A white paper with black text

AI-generated content may be incorrect.

Dưới đây là **giải chi tiết** bài toán về **priority with round-robin scheduling** với các tiến trình có thời gian burst và priority khác nhau. Chúng ta sẽ giải từng câu hỏi trong đề bài một cách chi tiết:

**Các thông số bài toán:**

* **Time quantum = 10** (tức mỗi tiến trình sẽ chạy tối đa 10 đơn vị thời gian trong mỗi lượt).
* **Tất cả tiến trình đều đến tại thời gian t = 0**.
* **Priority**: Số nhỏ hơn đại diện cho **ưu tiên cao hơn**.
* **Burst Time** của các tiến trình:
  + **P1**: 20, Priority = 4.
  + **P2**: 25, Priority = 3.
  + **P3**: 25, Priority = 1.
  + **P4**: 15, Priority = 5.
  + **P5**: 20, Priority = 5.

**Bước 1: Lập lịch các tiến trình theo thuật toán Priority với Round-Robin**

Chúng ta sẽ **lập lịch theo ưu tiên** trước, và khi các tiến trình có cùng **ưu tiên**, chúng ta sẽ sử dụng **Round-Robin**.

**Thứ tự ưu tiên** là:

1. **P3** (Priority = 1, ưu tiên cao nhất).
2. **P2** (Priority = 3).
3. **P1** (Priority = 4).
4. **P4 và P5** (Priority = 5, ưu tiên thấp nhất).

**Các bước thực hiện:**

1. **P3** (Priority = 1) sẽ được chọn đầu tiên vì nó có ưu tiên cao nhất. P3 có burst time = 25, nhưng thời gian quantum là 10, nên P3 sẽ chạy 10 đơn vị thời gian. Sau khi hết quantum, P3 quay lại cuối hàng đợi.
2. Tiếp theo là **P2** (Priority = 3). P2 có burst time = 25, cũng sẽ chạy 10 đơn vị thời gian. Sau khi hết quantum, P2 quay lại cuối hàng đợi.
3. Tiếp theo là **P1** (Priority = 4). P1 có burst time = 20, sẽ chạy 10 đơn vị thời gian và quay lại cuối hàng đợi.
4. **P4 và P5** đều có priority = 5, nhưng vì round-robin, chúng sẽ chạy theo thứ tự trong hàng đợi, mỗi tiến trình chạy 10 đơn vị thời gian.

Sau một vòng quay của tất cả tiến trình, ta sẽ tiếp tục vòng tiếp theo cho đến khi tất cả tiến trình hoàn thành.

**Biểu đồ Gantt:**

Để mô phỏng chính xác thứ tự chạy của các tiến trình, chúng ta sẽ vẽ **biểu đồ Gantt**:

| P3 | P3 | P2 | P2 | P1 | P1 | P2 | P2 | P1 | P4 | P4 | P5 | P5 | P3 | P3 | P2 | P2 | P1 | P1 | P1|

0 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180

A white background with black text

AI-generated content may be incorrect.

Ở đây, biểu đồ cho thấy tiến trình nào đang chạy trong mỗi khoảng thời gian (từ trái qua phải).

**Bước 2: Tính thời gian chờ trung bình (Average Waiting Time)**

**Thời gian chờ (Waiting Time)** là thời gian mà tiến trình phải đợi trước khi được thực thi. Công thức tính thời gian chờ là:



Để tính thời gian chờ, chúng ta cần biết **Turnaround Time** của mỗi tiến trình. **Turnaround Time** là tổng thời gian mà tiến trình mất từ lúc bắt đầu đến khi hoàn thành.

**Tính Turnaround Time (TAT):**

**Công thức**:





Vì tất cả các tiến trình đến tại thời gian t = 0, **Arrival Time** của tất cả tiến trình là 0.

**Bảng tính Turnaround Time và Waiting Time:**

**A table with numbers and a few black text

AI-generated content may be incorrect.**

**Giải thích:**

* **P1**: Chạy xong sau 40 đơn vị thời gian, **TAT = 40 - 0 = 40**, và **WT = 40 - 20 = 20**.
* **P2**: Chạy xong sau 60 đơn vị thời gian, **TAT = 60 - 0 = 60**, và **WT = 60 - 25 = 35**.
* **P3**: Chạy xong sau 30 đơn vị thời gian, **TAT = 30 - 0 = 30**, và **WT = 30 - 25 = 5**.
* **P4**: Chạy xong sau 45 đơn vị thời gian, **TAT = 45 - 0 = 45**, và **WT = 45 - 15 = 30**.
* **P5**: Chạy xong sau 50 đơn vị thời gian, **TAT = 50 - 0 = 50**, và **WT = 50 - 20 = 30**.

