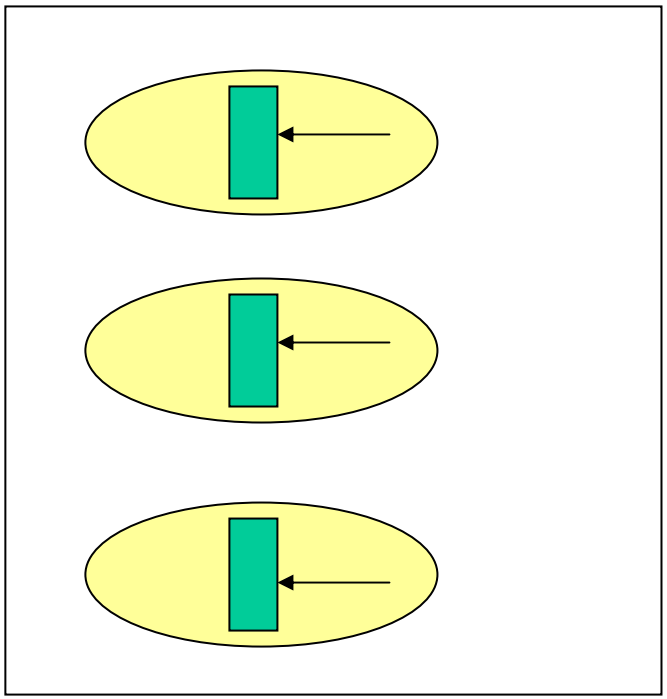
Thread – Luồng

Luồng:

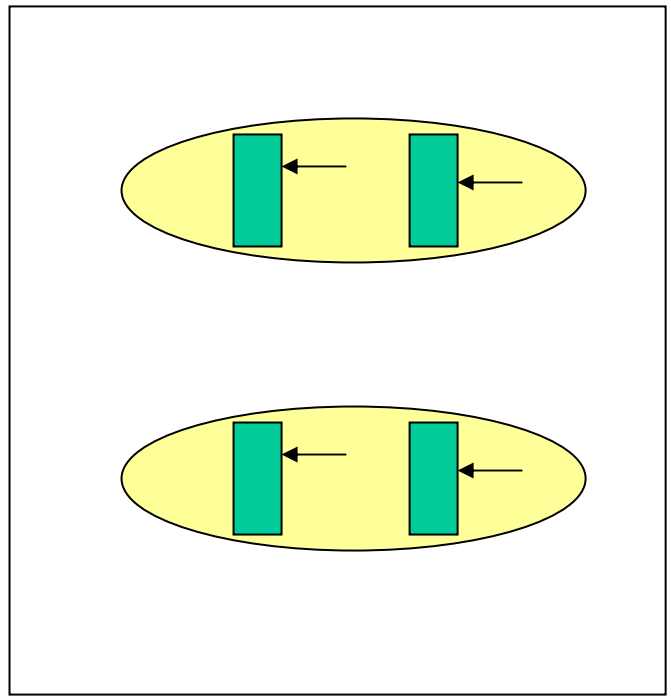
- là một đoạn chương trình chia sẻ cùng một địa chỉ vùng nhớ dữ liệu, mã chương trình, các bảng thông số tệp/tài nguyên đang thao tác
- mỗi luồng đều có stack, ip counter riêng

Khái niệm luồng đối với tiến trình cũng như khái niệm tiến trình đối với máy

Ví dụ về luồng



Tiến trình với 1 luồng và...



Tiến trình với đa luồng

So sánh luồng và tiến trình

Luồng	Tiến trình
•program counter •stack •tập thanh ghi •luồng con •trạng thái chạy	•không gian địa chỉ •biến tổng thể •các files mở •tiến trình con •bộ đếm thời gian •tín hiệu •semaphore •thông tin kiểm toán

Vì sao cần luồng?

