**1. Quản lý bộ nhớ (Memory Management)**

**Vai trò & mục tiêu chính**

* Theo dõi bộ nhớ (RAM): OS cần biết phần nào của bộ nhớ đang được sử dụng, phần nào còn trống, ai sử dụng, để cấp phát và thu hồi.
* Cấp phát bộ nhớ (allocation) cho các tiến trình khi chúng khởi động hoặc khi cần thêm bộ nhớ trong quá trình thực thi.
* Bảo vệ bộ nhớ: đảm bảo một tiến trình không thể truy cập vùng nhớ của tiến trình khác (đánh dấu vùng nào được phép đọc/ghi) — giúp ngăn lỗi và vi phạm bảo mật.
* Quản lý không gian ảo (virtual memory): khi RAM vật lý không đủ, OS có thể dùng ổ đĩa làm “bộ nhớ phụ” (swap / paging) để mở rộng không gian địa chỉ mà mỗi tiến trình “thấy”.
* Giải quyết nội/ngoại phân mảnh: theo dõi phân mảnh của bộ nhớ và nếu cần làm “compaction” hoặc tổ chức lại.

**Cách hoạt động trong hệ điều hành**

1. Khi tiến trình được tạo, OS xác định kích cỡ bộ nhớ cần thiết, yêu cầu cấp phát.
2. OS chọn vùng nhớ trống (physical frames) để gán cho các trang (pages) hoặc phân đoạn (segments) của tiến trình.
3. OS duy trì bảng chuyển đổi từ địa chỉ ảo (virtual address) sang địa chỉ thực (physical address), thường qua bảng trang (page table) trong cơ chế phân trang (paging).
4. Nếu tiến trình truy cập một trang mà trang đó chưa được load (page fault), OS phải mang trang từ ổ đĩa vào RAM, có thể hoán đổi (swap out) trang ít dùng đi.
5. Khi tiến trình kết thúc hoặc giải phóng bộ nhớ, OS thu hồi lại vùng nhớ đó để cấp lại cho tiến trình khác.

**Ví dụ thực tế**

* Khi bạn mở trình duyệt Chrome: OS cấp phát cho tiến trình Chrome một vùng nhớ đủ để chứa mã chương trình và dữ liệu tạm. Nếu bạn mở nhiều tab (tiến trình nhỏ hoặc luồng nhỏ), OS sẽ tiếp tục cấp phát thêm cho chúng khi cần.
* Nếu bạn mở quá nhiều tab khiến RAM không đủ, hệ điều hành (ví dụ Windows, Linux) sẽ đẩy (swap) một số trang ít dùng từ RAM xuống ổ cứng, để nhường chỗ cho các trang mới cần thiết — quá trình này có thể gây hiện tượng “disk thrashing” nếu swap quá nhiều.
* Trong Linux, cơ chế paging + yêu cầu trang (demand paging) và thuật toán thay thế trang (như LRU, CLOCK) được OS dùng để quyết định trang nào nên bị hoán đổi ra.

**2. Quản lý thiết bị nhập / xuất (I/O Device Management)**

**Vai trò & mục tiêu chính**

* Cung cấp giao diện trừu tượng để chương trình ứng dụng sử dụng các thiết bị (đĩa, bàn phím, chuột, máy in, mạng, USB) mà không cần biết cấu trúc phần cứng cụ thể.
* Quản lý driver: OS cần giữ và tải driver (trình điều khiển) cho từng thiết bị để biết cách giao tiếp.
* Quản lý truy cập đồng thời: nếu nhiều tiến trình muốn dùng cùng một thiết bị, OS cần điều phối — ví dụ xếp hàng (scheduling) cho các yêu cầu I/O.
* Sử dụng buffer/ caching/ spooling: để làm mượt việc truyền dữ liệu, giảm độ trễ và tăng hiệu quả.
* Xử lý ngắt (interrupts) từ thiết bị: khi thiết bị báo “tôi đã sẵn sàng / hoàn thành” qua ngắt, OS phản hồi và tiếp tục công việc.
* Quản lý lỗi I/O: nếu thiết bị gặp sự cố, OS phát hiện và thực hiện các biện pháp xử lý (retry, thông báo lỗi).

**Cách hoạt động trong hệ điều hành**

1. Khi tiến trình muốn thực hiện I/O (ví dụ ghi ra đĩa), nó sẽ gọi một “system call” như write() hoặc read().
2. OS kiểm tra xem driver thiết bị đó đã được load chưa; nếu chưa, tải driver tương ứng.
3. Giao yêu cầu đến driver, driver tương tác với phần cứng (thông qua cổng I/O, bus, lệnh điều khiển).
4. Khi thiết bị hoàn thành công việc (ví dụ đã ghi xong), nó gửi ngắt (interrupt) tới CPU. Kernel (phần trung tâm OS) xử lý ngắt, xác nhận hoàn thành, giao tiếp với tiến trình chờ.
5. Nếu có nhiều yêu cầu I/O đến cùng thiết bị, OS có thể sắp xếp thứ tự truy cập (I/O scheduling), ví dụ ưu tiên các yêu cầu nhỏ hoặc gần nhau, để tối ưu hóa hiệu suất đĩa.

