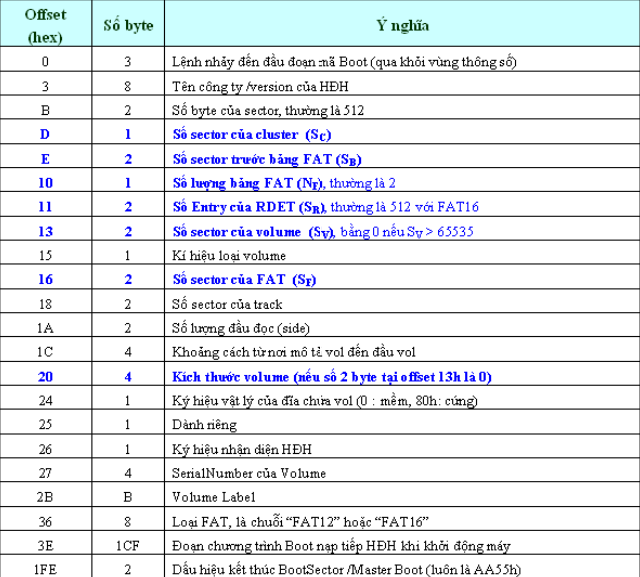
**CHƯƠNG: HỆ THỐNG TẬP TIN & FAT32  
Truy xuất mức vật lý:** Vị trí sector được thể hiện bằng 3 thông số: chỉ số sector, track và head. • Head được đánh số từ trên xuống bắt đầu từ 0. • Track được đánh số theo thứ tự từ ngoài vào bắt đầu từ 0. • Sector được đánh số bắt đầu từ 1 theo chiều ngược với chiều quay của đĩa.  
**Cơ chế đọc đĩa:** **Access time = Seek time + Rotational time + Read time**  
Tổ chức logic: Do truy xuất mức vật lý phải dùng đến 3 tham số rất bất tiện nên tổ chức logic được đưa ra để dễ hiểu, dễ thao tác, dễ tính toán hơn.◼ Cylinder: là tập các track có cùng bán kính (cùng số hiệu) trên tất cả các mặt. → Nhận xét: truy xuất sector theo từng cylinder sẽ đảm bảo *sau khi truy xuất sector K thì truy xuất sector K+1 là nhanh hơn so với tất cả các sector khác.* ◼ Tổ chức logic là một dãy sector được đánh chỉ số theo theo từng cylinder, bắt đầu từ 0. Mỗi lần truy xuất (đọc/ ghi đĩa) chỉ có thể thực hiện trên N sector liên tiếp (N>=1).  
**Sector vật lý <-> Sector logic:**  
• Sector vật lý → Sector logic: **l = t\*st\*hd + h\*st + s -1**  
• Sector logic → Sector vật lý:   
**s = (l mod st) +1; t = l div (st\*hd); h = (l div st) mod hd**  
Trong đó: l: chỉ số sector logic, h: chỉ số head, t: chỉ số track, s: chỉ số sector vật lý, st: số sector/track, th: số track/side (head), hd: tổng số side (head)  
**Thuộc tính tập tin:** trên các hệ thống tập tin khác nhau sẽ khác nhau, thường gồm các thuộc tính sau: • *Tên (tên + phần mở rộng)* • *Người sở hữu* • *Thuộc tính trạng thái: chỉ đọc, ẩn,…* • *Kích thước* • *Ngày giờ (tạo, truy cập, thay đổi)* • *Thuộc tính bảo vệ* • *Vị trí lưu trữ trên đĩa***Tổ chức tập tin - Cấp phát liên tục:** Mỗi tập tin chiếm các block liên tục trên đĩa. • Đơn giản, chỉ cần quản lý vị trí (chỉ số) block bắt đầu và chiều dài (số block). • Hỗ trợ truy xuất tuần tự & truy xuất trực tiếp. • Vấn đề External fragmentation. • Vấn đề khi kích thước tập tin tăng.  
- Hệ thống tập tin cấp phát theo *extent*: • Extent là một tập các block liên tục. • Cấp phát cho tập tin theo từng extent. • Một tập tin có thể chiếm một hoặc nhiều extent không liên tục nhau. • Kích thước các extent có thể khác nhau. • Cần quản lý 3 thông tin: vị trí block bắt đầu, số block và một con trỏ trỏ tới block đầu tiên của extent kế tiếp. • Vấn đề Internal fragmentation và External fragmentation.  
**Tổ chức tập tin - Cấp phát theo DSLK:** • Mỗi tập tin chiếm một tập các block theo kiểu danh sách liên kết. • Mỗi block sẽ chứa thông tin về địa chỉ của block kế tiếp. • Các block có thể nằm rãi rác trên đĩa. • Chỉ hỗ trợ truy xuất tuần tự. • Đơn giản, chỉ cần quản lý vị trí (chỉ số) block bắt đầu. • Không bị External fragmentation. • Tốn chi phí lưu địa chỉ block kế tiếp.  
**Tổ chức tập tin - Cấp phát theo chỉ mục:** • Gồm một hoặc nhiều block làm bảng chỉ mục chứa địa chỉ của các block dữ liệu. • Hỗ trợ truy xuất tuần tự & truy xuất trực tiếp. • Tốn không gian đĩa để lưu các block chỉ mục. • Không bị External fragmentation. • Một số mô hình mở rộng: Mô hình chỉ mục nhiều cấp, Mô hình chỉ mục kết hợp danh sách liên kết, Mô hình chỉ mục nhiều cấp kết hợp danh sách liên kết.  
**Quá trình khởi động hệ thống từ đĩa từ: 1.** POST (Power-On Self-Test) **2.** Tải MBR để đọc thông tin bảng phân vùng. Tìm phân vùng “active”. Nếu không tìm thấy phân vùng “active”, MBR có thể tải một boot loader và chuyển điều khiển cho nó. Boot loader này sẽ cho phép chọn HĐH trên một phân vùng. **3.** Chuyển quyền điều khiển về cho đoạn mã chương trình nằm trong Boot Sector của phân vùng được chọn. **4.** Tải HĐH tại phân vùng được chọn.  
**Tổ chức hệ thống tập tin trong Main Mem:** Các thông tin cần lưu trữ trong bộ nhớ:   
• Mounted Volume Table – Danh sách các volume được sử dụng trên hệ thống. • Directory Structure – Thông tin các thư mục mới được sử dụng: ( Con trỏ trỏ tới volume tương ứng). • System-wide open-file Table – Danh sách các tập tin đang được mở trên hệ thống: (Con trỏ tập tin, định vị tập tin trên đĩa. Quyền truy cập. Biến đếm tập tin đang mở.) • Per-process open-file Table – Danh sách các tập tin mà tiến trình đang thao tác: Con trỏ trỏ tới tập tin đang mở tương ứng trong system-wide open-file table.   
**Kết buộc (Mount) hệ thống tập tin:** • Một hệ thống tập tin phải được kết buộc (mount) trước khi có thể truy xuất. (giống như tập tin phải được mở trước khi sử dụng). • Các HĐH thường phát hiện và tự động kết buộc các hệ thống tập tin tồn tại trên hệ thống: (•Windows kết buộc hệ thống tập tin vào ổ đĩa. • Linux kết buộc hệ thống tập tin vào một thư mục.) • Một số HĐH cung cấp lệnh để thực hiện việc kết buộc hệ thống tập tin. (• Ví dụ: lệnh *mount* (Linux)).  
**FAT – Vùng Boot Sector:** • Gồm 1 số sector đầu tiên của phân vùng (partition), trong đó: • Sector đầu tiên (Boot Sector): (• Chứa các thông số quan trọng của phân vùng. • Chứa đoạn chương trình nhỏ để nạp HĐH khi khởi động máy.) • Các sector còn lại (nếu có): (• Chứa các thông tin hỗ trợ cho việc xác định tổng số cluster trống & tìm kiếm cluster trống được hiệu quả. • Chứa 1 sector bản sao của Boot sector.)  
**FAT 12, 16:**  
  