- Trường hợp một file server: nhiều client cùng yêu cầu đọc/ghi tệp
  - giải pháp: nhiều tiến trình trên server
  - giải pháp: 1 tiến trình, lưu giữ bảng các trạng thái chạy (finite-state automata)
  - giải pháp: một tiến trình, phân việc cho nhiều luồng

So sánh các mô hình

Mô hình	Đặc tính
Luồng	Song song, blocks system calls
Finite-state machine	Song song, không block system calls
Tiến trình đơn luồng	Không song song, blocks system calls

Ưu điểm sử dụng luồng

- Cho phép lập trình theo phương thức tuần tự, giải quyết bài toán xử lý song song
- Chia sẻ được bộ nhớ cache, tăng hiệu quả xử lý

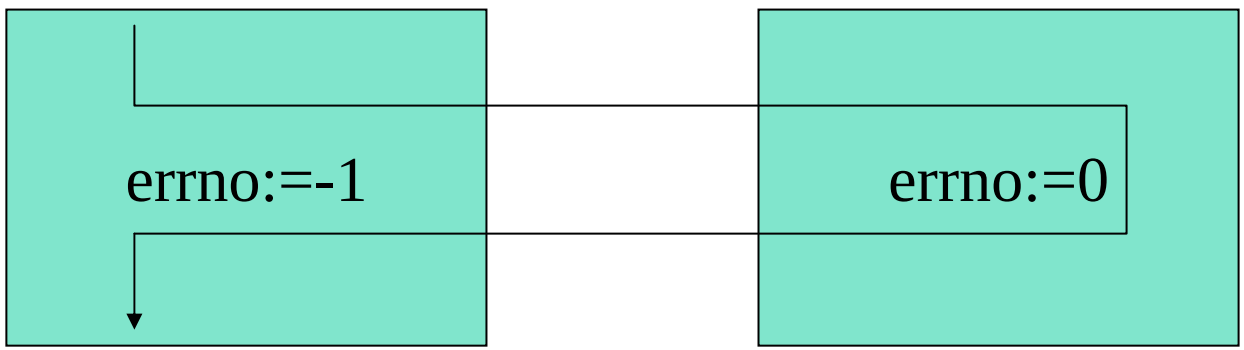
Vấn đề loại trừ lẫn nhau

- Sử dụng semaphore 2 trạng thái *mutex*
- Sử dụng biến điều kiện (condition variable) cho phép thực hiện *wait* và mở khoá *mutex* cùng lúc
- Ví dụ:

lock mutex kiểm tra tài nguyên while (tài nguyên bận) wait(condition variable) đánh dấu tài nguyên bận unlock mutex	lock mutex giải phóng tài nguyên unlock mutex wakeup(condition variable)

Vấn đề biến tổng thể

- Các luồng đều có cùng chung biến tổng thể. Sự cố:
  - lỗi gọi hàm trong 1 luồng được trả lại trong biến errno
  - CPU được chuyển sang cho 1 luồng khác và errno có thể bị ghi đè



