A table of tasks with text

AI-generated content may be incorrect.

Để giải bài toán Preemptive Priority Scheduling một cách chi tiết và chính xác, chúng ta sẽ đi qua từng bước tính toán cụ thể cho **biểu đồ Gantt**, **thời gian chờ trung bình**, và **thời gian hoàn thành trung bình**.

**1. Đầu tiên, mô phỏng và vẽ sơ đồ Gantt (Gantt Chart)**

**Thông tin về các tiến trình:**

| **Process** | **Arrival time** | **Burst time** | **Priority** |
| --- | --- | --- | --- |
| P1 | 0 | 5 | 3 |
| P2 | 1 | 3 | 2 |
| P3 | 2 | 2 | 1 |
| P4 | 3 | 4 | 4 |

**Các bước thực hiện thuật toán Preemptive Priority Scheduling:**

* **Bước 1**: Tại thời điểm 0, chỉ có **P1** đến (thời gian đến = 0). Tiến trình **P1** sẽ bắt đầu thực thi.
* **Bước 2**: Tại thời điểm 1, **P2** đến (thời gian đến = 1, ưu tiên = 2) và **P2** có ưu tiên cao hơn **P1** (P2 có ưu tiên 2 < P1 có ưu tiên 3), vì vậy **P2** sẽ **cướp CPU** của **P1** và tiếp tục thực thi.
* **Bước 3**: Tại thời điểm 2, **P3** đến (thời gian đến = 2, ưu tiên = 1). **P3** có ưu tiên cao hơn **P2**, vì vậy **P3** sẽ **cướp CPU** của **P2** và tiếp tục thực thi.
* **Bước 4**: Tại thời điểm 3, **P4** đến (thời gian đến = 3, ưu tiên = 4). Tuy nhiên, **P4** có ưu tiên thấp nhất nên phải chờ, và **P3** tiếp tục thực thi cho đến khi hoàn thành vào thời điểm 4.
* **Bước 5**: Sau khi **P3** hoàn thành, **P2** sẽ tiếp tục thực thi vì **P2** có ưu tiên cao hơn **P1** và **P4**.
* **Bước 6**: Sau khi **P2** hoàn thành, **P1** (có ưu tiên cao hơn **P4**) sẽ tiếp tục thực thi.
* **Bước 7**: Cuối cùng, **P4** sẽ bắt đầu thực thi khi các tiến trình khác đã hoàn thành.

**A white rectangular object with black text

AI-generated content may be incorrect.**

**2. Tính toán thời gian chờ trung bình (Average Waiting Time)**

**Công thức tính thời gian chờ**:

Thời gian chờ=Thời gian hoaˋn thaˋnh−Thời gian đeˆˊn−Thời gian xử lyˊ\text{Thời gian chờ} = \text{Thời gian hoàn thành} - \text{Thời gian đến} - \text{Thời gian xử lý}Thời gian chờ=Thời gian hoaˋn thaˋnh−Thời gian đeˆˊn−Thời gian xử lyˊ​

**Tính toán:**

1. **P1**:
   * **Completion Time**: 10 (P1 hoàn thành tại thời điểm 10).
   * **Waiting Time** = (Completion Time - Arrival Time) - Burst Time = (10 - 0) - 5 = 5.
2. **P2**:
   * **Completion Time**: 6 (P2 hoàn thành tại thời điểm 6).
   * **Waiting Time** = (Completion Time - Arrival Time) - Burst Time = (6 - 1) - 3 = 2.
3. **P3**:
   * **Completion Time**: 4 (P3 hoàn thành tại thời điểm 4).
   * **Waiting Time** = (Completion Time - Arrival Time) - Burst Time = (4 - 2) - 2 = 0.
4. **P4**:
   * **Completion Time**: 14 (P4 hoàn thành tại thời điểm 14).
   * **Waiting Time** = (Completion Time - Arrival Time) - Burst Time = (14 - 3) - 4 = 7.

**A math equation with black text

AI-generated content may be incorrect.**

**3. Tính toán thời gian quay vòng trung bình (Average Turnaround Time)**

**Công thức tính thời gian quay vòng**:

Thời gian quay voˋng=Completion Time−Arrival Time\text{Thời gian quay vòng} = \text{Completion Time} - \text{Arrival Time}Thời gian quay voˋng=Completion Time−Arrival Time

**Tính toán:**

1. **P1**:
   * **Turnaround Time** = Completion Time - Arrival Time = 10 - 0 = 10.
2. **P2**:
   * **Turnaround Time** = Completion Time - Arrival Time = 6 - 1 = 5.
3. **P3**:
   * **Turnaround Time** = Completion Time - Arrival Time = 4 - 2 = 2.
4. **P4**:
   * **Turnaround Time** = Completion Time - Arrival Time = 14 - 3 = 11.