**FAT 32:**  
 **Bảng thư mục gốc – REDT:**   
A table with text and numbers

Description automatically generated with medium confidenceA table with text and numbers

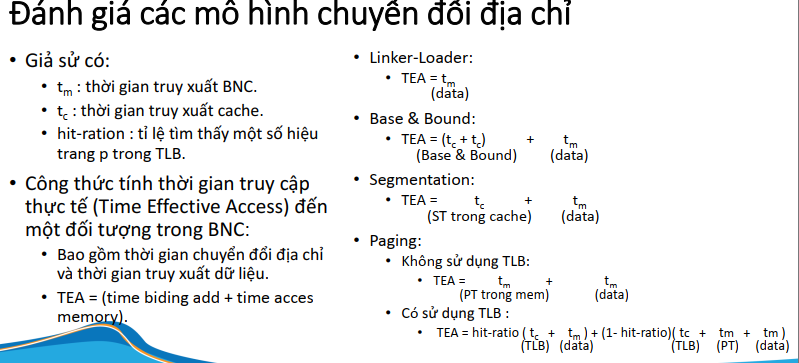
Description automatically generated **Công thức tương quan giữa cluster thứ k trên vùng dữ liệu và sector thứ i trên phân vùng: i = SB + SF\*NF + [SRDET] + (k – 2)\*Sc****SEDT – Bảng thư mục con:** • Chứa thông tin các tập tin/ thư mục con của một thư mục. • Nằm trên vùng dữ liệu, có cấu trúc hoàn toàn giống bảng thư mục gốc. • Mỗi SDET luôn có 2 entry ‘.‘ và ‘..‘ ở đầu bảng mô tả về chính thư mục này và thư mục cha của nó.  
**Thao tác trên HTTT FAT:  
• Đọc nội dung tập tin (TYPE):** • Xác định entry chính trong bảng thư mục (RDET/ SDET) chứa thông tin của tập tin dựa vào phần tên và phần mở rộng (lưu ý trường hợp tên dài). • Từ entry chính tìm được, ta có được chỉ số cluster/ phần tử FAT đầu tiên. • Từ phần tử FAT đầu tiên này, vào bảng FAT, xác định các phần tử còn lại của tập tin, tương ứng có được các cluster của tập tin này → các sector của tập tin. • Đọc các sector nội dung của tập tin.  
**• Liệt kê nội dung thư mục (DIR):** • Xác định entry chính trong bảng thư mục (RDET/ SDET) chứa thông tin của thư mục dựa vào phần tên (lưu ý trường hợp tên dài). • Từ entry chính tìm được, ta có được chỉ số cluster/ phần tử FAT đầu tiên. • Từ phần tử FAT đầu tiên này, vào bảng FAT, xác định các phần tử còn lại của tập tin, tương ứng có được các cluster của tập tin này → các sector của tập tin. • Đọc các sector nội dung tìm được theo từng entry (32 bytes) và hiển thị thông tin của các tập tin và thư mục con của thư mục này.  
**• Tạo tập tin (COPY CON):** • Tìm đủ số entry trống liên tiếp nhau trên bảng thư mục (RDET/ SDET) để chứa thông tin của tập tin (lưu ý trường hợp tên dài). • Kiểm tra trên bảng FAT xem còn đủ số cluster trống để chứa nội dung của tập tin không. • Lưu thông tin của tập tin vào các entry trống tìm được. • Ghi giá trị vào các phần tử FAT trống tìm được theo dạng danh sách liên kết, đồng thời lưu nội dung tập tin vào các cluster tương ứng (theo chỉ số sector).  
**• Tạo thư mục (MD):** • Tìm đủ số entry trống liên tiếp nhau trên bảng thư mục (RDET/ SDET) để chứa thông tin của thư mục (lưu ý trường hợp tên dài). • Kiểm tra trên bảng FAT xem còn cluster trống nào để chứa nội dung của thư mục không. • Lưu thông tin của thư mục vào các entry trống tìm được. • Ghi giá trị kết thúc vào phần tử FAT trống tìm được, đồng thời tạo 2 thư mục “.” Và “..” chiếm 2 entry đầu tiên trong cluster tương ứng.  
**• Xóa tập tin (DELETE):** • Xác định entry chính trong bảng thư mục (RDET/ SDET) chứa thông tin của tập tin dựa vào phần tên và phần mở rộng (lưu ý trường hợp tên dài). • Đặt giá trị E5h vào byte đầu tiên của entry chính và tất cả các entry phụ của tập tin (nếu có). • Từ entry chính tìm được, ta có được chỉ số cluster/ phần tử FAT đầu tiên. Vào bảng FAT, xác định được các phần tử còn lại của tập tin. • Đặt tất cả các phần tử FAT của tập tin về giá trị 0. • Lưu ý, hoàn toàn không thay đổi gì phần nội dung của tập tin.  
**• Xóa thư mục (RD):** • Thực hiện xóa đệ qui tất cả các tập tin và thư mục con từ cấp sâu nhất ra. Xóa thư mục rỗng tương tự như xóa tập tin.  
• **Sao chép tập tin (COPY):** • Tìm đủ số entry trống liên tiếp nhau trên bảng thư mục (RDET/ SDET) để chứa thông tin của tập tin đích (lưu ý trường hợp tên dài). • Kiểm tra trên bảng FAT xem còn đủ số cluster trống để chứa nội dung của tập tin đích không. • Copy thông tin (các entry) của tập tin nguồn sang các entry tìm được của tập tin đích. • Ghi giá trị vào các phần tử FAT trống tìm được theo dạng danh sách liên kết, đồng thời copy các sector nội dung tập tin nguồn vào các sector nội dung tương ứng tìm được của tập tin đích.  
**• Di chuyển tập tin (MOVE):** • Tìm đủ số entry trống liên tiếp nhau trên bảng thư mục (RDET/ SDET) để chứa thông tin của tập tin đích (lưu ý trường hợp tên dài). • Copy thông tin (các entry) của tập tin nguồn sang các entry tìm được của tập tin đích • Xóa thông tin của tập tin nguồn.  
**• Đổi tên tập tin/ thư mục (REN):** • Xác định entry chính trong bảng thư mục (RDET/ SDET) chứa thông tin của tập tin/ thư mục dựa vào phần tên và phần mở rộng (lưu ý trường hợp tên dài). • Nếu tên tập tin không cần thêm các entry phụ: (Cập nhật lại phần tên và phần mở rộng.) • Nếu tên tập tin cần thêm các entry phụ: (Tìm đủ số entry trống liên tiếp nhau trên bảng thư mục (RDET/ SDET) để chứa thông tin của tập tin đích (lưu ý trường hợp tên dài)). • Copy thông tin (các entry) của tập tin nguồn sang các entry tìm được của tập tin đích.  
**• Quick format:** • Giữ lại các thông số cũ của phân vùng. • Cập nhật lại trạng thái các cluster đang chứa dữ liệu thành trống và cho tất cả entry trên bảng thư mục gốc về trạng thái trống. • Chức năng này tương đương với việc xóa tất cả mọi tập tin & thư mục đang tồn tại trên phân vùng, nhưng thời gian thi hành rất nhanh, có thể nhanh hơn thời gian xóa một tập tin.  
**• Full format:** • Các thông số của từng thành phần trên phân vùng sẽ được xác định lại. • Để tạo ra những dạng thức mới phù hợp hơn cho phân vùng. Chức năng này dĩ nhiên cũng được dùng cho những phân vùng chưa được định dạng.