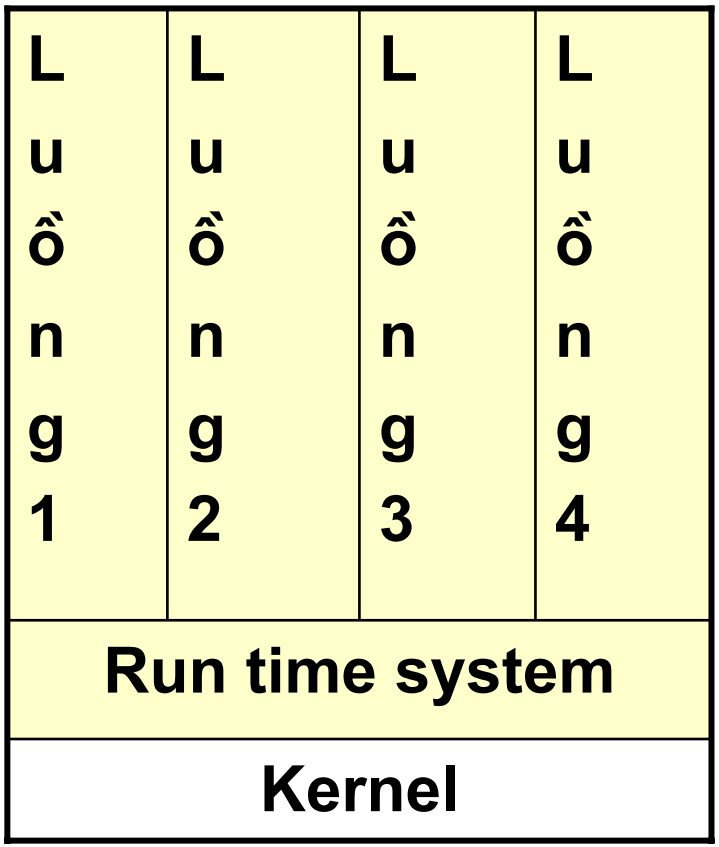
...vấn đề biến tổng thể

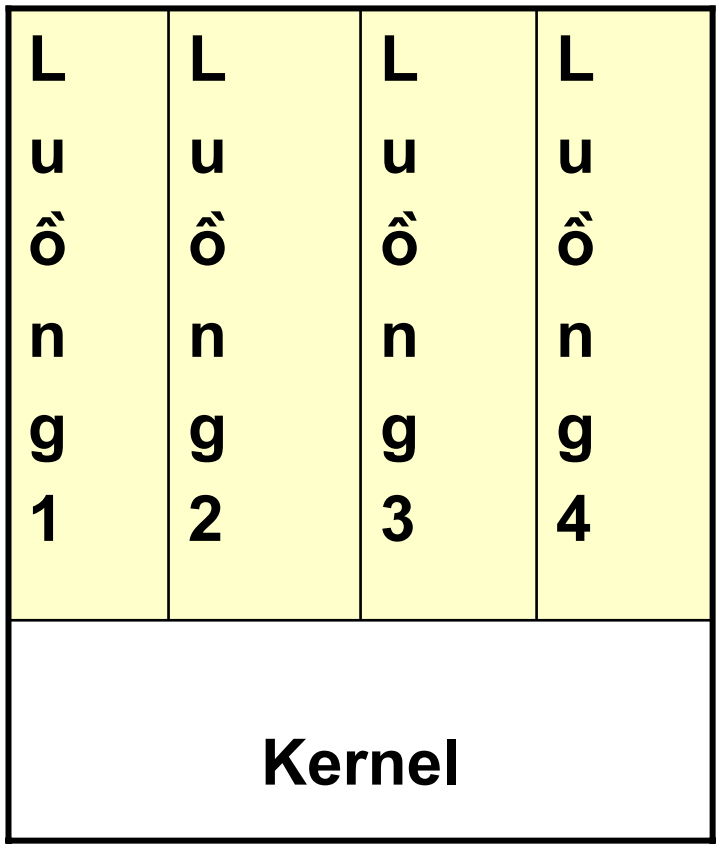
- xây dựng một loại biến tổng thể đặc biệt: tổng thể với mọi thủ tục con nhưng không chung cho mọi luồng
- Thao tác với các biến được thông qua các hàm:
  - create_global(“buf_ptr”)
  - set_global(“buf_ptr”, &buffer)
  - buf_ptr=read_global(“buf_ptr”)

Cài đặt luồng

- Có 2 cách cài đặt: vào nhân hệ điều hành hoặc như một tầng trung gian giữa ứng dụng với HĐH (mức người dùng)
- Cài đặt trong nhân HĐH: Windows NT hỗ trợ đa luồng cho mọi tiến trình
- Cài đặt tại mức người dùng: được cung cấp thành package bổ sung cho mỗi tiến trình sử dụng nó. Unix truyền thống là HĐH đa tiến trình nhưng đơn luồng. Các tiến trình đa luồng đều cài đặt dựa trên một thread package bổ sung

...cài đặt luồng





Gói thư viện luồng

- Cài đặt ở mức người dùng
- Ưu điểm:
  - dùng với mọi HĐH, kể cả loại đơn luồng
  - mỗi tiến trình có thể áp dụng chiến lược lập lịch và cấp phát riêng cho các luồng
  - không phụ thuộc vào không gian rỗi của hệ thống; nhất là khi có nhiều luồng được tạo ra
  - không phải thực hiện lời gọi hệ thống để trao đổi giữa các luồng, do vậy hiệu quả hơn

...gói thư viện luồng

- Nhược điểm:
  - cài đặt system call blocking: dẫn tới block luôn toàn bộ luồng trong tiến trình
  - việc phân chia thời gian CPU giữa các luồng trong 1 tiến trình là non-preemtive
  - Nghịch lý:
    - Nếu thường xảy ra block: khi 1 luồng blocked, các luồng khác phải chờ => không cải thiện được gì
    - Nếu không thường xảy ra block: CPU luôn được sử dụng => cần gì phải tạo ra luồng?

