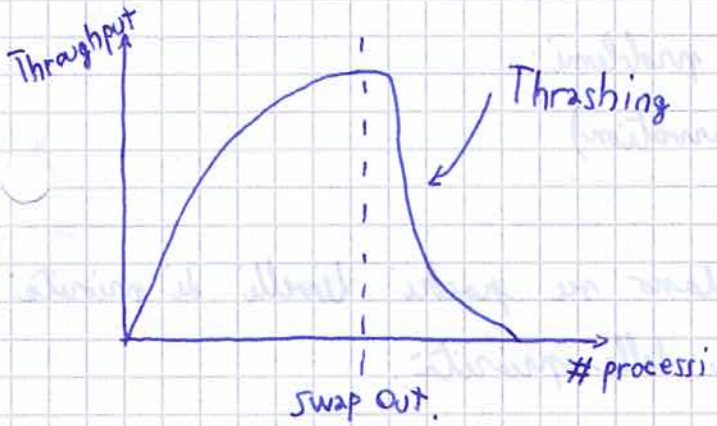
Un altro meccanismo di gestione della memoria è la paginazione, in cui la memoria è divisa in pagine (4 o 8 KB) che vengono caricate/scaricate dalla memoria di massa a tempo di esecuzione. Questo consente di multiplexare la memoria disponibile (i.e. 4 GB) con quella richiesta dai processi.
Per scegliere quale pagina scaricare dalla memoria per fare posto a ciò che mi serve, uso degli algoritmi, ad esempio LRU - Last Recently Used.

N.B. I sistemi real-time "seri" disabilitano o non supportano né paging né swapping. Le pagine dei thread real-time nei sistemi misti sono "pinned", fissate in memoria con una puntina.

## Schedulatori

- Long Term Scheduler: sceglie quali processi portare nello stato di pronto da memoria di massa; ha due funzioni:
  - controllare il grado di multiprogrammazione
  - controllare il job mix.

  Il long term scheduler è assente in sistemi di elaborazione interattivi (non voglio aspettare), mentre è indicato per sistemi batch. Usa politiche sofisticate, entra in funzione dopo secondi...
- Short Term Scheduler: sceglie, attraverso politiche cheap (poco onerose in termini di tempo), quale processo portare in esecuzione dallo stato di pronto. Entra in funzione molto spesso.
- Medium Term Scheduler: dove è assente il long term scheduler, se vede che ci sono troppi page-fault, diminuisce il grado di multiprogrammazione con il swap-out di un processo.




## Algoritmi di schedolazione di base

- First Come First Served + fair - basse prestazioni, effetto "carrello famiglie"
- Shortest Job First + ottimo. - richiede info preventive, starvation, unfair
- Round Robin - quanto massimo, ok per fairness, no starvation, prestazioni intermedie richiede preemption
- Priorità + consente di dare importanza a processi "valuable" - rischio di starvation, tempi medi di attesa, unfair

Posso poi aggiungere la preemption.

In realtà, in un sistema operativo c'è tutto questo nella politica di schedulazione:

- code multilivello con priorità e preemption
- FIFO entro ogni livello di priorità
- FIFO con round robin (quanto: 10-20 ms)
- SJF → lo schedulatore favorisce i processi interattivi che, solitamente, hanno CPU burst brevi, mantenendogli alta la priorità. Un processo che fa molto I/O ha priorità sempre alta, uno che usa molto la CPU avrà priorità decrescente.
- meccanismo di acquisizione e cessione di priorità, per quei processi che richiedono molta CPU all'inizio (la priorità cala) e poi fanno molto I/O: a seconda del tempo di attesa, gli viene aumentata la priorità.
- aging, per i processi in attesa da tempo.

Questo sistema "classico" ha alcuni problemi:
- sono molto sensibili al carico (starvation)
- imprevedibili
- ho tanti processi che si accumulano su pochi livelli di priorità
- overhead dovuto al calcolo frequente della priorità.

Ci sono alcune soluzioni a questi problemi:
- Solaris: prevede tre classi di livelli di priorità: user, system e real time (priorità più alta di tutte; thread pinned)
- Lottery Scheduling: quando un processo viene creato, gli assegno un certo numero di biglietti, che rimpinguo a dovere; la schedulazione consiste nell'estrazione casuale. Previene la starvation (tutti hanno almeno un biglietto)
- Epoch: evoluzione del precedente che assegna anche quanti di tempo ai processi, dividendo il tempo in epoche.
- CFS: Completely Fair Scheduler: aumenta fairness, costo dipende da # proces
- Stride Scheduling
- O(1) Scheduler: costo della schedulazione costante, non dipende dal numero di processi del sistema.

# SISTEMI IN TEMPO REALE

Sistema di elaborazione in grado di rispondere agli eventi rispettando precisi vincoli temporali.