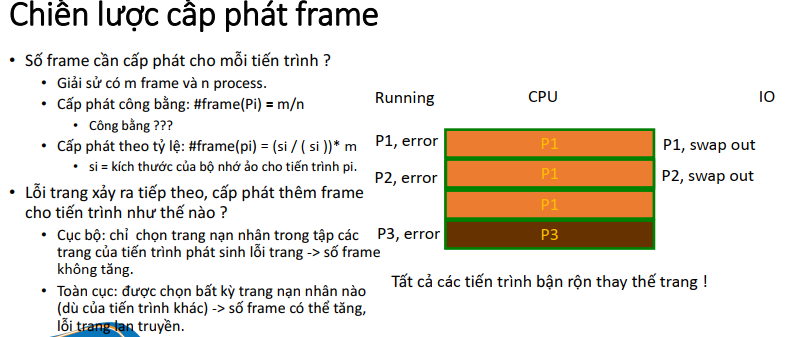
**CHƯƠNG: ĐIỀU PHỐI TIẾN TRÌNH**- CPU burst: là giai đoạn đang sở hữu CPU & đc thực thi (các phép tính toán…)  
- I/O burst: là giai đoạn chờ dl nhập/xuất.  
🡪 Việc thực thi 1 process là 1 sự luân phiên giữa CPU & I/O burst.  
**- CPU Scheduler:** Chọn các process đang trong ready queue để thực thi và giao CPU cho nó. 3 loại: long-term sheduler, short-term scheduler: lựa chọn process để giao cho CPU, medium-term scheduler. Sẽ diễn ra khi 1 process:  
+ Từ running sang waiting state. (1)  
+ Từ running sang exit. (2)  
+ Từ running sang ready state.  
+ Từ waiting sang ready state.  
(1) và (2): nonpreemptive còn lại là preemptive.  
**- Dispatcher:** giao CPU cho tiến tình đc chọn bởi short-term scheduler. (switching context, switching to user mode, nhảy vào vị trí cụ thể trong user program để khởi động ctr đó.)  
**- Scheduling criteria:**   
+ throughput: slg process h.thành trên đ.vị thời gian.  
+ turnaround time: khoảng tgian khi thực hiện 1 process. Từ khi ctrinh bắt đầu vào hệ thống đến khi kết thúc.  
+ waiting time: là lg tgian nằm trong ready queue.  
+ reponse time: là khoảng tgian khi process gửi y.cầu đến khi CPU respond lại.  
RMS (rate monotonic scheduling): tk nào có chu kỳ ngắn thì ưu tiên sớm nhất (vì dễ gặp deadline sớm nhất). EDF (Earliest Deadline First): ưu tiên tk nào có deadine sớm nhất chạy trước **NXét FCFS:** • Đơn giản, dễ cài đặt. • Chịu đựng hiện tượng tích lũy thời gian chờ. • Tiến trình có thời gian xử lý ngắn đợi tiến trình có thời gian xử lý dài. • Ưu tiên tiến trình CPU-bounded. • Có thể xảy ra tình trạng độc chiếm CPU • Không phù hợp với hệ thống tương tác người dung.  
**NXét SJF**: • Tối ưu thời gian chờ • Làm sao biết được thời gian còn lại cần thực thi của một tiến trình: **τ *n*+1 = α *tn* + (1-α )τ*n***  
• **Tại sao không dùng nhiều tiến trình để thay thế cho việc dùng nhiều tiểu trình?** • Các tác vụ điều hành tiểu trình (tạo, kết thúc, điều phối, chuyển đổi,…) ít tốn chi phí thực hiện hơn so với tiến trình. • Liên lạc giữa các tiểu trình thông qua chia sẻ bộ nhớ, không cần sự can thiệp của kernel.  
**Liên lạc giữa các tiến trình – Message Passing**: • Message: (• Dữ liệu có cấu trúc. • Cấu trúc và thông dịch msg được thỏa thuận giữa 2 tiến trình liên lạc.) • HĐH cung cấp 2 primitive chính: (• send(destination, message).• receive(source, message)) • Các vấn đề quan tâm: (• Direct or indirect addressing. • Blocking or non-blocking communication. • Reliable or unreliable communication. • Buffered or un-buffered communication.) • Là cơ chế IPC tổng quát: (• Hỗ trợ liên lạc giữa các tiến trính trên cùng máy. • Hỗ trợ liện lạc giữa các tiến trính trong hệ thống phân tán.) • Liên lạc giữa các hệ thống không đồng nhất ?