**Tính thời gian chờ trung bình:**

A math equations with numbers

AI-generated content may be incorrect.

**Bước 3: Tính thời gian quay vòng trung bình (Average Turnaround Time)**

**Turnaround Time (TAT)** là tổng thời gian mà tiến trình mất từ lúc bắt đầu đến khi hoàn thành. Công thức tính thời gian quay vòng trung bình là:

A math equations with numbers

AI-generated content may be incorrect.

**Kết quả cuối cùng:**

1. **Biểu đồ Gantt**:

Copy

|P3|P3|P2|P2|P1|P1|P4|P4|P5|P5|P3|P3|P2|P2|P1|P1|

1. **Average Waiting Time**: 24
2. **Average Turnaround Time**: 45

**Kết luận:**

* Bài toán yêu cầu bạn thực hiện lập lịch sử dụng thuật toán **Priority with Round-Robin**.
* Sau khi vẽ **biểu đồ Gantt** và tính toán **thời gian chờ** và **thời gian quay vòng**, chúng ta có kết quả như trên.

Nếu bạn có thêm câu hỏi hoặc cần làm rõ bất kỳ phần nào, hãy cho tôi biết!

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a math test

AI-generated content may be incorrect.

A white paper with black text

AI-generated content may be incorrect.

**ChatGPT said:**

Bài toán này yêu cầu sử dụng **Banker's Algorithm** để xác định xem hệ thống có an toàn hay không và kiểm tra yêu cầu cấp tài nguyên từ tiến trình **T1**.

**Dữ liệu bài toán:**

**A screenshot of a white sheet

AI-generated content may be incorrect.**

**Các tài nguyên tổng cộng** là A = 12, B = 9, C = 12. Mỗi tiến trình có **allocation** (tài nguyên đã cấp) và **max** (tài nguyên tối đa mà tiến trình có thể yêu cầu).

**Câu hỏi 1: Is the system in a safe state?**

Để xác định hệ thống có an toàn hay không, chúng ta sẽ sử dụng **Banker's Algorithm**.

**Các bước để kiểm tra an toàn**:

1. **Tính toán Need**: Need = Max - Allocation cho mỗi tiến trình.
2. **Kiểm tra tài nguyên hiện có (Available)**:
   * Available = Total - Allocation (tổng tài nguyên - tài nguyên đã cấp).
3. **Kiểm tra nếu có tiến trình có thể hoàn thành**:
   * Một tiến trình có thể hoàn thành nếu Need <= Available (tức là tài nguyên cần thiết của tiến trình không vượt quá tài nguyên có sẵn).
4. **Chạy thuật toán kiểm tra an toàn**:
   * Nếu một tiến trình có thể hoàn thành, giải phóng tài nguyên của tiến trình đó.
   * Cập nhật lại Available với tài nguyên mà tiến trình đã giải phóng.
   * Tiếp tục kiểm tra cho các tiến trình còn lại cho đến khi tất cả hoàn thành.

**Bước 1: Tính toán Need:**

**A screenshot of a table

AI-generated content may be incorrect.**

**Bước 2: Tính toán tài nguyên Available:**

A close-up of a math problem

AI-generated content may be incorrect.**Bước 3: Kiểm tra nếu hệ thống có an toàn không:**

Chúng ta có **Available = (2, 0, 3)** và tiến hành kiểm tra từng tiến trình:

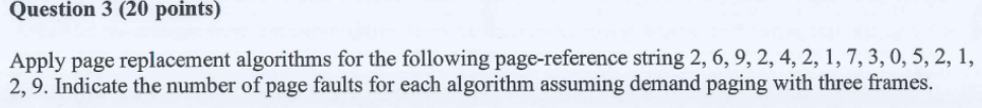
* **T1**: Need = (2, 6, 3). Vì Available = (2, 0, 3) không đủ để đáp ứng Need, nên **T1 không thể hoàn thành**.
* **T2**: Need = (4, 1, 0). Vì Available = (2, 0, 3) không đủ để đáp ứng Need, nên **T2 không thể hoàn thành**.
* **T3**: Need = (1, 0, 0). Vì Available = (2, 0, 3) đủ để đáp ứng Need, nên **T3 có thể hoàn thành**. Sau khi hoàn thành, **T3 giải phóng (5, 4, 3)**, cập nhật Available thành (7, 4, 6).

