# ĐỀ CƯƠNG ÔN TẬP

**I. Lý thuyết**

**1. Định nghĩa hệ điều hành?**

Chương trình trung gian giữa phần cứng máy tính và người sử dụng, có chức năng điều khiển và phối hợp việc sử dụng phần cứng và cung cấp các dịch vụ cơ bản cho các ứng dụng

**2. Trình bày các dịch vụ hệ điều hành cung cấp?**

* *Giao diện người dùng*: CLI, GUI, touch-screen, batch
* *Thực thi chương trình* (program execution)
* *Thao tác I/O* (I/O operations) theo yêu cầu của chương trình
* *Các thao tác trên hệ thống file* (file system manipulation)
* *Trao đổi thông tin giữa các tiến trình* bằng việc chia sẻ bộ nhớ (shared memory) và truyền thông điệp (message passing)
* *Phát hiện lỗi* do bộ nhớ, I/O, chương trình
* *Cấp phát tài nguyên* (CPU, bộ nhớ chính, ổ cứng) hoặc bằng thủ tục (routines)
* *Ghi nhật ký* (logging)
* *Bảo vệ* (protection): kiểm soát việc truy xuất tài nguyên của hệ thống (v.d. 2 tiến trình khác nhau không ảnh hưởng nhau)
* *Bảo mật* (security) bao gồm xác thực, phân quyền user cho phép truy cập vào các tài nguyên của hệ thống (v.d đăng nhập bằng username và password).

**3. Cho biết mục đích của lời gọi hệ thống (system call)?**

Mục đích của lời gọi hệ thống (system call) là cho phép các chương trình ứng dụng truy cập các dịch vụ do hệ điều hành cung cấp mà không cần trực tiếp thao tác với phần cứng. Cụ thể:

Cầu nối giữa chương trình người dùng và hệ điều hành

Cung cấp quyền truy cập an toàn đến tài nguyên hệ thống

Hỗ trợ thao tác hệ thống cơ bản như:

+ Quản lý tiến trình (tạo, kết thúc, đồng bộ hóa tiến trình)

+ Thao tác với tập tin và thư mục

+ Giao tiếp giữa các tiến trình

+ Truy cập thiết bị đầu vào/ra

**4. Cho biết mục đích của chương trình hệ thống (system program)?**

Mục đích của chương trình hệ thống (system program) là cung cấp một môi trường thuận tiện để phát triển và thực thi chương trình:

Hỗ trợ phát triển chương trình

Quản lý và thao tác tập tin

Cung cấp thông tin hệ thống

Thực thi chương trình

Giao tiếp giữa người dùng và hệ thống

Dịch vụ nền (background services/daemons)

**5. Có những giao tiếp (interface) nào giữa người dùng và hệ điều hành?**

Các loại giao tiếp (interface) giữa người dùng và hệ điều hành bao gồm:

Giao diện dòng lệnh (CLI – Command-Line Interface)

Giao diện đồ họa (GUI – Graphical User Interface)

Giao diện cảm ứng (Touchscreen Interface)

Giao diện lập trình ứng dụng (API – Application Programming Interface)

**6. Định nghĩa tiến trình? So sánh chương trình (program) và tiến trình (process)?**

Tiến trình là một chương trình đang được thực thi, bao gồm mã chương trình, dữ liệu, ngăn xếp (stack), và bộ đếm chương (program counter). Đây là đơn vị cơ bản để thực hiện công việc trong hệ điều hành​.

|  |  |  |
| --- | --- | --- |
| **Tiêu chí** | **Chương trình** | **Tiến trình** |
| Khái niệm | Là tập hợp các chỉ thị được lưu trữ trên địa hoặc bộ nhớ | Là chương trình đang được thực thi trong hệ thống |
| Tồn tại | Tồn tại vĩnh viễn cho đến khi bị xóa | Tồn tại tạm thời |
| Thành phần chính | Mã nguồn (code) | Mã chương trình, dữ liệu, ngăn xếp,… |
| Khả năng thực thi | Không tự thực thi | Có thể được lập lịch và thực thi bởi hệ điều hành |

**7. Trình bày các trạng thái của tiến trình (Process state)?**

New (Tạo mới): Tiến trình đang được tạo ra, chưa được hệ điều hành đưa vào hàng đợi thực thi​

Ready (Sẵn sàng): Tiến trình đã sẵn sàng để thực thi và chờ đợi CPU.

Running (Đang chạy): Tiến trình đang được CPU thực thi. Tại một thời điểm chỉ có một tiến trình chạy trên một CPU (đơn nhân)​

Waiting (Chờ): Tiến trình đang chờ một sự kiện xảy ra (ví dụ: khi CPU xử lý tác vụ khác hoăc chờ I/O hoàn tất)

Terminated (Kết thúc): Tiến trình đã hoàn thành thực thi và kết thúc xử lý.