**CHƯƠNG: ĐỒNG BỘ HÓA  
Race condition (tranh đoạt điều khiển):**   
- Khi nhiều process có d.liệu dùng chung & truy cập đồng thời 🡪 Data ko nhất quán & vô lý. Là tình huống mà kết quả của ctr phụ thuộc vào sự điều phối của hệ thống. (1 input nhưng cho ra nhiều output).  
**Giải pháp đúng phải thỏa:** - Mutal exclusion: chỉ 1 process đc thực thi trong CS tại 1 thời điểm. - Ko có giả thiết về tốc độ CPU, slg CPU. - Ko có process ở ngoài CS lại khóa cho process ≠ vào. - Starvation: ko có process nào đợi mãi mãi để vào CS.  
**I. Giải pháp Busy Waiting: (nhc điểm chug: lãng phí CPU do tiêu thụ CPU trong khi chờ vào CS). Khắc phục: Block các process ch đủ đk vào CS, nhường CPU cho process ≠. Phải nhờ đến scheduler  
a. G.pháp ph.mềm: Ko đòi hỏi trợ giúp của h.thống, dễ sai, khó mở rộng**1. Lock variable: có thể mở rộng cho N process. Nhưng 2 process có thể vào miền găng tại 1 th.điểm (vì biến lock cx là 1 biến dùng chung).  
2. Strict Alternation: chỉ dành cho 2 tiến trình. Giải quyết đc mutal exclusion. Nhưng vi phạm 1 process ngoài CS không cho process ≠ vào CS (vì thời gian cần thiết thực thi của các anh là ≠ nhau).  
3. Giải pháp Peterson: Giải pháp đáp ứng đc cả 4 đk trên. Nhược (priority inversion: xảy ra khi 1 process có độ ưu tiên cao hơn bị block bởi 1 process có độ ưu tiên thấp hơn), Không thể mở rộng cho N process.  
**b. Giải pháp phần cứng: Cần sự hỗ trợ của cơ chế phần cứng: ko dễ, khi các máy có nhiều bộ xử lý. Dễ mở rộng cho N process.**1. Vô hiệu hóa ngắt: (2 loại ngắt phổ biến)  
+ Timer interrupts: q.lý việc đ.phối CPU  
+ I/O device Interrupts: báo hiệu I/O đã xong (báo cho CPU biết để chuyển process từ state blocled sang ready.) PP này cho phép 1 process khi vào CS sẽ vô hiệu hóa các ngắt vì: (+ Khi 1 timer ngắt bật lên thì scheduler ms chạy 🡪 dẫn đến context switch) 🡪 ko ngắt 🡪 ko CPU scheduling 🡪 ko CS probs.  
+ Nhược: nếu 1 process bị block mà chưa bật lại ngắt 🡪 HDH die (system Halt). + ko có tác dụng vs hệ thống có nhiều CPU vì ko đảm bảo Mutal-exclusion.  
2. TSL (Test\_And\_Set)  
- Là 1 chỉ thị lệnh của CPU (≠ lock var là do lập trình viên).  
- Chỉ thị lệnh của CPU (atomic operation): nếu khối lệnh đc bỏ vào chỉ thị lệnh thì sẽ ko bị lấy lại CPU trong quá trình nó thực hiện khối lệnh này.  
- Chỉ có 1 process dc gọi chỉ thị test\_and\_set\_lock tại 1 th.đ.  
**II. Giải pháp Sleep & Wakeup:**• Từ bỏ CPU khi ch đc vào CS. • Khi CS trống, sẽ được đánh thức để vào CS. • Cần được HĐH hỗ trợ. • Vì phải thay đổi trạng thái tiến trình.• HDH hỗ trợ 2 primitive: • Sleep(): process gọi sẽ nhận trạng thái Blocked. • WakeUp(P): process P nhận trạng thái Ready. • Áp dụng: • Sau khi kiểm tra đkiện sẽ vào CS hay gọi Sleep() tùy vào kquả ktra. • process vừa sử dụng xong CS sẽ đánh thức các tiến trình bị Blocked trước đó.  
• Nguyên nhân: • Việc kiểm tra điều kiện và động tác từ bỏ CPU có thể bị ngắt quãng. • Bản thân các biến cờ hiệu không được bảo vệ.  
**1. Semaphore:** Là 1 biến có các thuộc tính: 1 gtri nguyên dg e(s), 1 hàng đợi f(s) lưu ds các process đang bị khóa. Cho phép ktra có đc vào CS hay ko thông qua 2 atomic operations down & up.  
+ down: giảm gtri của semaphore s đi 1 đvi nếu e(s)>0, & tiếp tục xử lý. Nếu e(s)=0, process phải chờ đến khi e(s)>0. (Trc khi vào CS, process sẽ gọi down để ktra có đc vào CS hay ko. Nếu biến này thỏa đk thì anh gọi hàm down đc vào, ko thì bị sleep).  
+ up:Khi ra khỏi CS, process sẽ gọi up để báo hiệu ra khỏi CS 🡪 tăng biến semaphore, đồg thời wakeup 1 anh nào đó đang bị ngủ khi đã gọi down để tiếp tục thực thi.  
Nxét: • Là một cơ chế tốt để thực hiện đồng bộ: Dễ dùng cho N tiến trình. • Nhưng ý nghĩa sử dụng không rõ ràng: (MutualExclusion: Down & Up; Rendez-vous: Down & Up. Chỉ phân biệt qua mô hình.) • Khó sử dụng đúng: Nhầm lẫn.  
**2. GPháp Mutex:** • Khi khả năng đếm của semaphore ko cần thiết, 1 phiên bản đơn giản của semaphore - mutex • Mutex: biến có thể ở 1 trong 2 trạng thái bị khóa (locked – 0), ko khóa (unlocked – 1 hoặc giá trị khác): Dễ sử dụng. • Phù hợp để sử dụng với tiểu trình ở ko gian người dùng: Tiểu trình (hoặc tiến trình) muốn truy cập CS, gọi mutex\_lock. Nếu mutex ở trạng thái được mở khóa (unlocked), lời gọi mutex\_lock sẽ thành công. Nếu không, tiểu trình sẽ bị khóa cho đến khi mutex\_unlock được gọi trong tiểu trình khác.  
**3. GPháp Monitor:** • Là cơ chế đồng bộ hoá do NNLT cung cấp: Hỗ trợ cùng các chức năng như Semaphore. Dễ sử dụng và kiểm soát hơn Semaphore: (Bảo đảm Mutual Exclusion một cách tự động. Sử dụng biến điều kiện để thực hiện Synchronization) • Là một module chương trình định nghĩa: Các CTDL, đối tượng dùng chung. Các phương thức xử lý các đối tượng này. Bảo đảm tính encapsulation. • Các tiến trình muốn truy xuất dữ liệu bên trong monitor phải dùng các phương thức của monitor: • P1 : M.C() // i=5 • P2 : M.B() // printf(j). • Tự động bảo đảm Mutual Exclusion: Tại 1 thời điểm chỉ có 1 tiến trình được thực hiện các phương thức của Monitor. Các tiến trình không thể vào Monitor sẽ được đưa vào Entry queue của Monitor. • Hỗ trợ Synchronization với các condition variables: Wait(c) : Tiến trình gọi hàm sẽ bị blocked. Signal(c): Giải phóng 1 tiến trình đang bị blocked trên biến điều kiện c. C.queue : danh sách các tiến trình blocked trên c. • Trạng thái tiến trình sau khi gọi Signal? Blocked. Nhường quyền vào monitor cho tiến trình được đánh thức. Tiếp tục xử lý hết chu kỳ, rồi blocked.  
**4. GPháp Message:** • Được hỗ trợ bởi HĐH. • Đồng bộ hóa trên môi trường phân tán. • 2 primitive Send & Receive. • Cài đặt theo mode blocking.

**CHƯƠNG: QUẢN LÝ BỘ NHỚ & BỘ NHỚ ẢO  
Chuyển đổi địa chỉ (address binding):**• Compile&Link-time: Phát sinh địa chỉ vật lý. Phải biết trước vị trí nạp chương trình. Phải biên dịch lại chương trình khi vị trí nạp thay đổi.  
• Load-time: Khi biên dịch chỉ phát sinh địa chỉ cục bộ của chương trình. Khi nạp, biết vị trí bắt đầu sẽ tính lại địa chỉ vật lý. Phải tái nạp khi vị trí bắt đầu thay đổi.  
• Execution-time: Khi biên dịch, nạp chỉ phát sinh địa chỉ cục bộ của chương trình. Trì hoãn thời điểm kêt buộc địa chỉ tuyệt đối đến khi thi hành. Khi đó, ai tính toán địa chỉ vật lý? Phần cứng: MMU (trong CPU chip). **Tổ chức bộ nhớ - cấp phát liên tục:** • Nguyên tắc: Chương trình được nạp toàn thể vào BNC để thi hành. Cần một vùng nhớ liên tục, đủ lớn để chứa Chương trình. • Không gian địa chỉ: liên tục. • Không gian vật lý: có thể tổ chức (Fixed partition. Variable partition.) • Mô hình chuyển đổi địa chỉ: (Linker – Loader. Base & Bound) **Fixed Partitioning:** • Phân chia KGVL thành các partitions. • Có 2 cách phân chia partitions: (Kích thước bằng nhau. Kích thước khác nhau.) • Mỗi tiến trình sẽ được nạp vào một partition để thi hành.  
Chiến lược cấp phát partition ko = nhau:  
• Sử dụng nhiều hàng đợi: • Cấp cho tiến trình partition với kích thước bé nhất (đủ lớn để chứa tiên trình). • Khuyết điểm : phân bố các tiến trình vào các partition không đều, một số tiến trình phải đợi trong khi có partition khác trống.  
• Sử dụng 1 hàng đợi: • Cấp cho tiến trình partition tự do với kích thước bé nhất (đủ lớn để chứa tiên trình). • Cần dùng một CTDL để theo dõi các partition tự do.  
Nxét: • Sử dụng BN không hiệu quả. • Internal fragmentation: kích thước