Un sistema in tempo reale è un sistema di elaborazione in cui la correttezza della elaborazione dipende non solo dai risultati prodotti in uscita, ma anche dall'istante in cui tali risultati vengono resi disponibili.

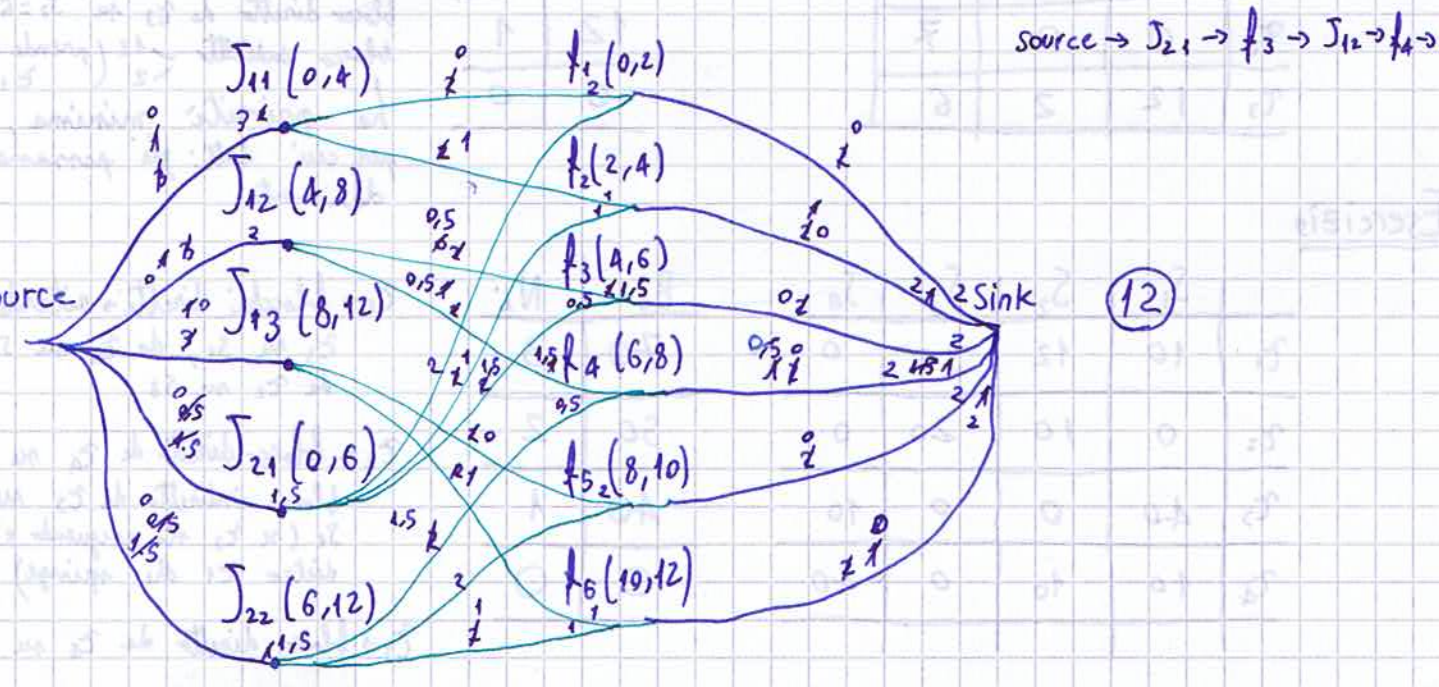
I vincoli temporali possono essere specificati sul singolo job oppure sull'intero task (insieme di job correlati)

Nei sistemi RT l'assegnazione dei task ai processori è statica (solitamente

Se la deadline è più stringente, la varianza diminuisce. Se, invece, la deadline è maggiore del periodo, avrò più task dello stesso tipo.

Task sporadici sono task aperiodici in cui posso garantire che trasc un tempo $T_i$ tra un rilascio e l'altro.

Es. pag 52 slides – m=2



$f_1(0,2) \rightarrow J_{11}$ per 2 unità (slicing)

$f_2(2,4) \rightarrow J_{11}$ per 1 unità
$J_{21}$ per 1 unità

$f_3(4,6) \rightarrow J_{12}$ per 1,5 unità
$J_{21}$ per 0,5 unità

$f_4(6,8) \rightarrow J_{12}$ per 1,5 unità
$J_{22}$ per 0,5 unità

$f_5(8,10) \rightarrow J_{13}$ per 2 unità

$f_6(10,12) \rightarrow J_{13}$ per 1 unità
$J_{22}$ per 1 unità

Martedì 24 e 31/5 esercitazione 13.30 - 16.30
↳ distribuzione assegnamento

## Esercizio

| | $S_1$ | $S_2$ | $S_3$ |
|---|---|---|---|
| $\tau_1$ | 5 | 10 | 0 |
| $\tau_2$ | 0 | 0 | 7 |
| $\tau_3$ | 12 | 2 | 6 |

non accade a $S_3$

massimo blocco ↓ | n° max inversione di priorità ↓

| $B_i$ | $N_i$ |
|---|---|
| 12 | 1 |
| 12 | 1 |
| 0 | 0 |

blocco diretto (priorità max) da $\tau_3$

blocco diretto da $\tau_3$ su $S_2 = 6$
blocco indiretto $\leq 12 - 2$ (prende priorità $\tau_1$)
ha priorità minima, per cui tutti gli passano davanti