**8. Process Control Block (PCB) là gì? Trình bày chi tiết?**

Process Control Block (PCB) là cấu trúc dữ liệu quan trọng mà hệ điều hành dùng để lưu trữ toàn bộ thông tin liên quan đến một tiến trình. Mỗi tiến trình trong hệ thống đều có một PCB riêng biệt.

PCB thường chứa các thông tin sau:

* **ID tiến trình (PID)**
* **Trạng thái tiến trình**
* **Bộ đếm chương trình**
* **Giá trị thanh ghi**
* **Thông tin về quản lý bộ nhớ**
* **Thông tin về các thiết bị I/O**
* **Thông tin lập lịch**

**9. Context Switch là gì?**

Context Switch (chuyển đổi ngữ cảnh) là quá trình CPU chuyển đổi tiến trình đang chạy sang một tiến trình khác bằng việc lưu trạng thái (context) của tiến trình hiện tại vào PCB, và tải trạng thái đã được lưu trước đó của tiến trình mới từ PCB vào CPU.

**10. Thread là gì? Cho biết các lợi ích khi sử dụng Thread?**

Thread (luồng) là đơn vị nhỏ nhất của tiến trình có thể được lên lịch và thực thi độc lập.

Một tiến trình (process) có thể chứa nhiều thread. Các thread trong cùng một tiến trình chia sẻ cùng không gian địa chỉ (code, dữ liệu, tài nguyên), nhưng có stack và thanh ghi riêng biệt

Lợi ích của việc dùng Thread:

Hiệu suất cao hơn

Phản hồi nhanh hơn

Tận dụng CPU thuận tiện

Chia sẻ dữ liệu dễ dàng

Thích hợp cho ứng dụng thời gian thự

**11. Cho biết 4 trường hợp lập lịch CPU (CPU Scheduling) thực hiện?**

Lập lịch CPU (CPU Scheduling) được thực hiện khi:

+ Chuyển từ trạng thái Running (Đang chạy) sang Waiting (Chờ Đợi)

+ Chuyển từ trạng thái Running (Đang chạy) sang Ready (Sẵn sàng)

+ Chuyển từ trạng thái Waiting (Chờ Đợi) sang Ready (Sẵn sàng)

+ Kết thúc thực thi (Terminated)

**12. Phân biệt Preemptive và Nonpreemptive Scheduling**

|  |  |  |
| --- | --- | --- |
| **Tiêu chí** | **Preemptive** | **Nonpreemptive** |
| **Quyền điều phối CPU** | OS có thể thu hồi CPU | CPU không bị thu hồi |
| **Mức độ phản hồi** | Tốt hơn vì không có 1 tiến trình độc chiếm CPU quá lâu | Chậm hơn |
| **Chi phí** | Cao do context switching nhiều | Thấp do ít context switching |
| **Thuật toán** | Round Robin, SRTF | FCFS, SJF (không ưu tiên) |

**13. Giải thích các tiêu chuẩn lập lịch sau:**

**Turnaround time**

Định nghĩa: Là khoảng thời gian từ khi tiến trình được đưa vào hệ thống đến khi hoàn tất.

Công thức: Turnaround Time = Completion Time - Arrival Time

**Waiting time**

Định nghĩa: Là tổng thời gian tiến trình chờ trong hàng đợi Ready trước khi được CPU xử lý

Công thức: Waiting Time = Turnaround Time - Burst Time

**Response time**

Định nghĩa: khoảng thời gian từ khi một process đưa ra yêu cầu đến khi yêu cầu đầu tiên được đáp ứng (trong các hệ thống thời gian chia sẻ - time-sharing và tương tác - interactive system)

Công thức: Response Time = First Run Time - Arrival Time

**14. Race condition là gì?**

Race Condition xảy ra khi nhiều tiến trình truy cập và thao tác đồng thời lên dữ liệu dùng chung (shared memory), và kết quả phụ thuộc vào thứ tự thực thi của các tiến trình.

Ví dụ: 2 tiến trình cùng lúc tăng biến x từ 0 đến 1 có thể dẫn đến giá trị x là 1 thay vì 2 nếu như không có cơ chế đồng bộ hóa

**15. Critical Section Problem là gì? Trình bày các điều kiện của bài toán Critical Section Problem?**

Critical Section là đoạn mã truy cập tài nguyên chia sẻ. CS Problem là bài toán đặt ra đảm bảo rằng chỉ có duy nhất một tiến trình được truy cập vào CS tại một thời điểm nhất định.

3 điều kiện cần:

* Mutual Exclusion: chỉ có duy nhất một tiến trình ở trong CS tại một thời điểm nhất định
* Progress: chỉ có các tiến trình không đang thực thi ở Remainder Section mới có thể quyết định tiến trình nào sẽ vào CS, và việc quyết định này không được diễn ra vô hạn.
* Bounded Waiting: một process được vào CS sau một khoảng thời gian hữu hạn.