chương trình không đúng bằng kích thước partition. • Mức độ đa chương của hệ thống (Số tiến trình được nạp) bị giới hạn bởi số partitions.  
**Dynamic Partitioning:** • BNC không được phân chia trước: Các partition có kích thước tùy ý, sẽ hình thành trong quá trình nạp các tiến trình vào hệ thống. • Mỗi tiến trình sẽ được cấp phát đúng theo kích thước yêu cầu: ko còn internal fragmentation. • Tiến trình vào sau không lấp đầy chỗ trống tiến trình trước để lại: External fragmentation.  
Thuật toán lựa chọn vùng trống: • Chọn lựa partition để cấp phát cho tiến trình? Đồng thời có nhiều partition tự do đủ lớn để chứa tiến trình. Dynamic Allocation problem. • Các chiến lược thông dụng để chọn partition: First-fit: chọn partition tự do đầu tiên. Best-fit: chọn partition tự do nhỏ nhất đủ chứa tiến trình. Worst-fit: chọn partition tự do lớn nhất đủ chứa tiến trình. • Giải quyết vấn đề External Fragmentation: Dồn các vùng bị phân mảnh lại với nhau để tạo thành partition liên tục đủ lớn để sử dụng. Chi phí thực hiện cao.  
**Chuyển đổi địa chỉ - Link-Loader:** • Tại thời điểm Link, giữ lại các địa chỉ logic. • Vị trí base của tiến trình trong bộ nhớ xác định được vào thời điểm nạp: **địa chỉ physic = địa chỉ logic + base** • Nhận xét: Ko cần sự hỗ trợ phần cứng để chuyển đổi địa chỉ (Loader thực hiện). • Bảo vệ: ko hỗ trợ. Dời chuyển sau khi nạp: Ko hỗ trợ tái định vị, Phải nạp lại.  
**Chuyển đổi địa chỉ - Base & Bound:** • Tại thời điểm Link, giữ lại các địa chỉ logic. • Vị trí base, bound được ghi nhận vào 2 thanh ghi: (Kết buộc địa chỉ vào thời điểm thi hành. => Tái định vị được: **địa chỉ physic = địa chỉ logic + base register** => Cần sự hỗ trợ của phần cứng.) • Bảo vệ: **Địa chỉ hợp lệ [base, bound]  
Cấp phát ko liên tục:** • Cho phép nạp tiến trình vào BNC ở nhiều vùng nhớ không liên tục. • Không gian địa chỉ: phân chia thành nhiều partition: (Phân đoạn (Segmentation). Phân trang (Paging)) • Không gian vật lý: có thể tổ chức: (Fixed partition : Paging. Variable partition : Segmentation)  
**Segmentation – Tổ chức:** • Địa chỉ logic : <segment-number, offset> • Địa chỉ physic : <real address> • Chuyển đổi địa chỉ : <s,d> <r> • Chuyển đổi địa chỉ vào thời điểm thi hành: (MMU thi hành. Sử dụng Segment Table (bảng phân đoạn) để lưu thông tin cấp phát BNC, làm cơ sở thực hiện ánh xạ địa chỉ. Mỗi tiến trình có một Segment Table) • Segment Table:   
• Số phần tử của Segment Table = Số Segment của chương trình  
• Mỗi phần tử của Segment Table mô tả cho 1 segment, và có cấu trúc: base: địa chỉ vật lý trong BNC của partition chứa segment. limit : kích thước segment.  
• Lưu trữ Segment Table? Cache : nếu đủ nhỏ. BNC : Segment-table base register (STBR), Segment-table length register (STLR).  
**Segmentation – Nhận xét:** • Cấp phát không liên tục => tận dụng bộ nhớ hiệu quả. • Hỗ trợ tái định vị: Từng Segment. • Hỗ trợ Bảo vệ và Chia sẻ được ở mức module. • Chuyển đổi địa chỉ phức tạp: Đã có MMU • Sử dụng dynamic partition: chịu đựng: (• Dynamic Allocation : chọn vùng nhớ để cấp cho một segment: First fit, Best fit, Worst fit. • External Fragmentation: Memory Compaction: chi phí cao.)  
**Cấp phát không liên tục – Paging:** • Hỗ trợ HĐH khắc phục bài toán cấp phát bộ nhớ động, và loại bỏ external fragmentation. • KGĐC : phân chia chương trình thành các page có kích thước = nhau: (Không quan tâm đến ngữ nghĩa của các đối tượng nằm trong page.) • KGVL : tổ chức thành các fixed partitions có kích thước bằng nhau gọi là frame. • Nạp tiến trình: • Mỗi page cần được nạp vào một frame tự do. • Các pages của cùng 1 chương trình có thể được nạp vào những frames không kế cận nhau. • Tiến trình kích thước N pages -> cần N frame tự do để nạp.  
**Paging – tổ chức:** • Điạ chỉ logic : <page-number, offset> • Địa chỉ physic : <frame-number, offset> • Chuyển đổi địa chỉ : <p,d> <f,d> • Chuyển đổi địa chỉ vào thời điểm thi hành: (MMU thi hành. Sử dụng Page Table để lưu thông tin cấp phát BNC, làm cơ sở thực hiện ánh xạ địa chỉ. Mỗi tiến trình có một Page Table.) • Page Table: (Số phần tử của Page Table = Số Page trong KGĐC của chương trình. Mỗi phần tử của bảng Page Table mô tả cho 1 page, và có cấu trúc: frame: số hiệu frame trong BNC chứa page.• Lưu trữ Page Table? Cache : không đủ. BNC : Page-table base register (PTBR), Page-table length register (PTLR))  
**Paging – Nhận xét:** • Loại bỏ: (Dynamic Allocation. External Fragmentation) • “Trong suốt” với LTV. • Hỗ trợ Bảo vệ và Chia sẻ ở mức page. • Internal Fragmentation. • Lưu trữ Page Table trong bộ nhớ: (Tốn chỗ. Những page không sử dụng: lãng phí. Tăng thời gian chuyển đổi địa chỉ.)   
**Page Table:** • Mỗi tiến trình lưu 1 Page Table. • Số lượng phần tử quá lớn -> Lưu BNC. • Mỗi truy xuất địa chỉ sẽ tốn 2 lần truy xuất BNC.

**Paging – vấn đề truy xuất bộ nhớ:** • Mỗi truy cập BNC cần truy xuất BNC 2 lần: (Tra cứu Page Table để chuyển đổi địa chỉ, Tra cưu bản thân data) • Làm gì để cải thiện: (Tìm cách lưu PT trong cache, Cho phép tìm kiếm nhanh.)   
**Giải pháp Translation Lookaside Buffer (TLB):** • Vùng nhớ Cache trong CPU được sử dụng để lưu tạm thời một phần của PT được gọi là Translation Lookaside Buffer (TLB): (Cho phép tìm kiếm tốc độ cao. Kích thước giới hạn (thường không quá 64 ptử). • Mỗi entry trong TLB chứa một số hiệu page và frame tương ứng đang chứa page. • Khi chuyển đổi địa chỉ, truy xuất TLB trước, nếu không tìm thấy số hiệu page cần thiết, mới truy xuất vào PT để lấy thông tin frame.  
**Bảo vệ và chia sẻ trong Segmentation và Paging:**   
• Bảo vệ: • Segmentation: mỗi phần tử trong ST được gắn thêm các bit bảo vệ. • Mỗi segment có thể được bảo vệ tùy theo ngữ nghĩa của các đối tượng bên trong segment. • Paging: mỗi phần tử trong PT được gắn thêm các bit bảo vệ. • Mỗi page không nhận thức được ngữ nghĩa của các đối tượng bên trong page, nên bảo vệ chỉ áp dụng cho toàn bộ trang, không phân biệt.  
• Chia sẻ: Cho nhiều phần tử trong KGĐC cùng trỏ đến 1 vị trí trong KGVL. • Segmentation : chia sẻ mức module chương trình. • Paging : chia sẻ các trang.  
****A diagram of a computer