Luồng cài đặt trong kernel

- Ưu điểm:
  - thực sự chia sẻ CPU theo mức độ ưu tiên
  - việc cài đặt blocking dựa trên system call
- Nhược điểm:
  - không hiệu quả do luôn phải thực hiện qua system call
  - các luồng đều chịu chung một lịch phân phối tài nguyên như nhau
- Nhược điểm chung: phải viết lại toàn bộ các hàm thư viện hệ thống để hỗ trợ đa luồng

Mô hình hệ thống

- Mô hình workstation dùng để:
  - swap trang và các files tạm
  - swap trang, files tạm và lưu mã hệ thống
  - swap trang, files tạm, mã hệ thống và caching
  - toàn bộ hệ thống tệp cục bộ

Sử dụng đĩa với các mô hình WS

Sử dụng đĩa	Lợi ích	Bất lợi
Diskless	chi phí thấp, quản trị dễ, mềm dẻo và đối xứng	Dùng nhiều băng thông, có thể gây cổ chai
swap trang, files tạm	giảm bớt tải mạng	chí phí cao hơn
swap trang, files tạm, mã ht	giảm tải mạng	chi phí cAo hơn, quản trị khó hơn
swap trang, files tạm, mã ht, caching	giảm tải mạng, giảm tải file server	chi phí cAO hơn, cần đảm bảo tính nhất quán cache
toàn hệ thống	ít cần giao thông mạng và file server	không đảm bảo tính trong suốt người dùng

Vấn đề sử dụng WS rỗi

- Có 3 vấn đề:
  - phát hiện WS rỗi
  - thực hiện chạy trên máy từ xa 1 cách trong suốt
  - xử lý khi WS trở nên bận (người dùng quay lại)

Xác định WS rỗi

- Khái niệm rỗi:
  - phụ thuộc tài nguyên của mỗi WS
  - thời gian sử dụng CPU
  - tài nguyên đĩa, bộ nhớ không sử dụng
  - thời gian không thao tác của người dùng đối với hệ thống (chạm bàn phím, chuột, kích hoạt tiến trình)

...xác định WS rỗi

- Phương thức 1:
  - các WS rỗi chủ động broadcast tới các WS khác
  - mỗi WS phải lưu giữ một danh sách các WS rỗi và cập nhật thường xuyên
- Phương thức 2:
  - WS client broadcast yêu cầu tới các WS khác
  - WS rỗi thực hiện delay theo mức độ rỗi, rồi trả lời. Do vậy máy rỗi nhất, trả lời trước tiên

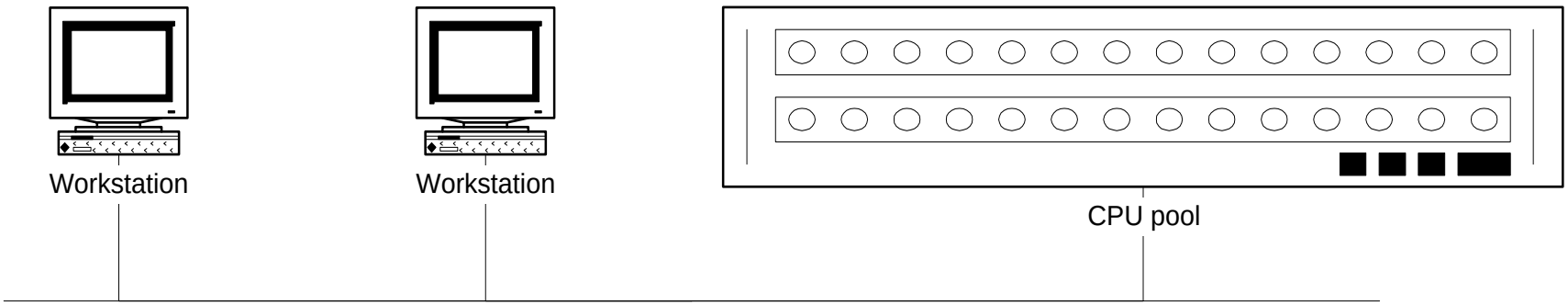
Thực hiện chạy từ xa

- Yêu cầu: môi trường remote WS cần giống như môi trường home WS:
  - hệ thống tệp, biến môi trường, thời gian...
- Cài đặt:
  - sử dụng RPC triệu gọi các thủ tục tại home WS để đọc môi trường gốc

Xử lý tình huống WS trở nên bận

- Cứ tiếp tục chạy cho xong
- Thông báo cho tiến trình thời hạn cần kết thúc, sau đó kill off
  - nhược điểm: thường các tiến trình không được chuẩn bị để xử lý tình huống trên.
- Giải pháp “di cư”: dịch chuyển tiến trình sang một WS rỗi khác. Tương tự như việc dump toàn bộ bộ nhớ tiến trình và chuyển đi

Mô hình processor pool



Đặt các CPU vào một ngăn tập trung, chia sẻ bởi tất cả các WS. Kênh truyền dựa trên cáp truyền.