**16. Trình bày giải pháp Peterson (Peterson’s solution)? Chứng minh giải pháp Peterson thỏa mãn các yêu cầu của bài toán CS?**

Peterson’s Solution là một giải thuật cổ điển, đơn giản để giải quyết bài toán vùng tranh chấp (Critical Section Problem) cho hai tiến trình (gọi là P0 và P1)​

Ý tưởng chính:

Dùng hai biến dùng chung:

flag[0], flag[1]: báo hiệu tiến trình P0 hoặc P1 muốn vào vùng tranh chấp.

turn: cho biết tiến trình nào được ưu tiên vào nếu có tranh chấp.

Code giải thuật Peterson:

// Biến dùng chung:

boolean flag[2]; // flag[0] và flag[1]

int turn;

// Code cho tiến trình Pi (i = 0 hoặc 1):

do {

flag[i] = true; // Báo hiệu muốn vào vùng tới hạn

turn = j; // Nhường quyền cho tiến trình kia

while (flag[j] == true && turn == j); // Đợi khi tiến trình kia muốn vào và đang được ưu tiên

// ---- Critical Section ----

flag[i] = false; // Thoát vùng tới hạn

// ---- Remainder Section ----

} while (true);

Chứng minh giải thuật đáp ứng điều kiện:

Mutual Exclusion: Nếu cả hai tiến trình cùng muốn vào (flag[0]=flag[1]=true), biến turn quyết định chỉ một tiến trình được vào.

Progress: Nếu một tiến trình không muốn vào (flag[j] = false), tiến trình kia sẽ không bị chặn trong vòng while.

Bounded Waiting: Sau một số lần nhường quyền (turn thay đổi), tiến trình đang chờ sẽ được vào vùng tới hạn trong thời gian hữu hạn.

**17. Định nghĩa Mutex? Cho ví dụ?**

Mutex (Mutual Exclusion Object) là một cơ chế đồng bộ hóa được dùng để bảo vệ vùng tới hạn, đảm bảo chỉ một tiến trình hoặc một luồng có thể truy cập tài nguyên dùng chung tại một thời điểm​.

Mutex hoạt động như một khóa (lock):

Lock (khóa): Tiến trình/luồng chiếm giữ quyền truy cập.

Unlock (mở khóa): Tiến trình/luồng trả lại quyền cho tiến trình khác.

Ví dụ:

// Khởi tạo Mutex

mutex m = unlocked;

// Thread i:

do {

wait(m); // lock mutex

// ---- Critical Section ----

signal(m); // unlock mutex

// ---- Remainder Section ----

} while (true);

**18. Định nghĩa Semaphore? Trình bày ví dụ về việc sử dụng trong các trường hợp sau:**

Semaphore là một biến đồng bộ hóa đặc biệt, được dùng để quản lý quyền truy cập tài nguyên chung giữa nhiều tiến trình, giúp giải quyết bài toán vùng tới hạn (Critical Section) và đồng bộ hóa tiến trình​

Semaphore có giá trị nguyên (int), và chỉ được thao tác bằng 2 phép toán nguyên tử:

wait(S) (hoặc P(S)): giảm giá trị S đi 1. Nếu S < 0, tiến trình bị chặn lại.

signal(S) (hoặc V(S)): tăng giá trị S lên 1. Nếu có tiến trình đang chờ, đánh thức một tiến trình đó.

Giải quyết n process truy xuất vào CS.

Dùng 1 Semaphore nhị phân (giống như Mutex) để đảm bảo chỉ 1 tiến trình vào vùng tranh chấp tại một thời điểm.

// Khởi tạo semaphore:

Semaphore mutex = 1;

Process i (i = 1 đến n) thực hiện:

do {

wait(mutex); // Yêu cầu quyền vào vùng tranh chấp

// ---- Critical Section ----

signal(mutex); // Thoát vùng tranh chấp

// ---- Remainder Section ----

} while (true);

Giải quyết đồng bộ giữa 2 process.

// Khởi tạo semaphore:

Semaphore sync = 0;

Process P1 (Producer):

{

// Produce data

signal(sync); // Báo hiệu đã sẵn sàng

}

Process P2 (Consumer):

{

wait(sync); // Chờ tín hiệu từ P1

// Consume data

}

**20. Trình bày các điều kiện để xảy ra Deadlock?**

Bốn điều kiện cần và đủ để xảy ra deadlock là:

Mutual Exclusion: Tài nguyên không thể chia sẻ được; tại mỗi thời điểm, chỉ một tiến trình sử dụng tài nguyên.

Hold and Wait: Một tiến trình đang giữ ít nhất một tài nguyên và chờ cấp phát thêm tài nguyên khác đang bị giữ bởi tiến trình khác.