Tiếp tục với tiến trình **T1**, **T2**, và **T4** sau khi tài nguyên được giải phóng từ T3.

**Tóm tắt các bước tiếp theo** sẽ giúp bạn xác định hệ thống có an toàn hay không.

**Câu hỏi 2: Nếu tiến trình T1 yêu cầu tài nguyên (0, 0, 0), yêu cầu này có thể được cấp ngay không?**

1. **Kiểm tra yêu cầu của T1**:
   * Yêu cầu tài nguyên của T1 là (0, 0, 0), và tài nguyên **Available = (2, 0, 3)**.
   * Vì yêu cầu của T1 là không đáng kể (0, 0, 0), tài nguyên có sẵn đủ đáp ứng yêu cầu này.
2. **Kết luận**: **Yêu cầu của T1 có thể được cấp ngay** vì tài nguyên cần thiết là 0 cho mọi loại tài nguyên.



Câu hỏi này yêu cầu áp dụng các thuật toán **page replacement** để xử lý chuỗi tham chiếu trang. Bạn cần thực hiện các thuật toán thay thế trang với chuỗi tham chiếu sau:

Copy

2, 6, 9, 2, 4, 2, 1, 7, 3, 0, 5, 2, 1, 2, 9

**Các thuật toán thay thế trang (page replacement algorithms):**

1. **FIFO (First-In-First-Out)**.
2. **LRU (Least Recently Used)**.
3. **Optimal (OPT)**.

Mỗi thuật toán sẽ được áp dụng trên **3 khung trang (frames)**, và bạn cần xác định **số lượng lỗi trang (page faults)** cho từng thuật toán.

**Các bước giải quyết:**

**1. FIFO (First-In-First-Out):**

* FIFO là thuật toán thay thế trang đơn giản, nơi trang được đưa vào bộ nhớ theo thứ tự đến trước và sẽ bị thay thế theo thứ tự đó.
* Mỗi khi một trang không có trong bộ nhớ (lỗi trang), nó sẽ thay thế trang đầu tiên đã vào bộ nhớ.

**2. LRU (Least Recently Used):**

* LRU thay thế trang mà đã lâu không được truy cập.
* Mỗi khi có lỗi trang, LRU sẽ thay thế trang ít được sử dụng gần đây nhất.

**3. Optimal (OPT):**

* Thuật toán OPT chọn trang sẽ không được sử dụng lâu nhất trong tương lai để thay thế.
* Đây là thuật toán lý tưởng, tuy nhiên khó áp dụng trong thực tế vì nó yêu cầu biết trước các tham chiếu trang trong tương lai.

A close-up of a document

AI-generated content may be incorrect.

Câu hỏi này yêu cầu bạn **cài đặt một hệ thống đồng bộ** để quản lý việc qua sông của **hackers** và **employees** trong một chiếc thuyền với giới hạn chỉ chứa 4 người, và có một số điều kiện về cách thức tổ chức.

**Yêu cầu chi tiết:**

* **Thuyền** chỉ chứa tối đa **4 người** và phải luôn đầy (4 người) khi qua sông.
* **Không được để 3 employee và 1 hacker trong cùng một chiếc thuyền**, vì có thể gây nguy hiểm cho hacker.
* **Hacker** và **employee** sẽ lần lượt gọi các phương thức hackerArrives và employeeArrives khi họ đến bờ sông.
* Các phương thức này sẽ tổ chức các hacker và employees thành các nhóm an toàn để qua sông.

**Giải quyết vấn đề:**

Để giải quyết bài toán này, ta sẽ sử dụng cơ chế đồng bộ trong Java (ví dụ wait(), notify()) để điều khiển tiến trình của các hacker và employees khi họ đến bờ sông. Cần phải đảm bảo rằng khi một nhóm đã đủ (tức là 4 người), họ sẽ được cho qua sông, đồng thời tránh tình trạng ba employee và một hacker trên cùng một thuyền.

**Chiến lược:**

1. **Sử dụng hai phương thức hackerArrives và employeeArrives** để các hacker và employee có thể "đến" bờ sông và chờ đợi thuyền đầy.
2. **Kiểm tra điều kiện an toàn**: Tránh trường hợp có 3 employee và 1 hacker trên cùng một thuyền.
3. **Đồng bộ hóa**: Chỉ khi thuyền đầy đủ, các người sẽ được cho qua sông.