**Ví dụ thực tế**

* Khi bạn in một tài liệu PDF: ứng dụng gửi yêu cầu in, OS chuyển dữ liệu đến driver máy in (spooling): dữ liệu được lưu vào một hàng đợi (spool) trước khi máy in thực sự xử lý.
* Khi bạn gắn USB: OS phát hiện thiết bị mới, tải driver USB, gán mount point (Linux), và cho phép truy cập file trên USB.
* Trong hệ thống Linux, driver mạng (network driver) giúp OS gửi/ nhận gói tin qua cạc mạng; nếu cạc mạng ngắt khi dữ liệu đến, kernel nhận ngắt, trích xuất gói tin và chuyển vào stack mạng (IP, TCP).

**3. Cung cấp giao diện người dùng (User Interface / Shell / GUI)**

**Vai trò & mục tiêu chính**

* Cung cấp cách để người dùng tương tác với hệ điều hành: chạy chương trình, quản lý file, cấu hình hệ thống, v.v.
* Hai kiểu phổ biến: giao diện dòng lệnh (CLI – Command Line Interface) hoặc giao diện đồ hoạ (GUI – Graphical User Interface).
* Ẩn chi tiết phức tạp phần cứng và hệ thống, cung cấp các lệnh, menu, cửa sổ, biểu tượng, thanh taskbar,…
* Cho phép người dùng gửi yêu cầu (như “mở file”, “sao chép”, “xóa”, “cài đặt”) và nhận kết quả hiển thị.

**Cách hoạt động trong hệ điều hành**

* Khi người dùng nhập lệnh hoặc bấm chuột, giao diện (shell hoặc GUI) chuyển yêu cầu đó thành system call tương ứng (ví dụ open, exec, read, write).
* GUI chứa các thành phần đồ họa (cửa sổ, widget, biểu tượng) do hệ thống quản lý cửa sổ (window manager) phối hợp với kernel và thư viện đồ hoạ (như X11, Wayland, Windows GDI).
* Giao diện người dùng cũng nhận sự kiện từ thiết bị đầu vào (chuột, bàn phím), chuyển chúng tới ứng dụng tương ứng.
* OS hoặc thành phần giao diện đảm bảo hiển thị kết quả (vẽ cửa sổ, cập nhật màn hình) và phản hồi giao diện mượt mà.

**Ví dụ thực tế**

* Windows: bạn nhấn nút Start → chọn “Notepad” → OS thực hiện lệnh để tải Notepad.exe, tạo tiến trình, hiển thị cửa sổ.
* Linux (Ubuntu, GNOME): bạn bấm biểu tượng Files → GUI shell (GNOME Shell) gọi chương trình quản lý file (nautilus), cửa sổ mở.
* CLI: trong Linux, bạn gõ ls -l /home/user, shell (bash / zsh) nhận lệnh, gọi system call execve() để chạy lệnh ls, rồi hiển thị kết quả trong terminal.

**4. Quản lý hệ thống tệp (File System Management)**

**Vai trò & mục tiêu chính**

* Ẩn chi tiết vật lý của thiết bị lưu trữ (đĩa cứng, SSD, USB) khỏi người dùng — cung cấp cấu trúc logic (cây thư mục, files).
* Cung cấp các thao tác cơ bản: tạo file, mở, đọc, ghi, đóng, xóa, đổi tên.
* Theo dõi metadata (thuộc tính file): tên, kích thước, ngày tạo, ngày sửa, quyền truy cập.
* Quản lý không gian ổ đĩa: biết phần nào của đĩa còn trống, phần nào đã dùng, phân bổ khối (blocks) cho file.
* Bảo mật và quyền truy cập: kiểm soát ai (user, group) có quyền đọc/ghi/thực thi file.
* Liên kết vật lý & logic: xác định vị trí khối dữ liệu của file trên đĩa, thông qua bảng phân bổ (allocation tables), cây chỉ mục, inode, v.v.
* Duy trì tính nhất quán: bảo vệ file không bị hỏng khi đột ngột mất điện; dùng các giao dịch journaling, kiểm tra lỗi (fsck).