Description automatically generated **Thay thế trang – FIFO – Nhận xét:** • Đơn giản. • Công bằng? • Không xét đến tính sử dụng! • Trang được nạp vào lâu nhất có thể là trang cần sử dụng thường xuyên! • Sử dụng càng nhiều frame ... càng có nhiều lỗi.  
**LRU – Cài đặt sử dụng hỗ trợ phần cứng:** • Thêm trường reference time cho mỗi phần tử trong bảng trang. • Thêm vào cấu trúc của CPU một bộ đếm counter. • Mỗi lần có sự truy xuất đến một trang trong bộ nhớ: Giá trị của counter tăng lên 1. • Giá trị của counter được ghi nhận vào reference time của trang tương ứng. Thay thế trang có reference time là min. Tuy nhiên, hệ thống có hỗ trợ phần cứng hoàn chỉnh để cài đặt LRU?  
**LRU – Cài đặt sử dụng stack:** • Tổ chức một stack lưu trữ các số hiệu trang. • Mỗi khi thực hiện một truy xuất đến một trang, số hiệu của trang sẽ được xóa khỏi vị trí hiện hành trong stack và đưa lên đầu stack. • Trang ở đỉnh stack là trang được truy xuất gần nhất, và trang ở đáy stack là trang lâu nhất chưa được sử dụng. • Thêm 1 bit “reference” có giá trị 0/1 vào mỗi phần tử của bảng trang: (• Được khởi gán là 0. • Được phần cứng đặt giá trị 1 mỗi lần trang tương ứng được truy cập. • Được phần cứng gán trở về 0 sau từng chu kỳ qui định trước. • Bit “reference” chỉ giúp xác định những trang có truy cập, không xác định thứ tự truy cập.) • Thêm N bit history phụ trợ vào mỗi phần tử của bảng trang: (Sau từng chu kỳ, bit reference sẽ được chép lại vào một bit history trước khi bị reset. N bit history sẽ lưu trữ tình hình truy xuất đến trang trong N chu kỳ cuối cùng.)  
**Second- chance algorithm (clock model):** • Sử dụng một bit reference duy nhất. • Chọn được trang nạn nhân theo FIFO. • Kiểm tra bit reference của trang đó:  
• Nếu reference = 0, đúng là nạn nhân rồi. • Nếu reference = 1, cho trang này một cơ hội thứ 2: (• reference = 0. • Thời điểm vào Ready List được cập nhật lại là thời điểm hiện tại.) • Chọn trang FIFO tiếp theo ...  
• Nhận xét: • Một trang đã được cho cơ hội thứ hai sẽ không bị thay thế trước khi hệ thống đã thay thế hết những trang khác. • Nếu trang thường xuyên được sử dụng, bit reference của nó sẽ duy trì được giá trị 1, và trang hầu như không bao giờ bị thay thế.  
**Not recently used algorithm**: • Sử dụng 2 bit Reference và Modify. • Với hai bit này, có thể có 4 tổ hợp tạo thành 4 lớp sau: • (0,0) không truy xuất, không sửa đổi. • (0,1) không truy xuất gần đây, nhưng đã bị sửa đổi. • (1,0) được truy xuất gần đây, nhưng không bị sửa đổi. • (1,1) được truy xuất gần đây, và bị sửa đổi. • Chọn trang nạn nhân là trang có độ ưu tiên cao nhất khi kết hợp bit R và bit M.  


**CHƯƠNG: GIỚI THIỆU**

**Chức năng của HDH:**- Cung cấp 1 giao diện máy tính ảo để sử dụng máy tính 1 cách trừu tượng 🡪 dễ dàng và an toàn.  
- Quản lý tài nguyên cho phép chương trình/người dùng chia sẻ phần cứng/tài nguyên 🡪 hiệu quả & công bằng.  
- Giúp đơn giản hóa với người dùng/lập trình viên bằng cung cấp các hàm hệ thống (system calls).- **Batch OS** (Hệ thống theo lô): chạy xong 1 ctr mới nạp ctr mới. Nhược điểm: thiếu sự tương tác với user; CPU phải đợi trong quá trình I/O.  
- **Spooling**:  
+ Tách thành 3 giai đoạn: input, process, output.  
+ Các ctr chạy cuộn vào nhau.  
- **Multiprogramming OS** (Đa chương):   
+ Có thể nạp nhiều ctr vào hệ thống cùng 1 lúc nhưng bản chất chỉ chạy 1 ctr tại 1 thời điểm. 🡪 tối ưu hiệu suất CPU, không để CPU rảnh.  
+ Những khoảng trống (wait) thì CPU không tận dụng.  
+ Nhược điểm: nếu 1 ctr chiếm quá lâu thì các ctr còn lại phải đợi (convoy effect); HDH không có khả năng thu hồi lại CPU khi 1 ctr đang thực hiện.  
- **Time-sharing OS**:  
+ Các chương trình sẽ luân phiên sở hữu CPU trong thời gian nhất định (time slice).  
+ Thay đổi đủ nhanh để user cảm giác là các ctr chạy // (pseudo - parallelsim) nhưng thực tế CPU chỉ thực hiện 1 ctr tại 1 thời điểm.  
+ HDH có khả năng thu hồi lại CPU/phần cứng (dừng 1 process đang chạy để nạp 1 process mới).