No Preemption: tài nguyên không thể bị lấy lại mà chỉ có thể được trả lại từ tiến trình đang giữ tài nguyên đó khi tiến trình hoàn thành

Circular Wait: Tồn tại một chuỗi tiến trình {T1, T2, ..., Tn}, trong đó:

T1 chờ tài nguyên do T2 giữ,

T2 chờ tài nguyên do T3 giữ,...

Tn lại chờ tài nguyên do T1 giữ.

**21. Liệt kê các phương pháp giải quyết Deadlock?**

Phòng ngừa Deadlock: Loại bỏ trước điều kiện gây deadlock

Tránh Deadlock: Dùng thuật toán kiểm tra trạng thái an toàn

Phát hiện Deadlock và Phục hồi: Phát hiện khi xảy ra, sau đó giải quyết

Bỏ qua Deadlock: Không xử lý, có thể bỏ qua

**22. Trình bày thuật giải đồ thị cấp phát tài nguyên (Resource-Allocation-Graph Algorithm) ?**

Resource-Allocation Graph (RAG) là một cách biểu diễn trực quan mối quan hệ giữa tiến trình và tài nguyên trong hệ thống để phát hiện hoặc tránh deadlock​

Cấu trúc đồ thị:

Đỉnh (Nodes):

+ P\_i: Tiến trình (hình tròn).

+ R\_j: Tài nguyên (hình vuông).

Cạnh (Edges):

+ Cạnh yêu cầu (Request Edge): từ tiến trình Pi → tài nguyên Rj. (Pi yêu cầu Rj)

+ Cạnh cấp phát (Assignment Edge): từ tài nguyên Rj → tiến trình Pi. (Rj cấp phát cho Pi)

Nguyên tắc thuật toán:

Không có chu trình (Cycle) trong đồ thị → Không có deadlock.

Có chu trình (Cycle):

+ Nếu mỗi tài nguyên chỉ có 1 instance → Chu trình đồng nghĩa với deadlock.

+ Nếu tài nguyên có nhiều instance → Chu trình có thể có deadlock hoặc không.

**23. Trình bày khái niệm Dynamic loading và Dynamic linking?**

|  |  |  |
| --- | --- | --- |
| **Khái niệm** | **Dynamic Loading** | **Dynamic Linking** |
| **Định nghĩa** | Dynamic Loading là kỹ thuật chỉ nạp (load) các đoạn mã chương trình vào bộ nhớ khi thực sự cần thiết trong quá trình thực thi​ | Dynamic Linking là quá trình liên kết các thư viện ngoài (shared libraries) vào chương trình tại thời điểm chạy (runtime) thay vì liên kết tĩnh lúc biên dịch​ |
| **Lợi ích** | Tiết kiệm bộ nhớ lúc đầu | Tiết kiện không gian lưu trữ, dễ cập nhật |
| **Thời gian thực hiện** | Khi gọi mã đến thực thi | Khi chương trình cần dùng thư viện trong thời gian thực |

**24. Phân mảnh (Fragmentation) bộ nhớ là gì? Phân biệt External fragmentation và Internal fragmentation?**

Phân mảnh bộ nhớ là hiện tượng không gian bộ nhớ bị lãng phí do cách cấp phát bộ nhớ không hiệu quả, dẫn đến không thể sử dụng hết dung lượng khả dụng, dù tổng bộ nhớ còn đủ​

|  |  |  |
| --- | --- | --- |
| **Loại phân mảnh** | **External** | **Internal** |
| **Định nghĩa** | Bộ nhớ còn trống rải rác thành nhiều mảnh nhỏ → không ghép thành khối lớn được. | Bộ nhớ được cấp phát lớn hơn yêu cầu thực tế → lãng phí bên trong khối cấp phát. |
| **Nguyên nhân** | Do cấp phát và giải phóng bộ nhớ động (liên tục tạo và xóa tiến trình). | Do hệ thống cấp phát bộ nhớ cố định (ví dụ: khối bộ nhớ chia đều). |
| **Biểu hiện** | Có nhiều vùng trống nhỏ nằm xen kẽ giữa các vùng đã dùng. | Bên trong mỗi vùng cấp phát, có phần dư thừa không sử dụng. |
| **Giải pháp** | Gộp vùng trống (Compaction) để gom bộ nhớ. | Thiết kế kích thước cấp phát phù hợp.  - Sử dụng phân trang (paging) hoặc phân đoạn (segmentation). |

**25. Kỹ thuật phân trang có giải quyết được vấn đề Internal fragmentation không? Vì sao?**

Phân trang không giải quyết Internal Fragmentation vì trang cuối cùng của tiến trình có thể không dùng hết, phần dư bên trong tiến trình vẫn có thể bị lãng phí.