...processor pool

- Có thể mô hình hoá theo queueing theory:
  - input rate: λ
  - process rate: μ
  - thời gian trung bình trả lời: T

    $T=1/(\mu -\lambda)$
  - Thay hệ thống n queue bởi 1 hệ thống queue tương đương:

    $T=1/(n\mu -n\lambda)=T/n$

Cấp phát bộ xử lý

- Hai loại:
  - non-migratory
  - migratory
- Các yếu tố:
  - cực đại thời gian dùng CPU
  - thời gian trả lời (response time)
  - tỷ suất thời gian trả lời (response ratio): response time 1 /response time 2

Các thuật toán cấp phát

- deterministic vs heuristic
- tập trung vs phân tán
- tối ưu vs xấp xỉ
- cục bộ vs tổng thể
- khởi xướng bởi ws client hoặc ws rỗi

Vấn đề cài đặt thuật toán

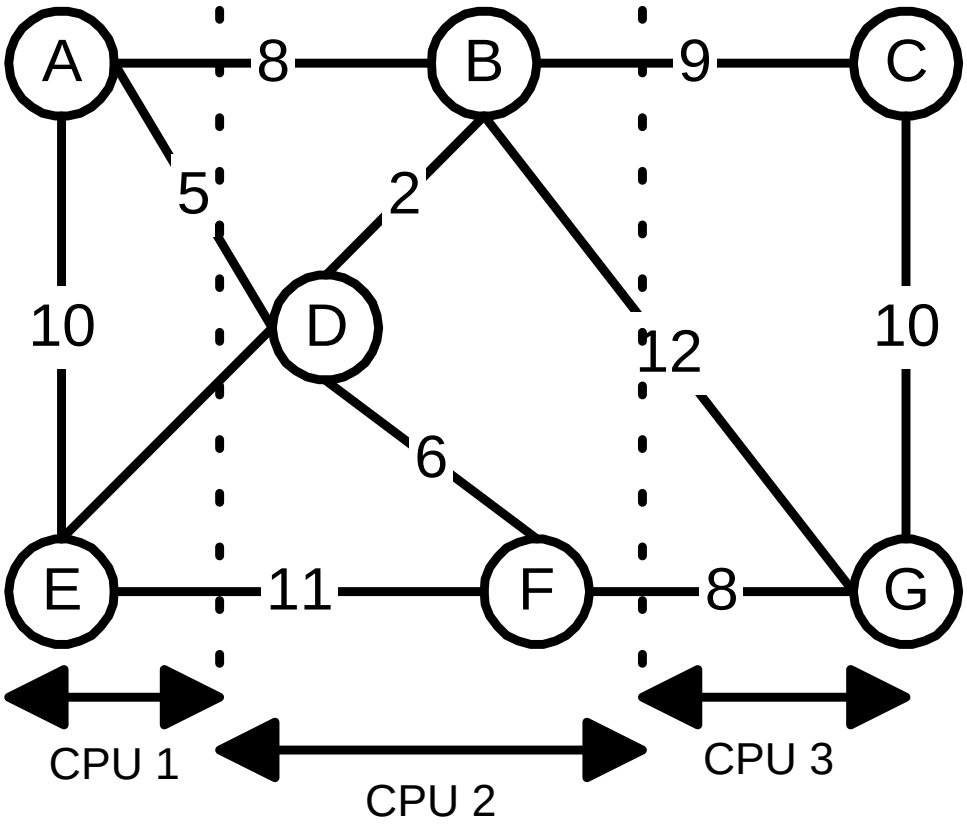
- cần tính chi phí tải mạng, lưu lượng trao đổi giữa các tiến trình, độ phức tạp của thuật toán...
- Ví dụ bởi Eager và đồng nghiệp (1986):
  - Thuật toán 1: chọn ws ngẫu nhiên và gửi tiến trình tới đó. Tiếp diễn như trên...
  - Thuật toán 2: chọn ws ngẫu nhiên và thử, nếu được thì gửi tiến trình. Tiếp như trên...
  - Thuật toán 3: thử toàn bộ các ws và chọn ws rỗi nhất, gửi tiến trình tới đó

  Kết quả: thuật toán 3 hơn chút đỉnh tt 2. Nhưng...