**CHƯƠNG: ARCHITECTURE TOUR  
Bản chất của máy tính:** là thực hiện vòng lặp.  
- Nạp lệnh từ bộ nhớ  
- Biên dịch (giải mã) để xem nó chạy cái gì  
- Thực hiện & lưu KQ  
🡪 Bước nạp lệnh là chậm nhất vì phải đi sao chép tập lệnh từ bộ nhớ (RAM…) còn 2 bước kia đều thực hiện trong nội bộ CPU (các th.ghi) 🡪 nhanh hơn.  
**Process modes:**  
- Tập lệnh thực thi ở: user mode hoặc system mode.  
- Th.ghi mode sẽ giữ CPU đang ở mode nào, và chỉ có thể điều chỉnh ở system mode.  
- OS code được thực thi ở system mode, chỉ có OS mới điều chỉnh được bộ nhớ của nó.  
- Tất cả đ.chỉ <100 đc dành riêng cho việc sdung HDH.  
**Exceptions:**- Xảy ra khi: CPU gặp lệnh ko thực thi đc. (VD: khi 1 user cố truy cập vào vùng nhớ chứa d.liệu/code của HDH hoặc do lỗi toán học như chia cho 0).  
- Khi xảy ra exception, CPU sẽ nhảy đến một đ.chỉ đc xác định trước trong HDH để xử lý lỗi đó. Các lỗi ≠ nhau sẽ nhảy đến các đ.chỉ ≠ nhau trong OS, gọi là "vectored".  
- Được sử dụng để xử lý các lỗi nghiêm trọng trong quá trình thực thi ctr và bảo vệ vùng nhớ hệ thống khỏi các truy cập bất hợp pháp từ user program.  
**Trap:**- Là lệnh đặc biệt trong user program y.cầu các dịch vụ từ HDH 1 cách hợp lệ. Trap ko là 1 lỗi mà là 1 cơ chế cho phép user program chuyển quyền điều khiển sang HDH để gọi các dịch vụ hệ thống.  
- Khi gặp trap, CPU sẽ nhảy đến 1 đ.chỉ cụ thể trong HDH & chuyển từ user mode 🡪system mode. Trap còn chứa các tham số để thông báo cho HDH biết dịch vụ cụ thể mà ctr y.cầu. Các tham số này được mã hóa trực tiếp trong trap hoặc trong registers.  
- Là nền tảng cho system call, cho phép user program truy cập các tài nguyên của HDH (tập tin, bộ nhớ…).  
**Interrupt:  
-** Là 1 sự kiện bên ngoài (ngoại lệ bên ngoài) khiến CPU phải tạm dừng ctr đang chạy để thực hiện 1 tác vụ ≠. Ngắt thường được tạo ra bởi p.cứng như bộ đếm thời gian (clock) hoặc t.bị I/O để yêu cầu HDH xử lý 1 sự kiện.  
**-** Khi có interrupt, CPU sẽ nhảy đến 1 đ.chỉ xác định để xử lý sự kiện & chuyển vào system mode. Q.trình yêu cầu CPU kiểm tra 1 đường tín hiệu bên ngoài đc thiết lập theo chu kỳ bởi clock/thiết bị.  
- Interrupt là cách để HDH giành lại quyền điều khiển CPU từ user program. (1 bộ đếm thời gian định kỳ tạo interrupt để CPU kiểm tra xem có cần chuyển đổi giữa các tiến trình đang chạy không – timesharing OS).  
Input & output:  
Card mạng có 2 th.ghi:  
+ TX: lưu dl gửi đi + RX: lưu dl gửi về  
**Giải pháp Memory-mapped:** + lệnh đơn giản: chỉ sao chép data cần gửi sang đ.chỉ ô TX 🡪 CPU ko cần xây thêm lệnh ms để chạy tbi ms. + Tự động các user program ko truy cập trực tiếp vào tbi.  
**Kỹ thuật ktra trạng thái bit (khi nào bit 1/0):** (chọn pp nào tùy vào tần suất đến của dl)  
+ **Polling** (tốt hơn trong t.huống dlieu đến liên tục, sẽ tiết kiệm chi phí hơn): CPU liên tục kiểm tra trạng thái của thiết bị I/O để xem liệu có dữ liệu nào cần xử lý không. (🡪 lãng phí tài nguyên; hiệu suất thấp khi truyền dl đến TX; tốn công sức ktra khi nhận từ RX)  
+ **Interrupt** (tốt hơn trong t.huống ko đoán đc tần suất đến của dlieu): khi có dữ liệu mới để nhận hoặc khi dữ liệu đã được truyền xong, thiết bị sẽ tạo ra một tín hiệu ngắt. (Chi phí của Interrupts: Mỗi lần có 1 ngắt, CPU phải dừng công việc hiện tại, xác định nguyên nhân của ngắt, lưu trạng thái của user, xử lý yêu cầu và sau đó khôi phục trạng thái. Quá trình này có thể tiêu tốn nhiều thời gian, đặc biệt nếu các ngắt xảy ra với tần suất cao.)  
**G.pháp copy dlieu từ vùng nhớ h.tại đến th.ghi TX:** (Kỹ thuật nào tốt hơn phụ thuộc vào kích thước gói tin) **1. Programmed I/O (PIO):**+ PIO yêu cầu CPU tự thực hiện toàn bộ quá trình truyền dữ liệu, bao gồm tải và lưu dữ liệu vào các thanh ghi của thiết bị I/O và liên tục kiểm tra các status register.  
+ Nhược: Tiêu tốn CPU cho toàn bộ quá trình, vì CPU phải tham gia vào từng byte dữ liệu truyền đi 🡪 PIO kém hiệu quả khi xử lý các gói dữ liệu lớn. Ưu: Chi phí xử lý thấp khi xét từng thao tác đơn lẻ, vì CPU chỉ cần kiểm tra trạng thái và gửi hoặc nhận dữ liệu. Phù hợp với các gói dữ liệu nhỏ.  
**2. Direct Memory Access (DMA):**+ DMA cho phép thiết bị tự động truy cập và truyền dữ liệu trực tiếp từ/đến bộ nhớ chính mà không cần sự can thiệp từng bước của CPU. Sau khi truyền dữ liệu xong, thiết bị gửi một tín hiệu ngắt để báo cho CPU rằng quá trình đã hoàn thành, giúp CPU không cần phải theo dõi liên tục.  
+ Nhược: Chi phí xử lý ban đầu cao hơn do cần thiết lập các thanh ghi base, count và chờ ngắt khi hoàn tất. Ưu: DMA hiệu quả hơn cho các gói dữ liệu lớn, vì CPU có thể tiếp tục xử lý các tác vụ khác trong khi quá trình truyền diễn ra tự động và không bị ràng buộc vào kích thước gói dữ liệu.  
**Booting:**- Là q.trình khởi động HDH từ lúc máy tính bật lên đến khi HDH ready để use. Quy trình gồm một chuỗi các sự kiện phần cứng & phần mềm, tạo thành 1 g.thức boot để bắt đầu nạp & chạy HDH.  
Giao thức boot diễn ra qua 3 giai đoạn:  
**1.** CPU bắt đầu từ 1 đ.chỉ cố định: Khi máy tính bật lên, CPU sẽ đc thiết lập để start execute từ 1 đ.chỉ cố định trong bộ nhớ (vd: kiến trúc x86, địa chỉ này là 0xFFFF0. Địa chỉ này thường được ánh xạ tới bộ nhớ chỉ đọc (ROM), chứa mã khởi động ban đầu.  
**2.** ROM chứa mã khởi động (firmware): ROM sẽ chứa 1 p.mềm đặc biệt gọi là firmware. Trên x86, firmware này được gọi là BIOS (Basic Input/Output System). BIOS chứa lượng mã tối thiểu để có thể đọc khối dữ liệu từ ổ đĩa. Khối dữ liệu này sẽ đc tải vào bộ nhớ & thực thi. Ctr này là boot loader.  
**3.** Boot loader nạp HDH: Boot loader là ctr chịu trách nhiệm nạp phần còn lại của HDH từ disk vào bộ nhớ. Boot loader đc cấu hình để nhận diện nhiều HDH hoặc nhiều phiên bản ≠ nhau của HDH, cho phép user chọn HDH để khởi động.  
**Tại sao lại cần giao thức boot?  
1.** Với giao thức boot, HDH ko lưu trữ trực tiếp trong ROM, giúp dễ dàng thay đổi or cập nhật mà ko ảnh hưởng đến phần cứng. **2.** Boot loader có thể nạp nhiều HDH ≠ nhau hoặc các phiên bản khác nhau của cùng 1 HDH. **3.** Giao thức boot cho phép khởi động từ nhiều th.bị ≠ nhau, hỗ trợ các tính năng bảo mật khi khởi động qua mạng. **4.** HDH hiện đại có kích thước từ 4-8MB, quá lớn để lưu trữ toàn bộ trong ROM. Vì vậy, sử dụng boot loader để nạp OS từ đĩa là giải pháp hiệu quả hơn.