## Esercizio

| | $S_1$ | $S_2$ | $S_3$ | $S_4$ |
|---|---|---|---|---|
| $\tau_1$ | 10 | 12 | 30 | 0 |
| $\tau_2$ | 0 | 10 | 20 | 0 |
| $\tau_3$ | 40 | 0 | 0 | 10 |
| $\tau_4$ | 10 | 10 | 0 | 40 |

| $B_i$ | $N_i$ |
|---|---|
| 70 | 3 |
| 50 | 2 |
| 40 | 1 |
| 0 | 0 |

$\tau_1 \rightarrow$ blocchi diretti → ritardato da $\tau_3$ su $S_1$, da $\tau_4$ su $S_2$ e da $\tau_2$ su $S_3$

$\tau_2 \rightarrow$ blocco diretto da $\tau_4$ su $S_2$
blocco indiretto da $\tau_3$ su $S_1$ (se $\tau_3$ sta eseguendo e ha dietro $\tau_1$ che spinge)

$\tau_3 \rightarrow$ blocco diretto da $\tau_4$ su $S_4$

# Esercizi Monitor

## ① Deposito sci

Parto solo quando ho tutte le attrezzature e le riservo solo quando parto.

## ② Boat pooling

Se arriva un linux-hacker solo e tanti microsoft-employee, il linux-hacker avrà starvation.

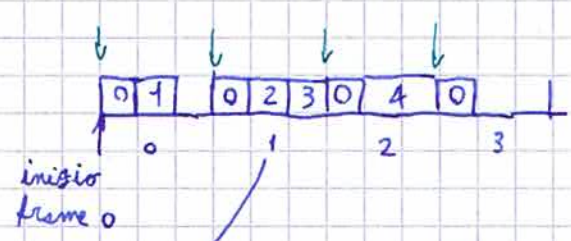
board_avail → la barca è pronta e si può fare l'imbarco
crew_complete → barca al completo.
crew → n° persone nell'equipaggio
crossing → copre attesa completamento equipaggio e attesa trasporto

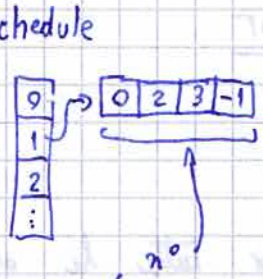
I servitori fanno ACCETTA - SERVI - RILASCIA . ASRASRASR..
Potrei fare RILASCIA&ACCETTA - SERVI, ma la prima volta devo fare solo accetta.

Se hoare devo stare attento a chi risvegliare (se ci sono meno di 2 LH in coda non li risveglio).

## ③ Pizzeria

Gestisco le ordinazioni di più pizze come un'unica ordinazione (durata variabile della preparazione delle pizze).



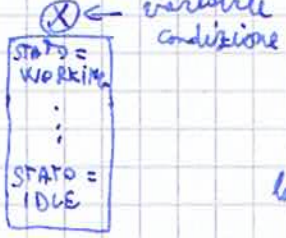

```
SCHEDULE[1] = (int *) malloc (sizeof (int) * 4)
```

void busy_wait (unsigned milli) / qualcosa che occupa la CPU (pausa attiva)
↳ get time of day

```
{
  struct timeval end;
  struct timeval now;
  gettimeofday (&end, NULL);
  end.tv_sec += (end.tv_usec + milli * 1000) / 1000000;
  end.tv_usec = (end.tv_usec + milli * 1000) % 1000000;
  do
    {
    gettimeofday (&now, NULL)
    }
  while ((now.tv_sec < end.tv_sec) || ((now.tv_sec = end.tv_sec) && (now.tv_usec < end.tv_usec)));
}
```

Thread
- stato fun.
- suspended
- thread id
- cond. var.
- mutex

Il thread executive si sveglia solo negli istanti verdi e controlla che i thread precedenti abbiano terminato (sono fermi sul semaforo) e mette in esecuzione i successivi modificandone la priorità per garantire l'ordine. Controllo anche deadline miss (due stati per task: idle o working, modificati dall'executive, oppure pending se il task non ha nemmeno fatto in tempo a partire perché il precedente si è protratto troppo a lungo, fino alla fine del frame)




executive → mette lo stato in pending
task → mette il suo stato in working o idle.

lascio finire
↑

Se STATO[I-1] = IDLE OK, se = WORKING non ha finito (deadline miss) in stato inconsistente, se PENDING faccio kill / rimetto IDLE)
↑ printf.