Thuật toán đồ thị

- Là thuật toán deterministic
- Các tiến trình là các đỉnh, cạnh có trọng số là lưu lượng trao đổi giữa các tiến trình
- Tối ưu hoá: giảm thiểu lưu lượng mạng và đảm bảo phân phối CPU cho các tiến trình
- Giải pháp: thuật toán tìm k nhát cắt nhỏ nhất (k là số CPU để cấp phát)

Ví dụ: thuật toán đồ thị



Lưu lượng mạng giữa CPU 1 với bên ngoài là: 8 + 5 + 11

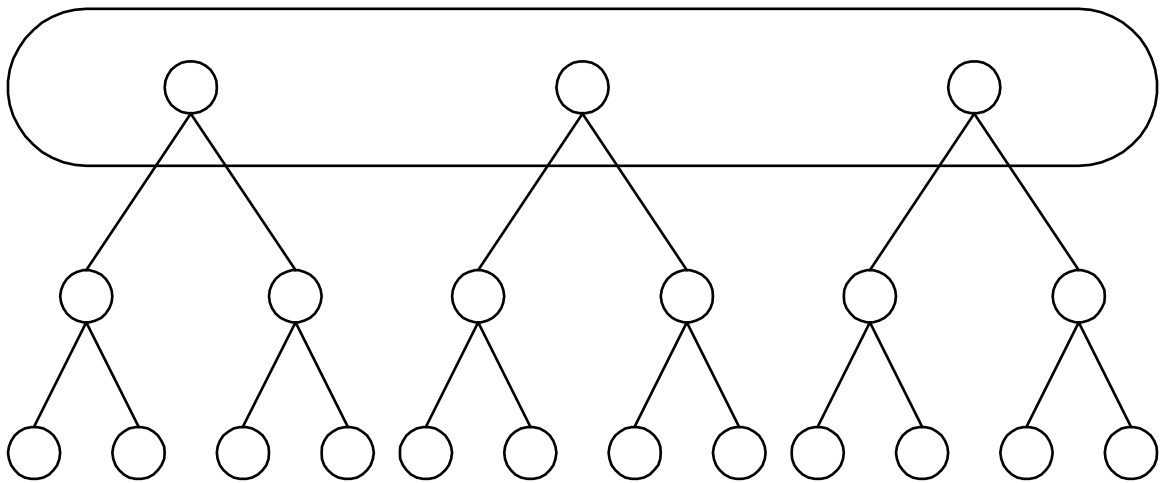
...

Thuật toán heuristic tập trung

- Thuật toán up-down (Mutka-Livny,'87)
- Máy cordinator giữ bảng điểm sử dụng của các ws
- Điểm penalty được cho như sau:
  - điểm được cộng lên khi tiến trình từ xa chạy
  - điểm được trừ đi khi phải chờ để chạy
  - điểm được chia nhỏ giảm dần về không khi không chạy tiến trình từ xa
- Note: thuật toán không cực đại hoá thời gian dùng CPU mà cung cấp giải pháp fair cho các tiến trình

Thuật toán heuristic phân cấp

- Cải tiến tình trạng cổ chai và single-point failure của thuật toán tập trung
- Tổ chức phân cấp các máy, tổ chức bầu cử máy cordinator từng mức vùng



Thuật toán heuristic phân tán

- Eager:
  - chọn ngẫu nhiên một ws
  - hỏi ngưỡng bận của ws đó, nếu chưa vượt quá ngưỡng thì gửi tiến trình tới chạy
  - quá trình lặp lại tới n lần nếu chưa có ws nào dưới ngưỡng bận

Giải pháp đấu thầu

- Đề xuất bởi Ferguson và cộng sự (’88)
- Mô phỏng một cuộc đấu thầu trong ‘nền kinh tế thị trường thu nhỏ’
- Người mua: các tiến trình; tìm nơi thích hợp chạy công việc
- Người bán: các CPU; bán thời gian sử dụng CPU

...giải pháp đấu thầu

- Mới dừng lại ở ý tưởng
- Rất nhiều vấn đề đặt ra, ví dụ:
  - tiến trình lấy tiền ở đâu để mua?
  - Có lương cho tiến trình không?
  - Nếu có thì xếp lương cao hơn nhân viên không?
  - Các CPU có thể thoả hiệp nâng giá không?
  - Các tiến trình có được phép lập công đoàn không?

  :-) ☺

Vấn đề lập lịch trong ht phân tán

- Vấn đề đồng bộ nhịp thời gian CPU được phân giữa các tiến trình cùng nhóm (ví dụ: pipe)
- Giải pháp: lập bảng tiến trình x time slot

CPU 0	CPU 1	CPU 2	CPU 3
x		x	
	x		x
x		x	