**CHƯƠNG: PROCESSES**- Ctr: là tập hợp các dòng lệnh viết = nnlt  
- Procces: là 1 p.bản đang đc thực thi của ctr; là đ.tg động của ctr khi nó đc nạo vào bộ nhớ & gắn vs những resource để execute nó.  
- Biến toàn cục đc: cấp phát khi ctr bắt đầu; thu hồi khi ctr kết thúc.  
- Biến cục bộ: đc cấp phát khi gọi hàm chứa nó.  
- Stack: đc HDH cung cấp giới hạn nên nếu khai báo với k/thước lớn 🡪 stackoverflow.  
- Heap: đc HDH cung cấp linh động nên dù có khai báo lớn ntn thì ko bị gì.  
**Context switch**: là q/trình HDH stop 1 process đang thực thi để thực thi 1 process ≠ khi process bị lấy lại HDH. (Lưu state của 1 process h.tại vào PCB của nó; reload state từ PCB của process muốn thực thi). Bị lấy lại khi: 1. Có sự kiện xảy ra 🡪 ko chạy đc nữa 2. Interrupt từ hệ thống phát ra.  
**- 3 cách để 1 process đc tạo ra:**+ chính hệ thống là những process system để thực thi những process ≠.  
+ click vào biểu tượng ctr (như click word trên screen)  
+ chủ động gọi những system call để tạo 1 process ms.  
- Khi 1 process ms đc tạo thì HDH sẽ:  
+ Tạo 1 id ms cho process  
+ Tạo 1 PCB cho process ms này.  
**Interprocess communication (IPC):**1. Message passing (tiêu biểu socket)  
2. Shared-memory: A sẽ tạo 1 vùng shared-memory trong vùng nhớ dành riêng cho A = system calls. B muốn sdung thì sẽ attach vùng nhớ dùng chung này vào ko gian địa chỉ của B.  
(3. Pipe: sử dụng 1 file đặc biệt như là 1 pipe để khi 1 process muốn process ≠ đọc dlieu thì ghi vào pipe này. Chia sẻ theo 1 chiều: writer 🡪 reader (reader ko viết lại cho writer đc. Nếu muốn viết thì tạo 1 pipe ms, reader đóng v.trò writer & ngược lại).  
Hai loại: ordinary pipes: dành cho mqh cha con; name\_pipes: 2 process bất kỳ.)

**CHƯƠNG: PROCESS SCHEDULING**- CPU burst: là giai đoạn đang sở hữu CPU & đc thực thi (các phép tính toán…)  
- I/O burst: là giai đoạn chờ dl nhập/xuất.  
🡪 Việc thực thi 1 process là 1 sự luân phiên giữa CPU & I/O burst.  
**- CPU Scheduler:** Chọn các process đang trong ready queue để thực thi và giao CPU cho nó. 3 loại: long-term sheduler, short-term scheduler: lựa chọn process để giao cho CPU, medium-term scheduler. Sẽ diễn ra khi 1 process:  
+ Từ running sang waiting state. (1)  
+ Từ running sang exit. (2)  
+ Từ running sang ready state.  
+ Từ waiting sang ready state.  
(1) và (2): nonpreemptive còn lại là preemptive.  
**- Dispatcher:** giao CPU cho tiến tình đc chọn bởi short-term scheduler. (switching context, switching to user mode, nhảy vào vị trí cụ thể trong user program để khởi động ctr đó.)  
**- Scheduling criteria:**   
+ throughput: slg process h.thành trên đ.vị thời gian.  
+ turnaround time: khoảng tgian khi thực hiện 1 process. Từ khi ctrinh bắt đầu vào hệ thống đến khi kết thúc.  
+ waiting time: là lg tgian nằm trong ready queue.  
+ reponse time: là khoảng tgian khi process gửi y.cầu đến khi CPU respond lại.  
RMS (rate monotonic scheduling): tk nào có chu kỳ ngắn thì ưu tiên sớm nhất (vì dễ gặp deadline sớm nhất).  
EDF (Earliest Deadline First): ưu tiên tk nào có deadine sớm nhất chạy trước

**CHƯƠNG: ĐỒNG BỘ  
Race condition (tranh đoạt điều khiển):**   
- Khi nhiều process có d.liệu dùng chung & truy cập đồng thời 🡪 Data ko nhất quán & vô lý. Là tình huống mà kết quả của ctr phụ thuộc vào sự điều phối của hệ thống. (1 input nhưng cho ra nhiều output).  
**Critical section (miền găng):** đoạn mã nguồn đọc/ghi dl lên vùng nhớ dùng chung.  
**Giải pháp đúng phải thỏa:**  
- Mutal exclusion: chỉ 1 process đc thực thi trong CS tại 1 thời điểm.  
- Ko có giả thiết về tốc độ CPU, slg CPU.  
- Ko có process ở ngoài CS lại khóa cho process ≠ vào.  
- Starvation: ko có process nào đợi mãi mãi để vào CS.  
**I. Giải pháp Busy Waiting: (nhc điểm chug: lãng phí CPU)  
a. Giải pháp phần mềm:**1. Lock variable: 2 process có thể vào miền găng tại 1 th.điểm (vì biến lock cx là 1 biến dùng chung)  
2. Strict Alternation: giải quyết đc mutal exclusion. Nhưng vi phạm 1 process ngoài CS không cho process ≠ vào CS (vì thời gian cần thiết thực thi của các anh là ≠ nhau).  
3. Giải pháp Peterson: Nhược (priority inversion: xảy ra khi 1 process có độ ưu tiên cao hơn bị block bởi 1 process có độ ưu tiên thấp hơn).  
**b. Giải pháp phần cứng:**1. Vô hiệu hóa ngắt: (2 loại ngắt phổ biến)  
+ Timer interrupts: q.lý việc đ.phối CPU  
+ I/O device Interrupts: báo hiệu I/O đã xong (báo cho CPU biết để chuyển process từ state blocled sang ready.) PP này cho phép 1 process khi vào CS sẽ vô hiệu hóa các ngắt vì: (+ Khi 1 timer ngắt bật lên thì scheduler ms chạy 🡪 dẫn đến context switch) 🡪 ko ngắt 🡪 ko CPU scheduling 🡪 ko CS probs.  
+ Nhược: nếu 1 process bị block mà chưa bật lại ngắt 🡪 HDH die. + ko có tác dụng vs hệ thống có nhiều CPU.  
2. TSL (Test\_And\_Set)  
- Là 1 chỉ thị lệnh của CPU (≠ lock var là do lập trình viên).  
- Chỉ thị lệnh của CPU (atomic operation): nếu khối lệnh đc bỏ vào chỉ thị lệnh thì sẽ ko bị lấy lại CPU trong quá trình nó thực hiện khối lệnh này.  
- Chỉ có 1 process dc gọi chỉ thị test\_and\_set\_lock tại 1 th.đ.  
**II. Giải pháp Sleep & Wakeup:**1. Semaphore:  
- Là 1 biến có các thuộc tính: 1 gtri nguyên dg e(s), 1 hàng đợi f(s) lưu ds các process đang bị khóa. Cho phép ktra có đc vào CS hay ko thông qua 2 atomic operations down & up.  
+ down: giảm gtri của semaphore s đi 1 đvi nếu e(s)>0, & tiếp tục xử lý. Nếu e(s)=0, process phải chờ đến khi e(s)>0. (Trc khi vào CS, process sẽ gọi down để ktra có đc vào CS hay ko. Nếu biến này thỏa đk thì anh gọi hàm down đc vào, ko thì bị sleep).  
+ up:Khi ra khỏi CS, process sẽ gọi up để báo hiệu ra khỏi CS 🡪 tăng biến semaphore, đồg thời wakeup 1 anh nào đó đang bị ngủ khi đã gọi down để tiếp tục thực thi.