**Cách hoạt động trong hệ điều hành**

1. Khi người dùng muốn mở file: gọi system call như open(), OS kiểm tra quyền, tìm vị trí khối dữ liệu, đọc từ đĩa vào bộ đệm (buffer cache).
2. Khi ghi file: OS ghi dữ liệu vào bộ đệm (cache), sau đó định kỳ flush xuống đĩa.
3. OS cập nhật metadata (thời gian sửa, kích thước).
4. OS giữ bảng khối trống, bảng khối đã dùng để biết chỗ nào vẫn trống để cấp cho file mới.
5. Trong hệ thống tập tin có journaling (ví dụ ext4, NTFS), trước khi thực hiện thay đổi dữ liệu/metadata, OS ghi lệnh vào nhật ký (journal) để có thể hồi phục khi có lỗi.

**Ví dụ thực tế**

* Trên Windows: hệ thống tệp NTFS – khi bạn lưu tài liệu Word, OS ghi dữ liệu vào các cluster, cập nhật MFT (Master File Table) để lưu thông tin file như tên, quyền, vị trí khối.
* Trên Linux: hệ thống ext4 hoặc XFS – bạn tạo thư mục mới mkdir /home/user/newdir, OS tạo một entry trong thư mục cha, cấp một inode mới cho thư mục, cập nhật bảng block.
* Khi ổ đĩa bị cắt điện bất thường: nếu hệ thống dùng journaling, trên lần khởi động tiếp theo OS kiểm tra journal và hoàn thiện hoặc khôi phục các thay đổi chưa kịp ghi hoàn chỉnh.

**5. Quản lý tiến trình (Process / Process Management)**

**Vai trò & mục tiêu chính**

* Quản lý các chương trình đang chạy (các tiến trình), từ khi khởi tạo đến khi kết thúc.
* Sắp xếp tiến trình (scheduling): quyết định tiến trình nào được chạy, trong bao lâu, ưu tiên nào.
* Chuyển đổi ngữ cảnh (context switching): khi đổi giữa các tiến trình, OS lưu trạng thái (registers, PC, bộ đếm stack) của tiến trình hiện tại và khôi phục tiến trình khác.
* Đồng bộ hoá & giao tiếp giữa tiến trình (inter-process communication – IPC): khi tiến trình cần chia sẻ dữ liệu hoặc phối hợp (shared memory, pipe, message queue, semaphore).
* Xử lý deadlock và tránh xung đột tài nguyên: khi nhiều tiến trình cần tài nguyên hạn chế (ví dụ in ấn, thiết bị), OS phải ngăn chặn hoặc giải quyết trường hợp deadlock.
* Thực hiện đa nhiệm (multitasking), chia sẻ CPU giữa nhiều tiến trình, đảm bảo tính đáp ứng (responsiveness).

**Cách hoạt động trong hệ điều hành**

1. Khi người dùng hoặc hệ thống yêu cầu chạy một chương trình, OS tạo một tiến trình mới: cấp PID, tạo Process Control Block (PCB) chứa thông tin trạng thái, con trỏ đến mã, bộ nhớ, tài nguyên. [Wikipedia+1](https://en.wikipedia.org/wiki/Process_control_block?utm_source=chatgpt.com)
2. OS đưa tiến trình vào hàng chờ (ready queue).
3. Bộ lập lịch (scheduler) chọn tiến trình tiếp theo chạy dựa trên thuật toán (Round Robin, priority, SJF, multi-level queue).
4. Khi tiến trình bị ngắt (do timer, I/O, yield), OS thực hiện context switch: lưu trạng thái hiện tại vào PCB, load trạng thái của tiến trình khác.
5. Các tiến trình muốn giao tiếp sẽ sử dụng IPC: ví dụ gửi tín hiệu, dùng vùng nhớ chung, message passing.
6. Khi tiến trình kết thúc, OS thu hồi tài nguyên (bộ nhớ, file mở, thiết bị) và loại khỏi bảng tiến trình.

**Ví dụ thực tế**

* Khi bạn mở một trình duyệt: OS tạo tiến trình (process) cho trình duyệt, sau đó nếu bạn mở thêm tab hoặc mở thêm cửa sổ, có thể OS tạo nhiều tiến trình con hoặc nhiều luồng (threads) cho từng tab.
* Task Manager trong Windows hiển thị danh sách tiến trình, CPU%, memory usage — đó là “mặt phẳng” người dùng cho quản lý tiến trình. [Wikipedia](https://en.wikipedia.org/wiki/Task_Manager_%28Windows%29?utm_source=chatgpt.com)
* Trong Linux, mỗi lệnh như ps aux liệt kê các tiến trình, kill PID gửi tín hiệu để kết thúc tiến trình. OS xử lý tín hiệu trong kernel, tiến trình nhận tín hiệu và nếu không xử lý được sẽ bị dừng.
* Trong hệ thống server đa nhiệm, OS chia CPU giữa web server, cơ sở dữ liệu, background job để đảm bảo không có tiến trình nào chiếm dụng CPU vô tận — nhờ thuật toán lập lịch ưu tiên, tỷ lệ thời gian, preemptive scheduling.