**A math equation with numbers

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

1. **Thời gian chờ trung bình**: 3.5
2. **Thời gian quay vòng trung bình**: 7

**Giải thích thêm:**

* **Thuật toán Preemptive Priority Scheduling** có tính năng **ngắt tiến trình đang thực thi** khi có một tiến trình đến với **ưu tiên cao hơn**.
* Sau khi tính toán thời gian chờ và thời gian quay vòng, ta có kết quả đúng như trên.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

Tôi sẽ giải bài toán này một cách chi tiết bằng tiếng Việt, sau đó cung cấp mã Java để mô phỏng và kiểm tra kết quả. Bài toán có hai phần: một phần liên quan đến lập trình hệ thống với fork() (C programming) và một phần liên quan đến lập trình đa luồng với pthread (POSIX threads). Tôi sẽ giải từng phần một cách rõ ràng.

**Phần a: Lập trình hệ thống với fork()**

**Phân tích bài toán**

Chương trình sử dụng fork() để tạo một tiến trình con từ tiến trình cha. Chúng ta cần xác định giá trị của biến value được in tại **Line A** và **Line B**.

**Mã nguồn (C):**

A screenshot of a computer code

AI-generated content may be incorrect.

**Giải thích chi tiết**

1. **Hàm fork()**:
   * fork() tạo một tiến trình con (child process) từ tiến trình cha (parent process).
   * Sau khi gọi fork(), cả tiến trình cha và tiến trình con đều tiếp tục thực thi từ dòng mã ngay sau fork().
   * Giá trị trả về của fork():
     + Trong tiến trình cha: pid là ID của tiến trình con (một số dương).
     + Trong tiến trình con: pid là 0.
     + Nếu fork() thất bại: trả về -1.
2. **Biến value**:
   * Ban đầu, value = 10.
   * Khi fork() được gọi, tiến trình con được tạo ra và nó nhận một bản sao của không gian địa chỉ của tiến trình cha, bao gồm biến value. Tuy nhiên, sau khi fork(), tiến trình cha và tiến trình con có **bộ nhớ riêng biệt**, nên thay đổi giá trị của value trong tiến trình con không ảnh hưởng đến tiến trình cha và ngược lại.
3. **Luồng thực thi**:
   * **Tiến trình con (pid == 0)**:
     + In "welcome to EIU\n".
     + Cập nhật value += 20, nên value trong tiến trình con trở thành 10 + 20 = 30.
     + In tại **Line A**: "A. Value = 30\n".
     + Tiến trình con kết thúc.
   * **Tiến trình cha (pid > 0)**:
     + In "bye\_bye\n".
     + Cập nhật value += 15, nên value trong tiến trình cha trở thành 10 + 15 = 25.
     + In tại **Line B**: "B. Value = 25\n".
     + Gọi wait(NULL) để đợi tiến trình con kết thúc trước khi tiến trình cha kết thúc.
4. **Kết quả in**:
   * **Line A** (tiến trình con): A. Value = 30.
   * **Line B** (tiến trình cha): B. Value = 25.
5. **Lưu ý về thứ tự in**:
   * Vì tiến trình cha và tiến trình con chạy song song, thứ tự in có thể thay đổi tùy thuộc vào hệ điều hành và cách lập lịch tiến trình. Tuy nhiên, trong trường hợp lý tưởng:
     + Nếu tiến trình con chạy trước: "welcome to EIU\n" và "A. Value = 30\n" sẽ được in trước.
     + Nếu tiến trình cha chạy trước: "bye\_bye\n" và "B. Value = 25\n" sẽ được in trước.
     + Tuy nhiên, câu hỏi chỉ yêu cầu giá trị tại **Line A** và **Line B**, nên chúng ta không cần quan tâm đến thứ tự in, chỉ cần xác định giá trị của value.

**Kết luận phần a**

* **Line A**: A. Value = 30.
* **Line B**: B. Value = 25.

**Phần b: Lập trình đa luồng với pthread**

**Phân tích bài toán**

Chương trình sử dụng POSIX threads (pthread) để tạo một luồng mới. Chúng ta cần xác định giá trị của biến value được in tại **Line A** và **Line B**.

A screenshot of a computer code

AI-generated content may be incorrect.

**Giải thích chi tiết**

1. **Luồng và bộ nhớ**:
   * Trong lập trình đa luồng với pthread, tất cả các luồng trong cùng một tiến trình **chia sẻ cùng không gian địa chỉ bộ nhớ**. Điều này có nghĩa là biến value là biến toàn cục và được chia sẻ giữa luồng chính (main thread) và luồng mới được tạo (runner).
   * Bất kỳ thay đổi nào đối với value trong một luồng sẽ ảnh hưởng đến giá trị của value trong các luồng khác.
2. **Luồng thực thi**:
   * **Luồng chính (main thread)**:
     + Khởi tạo thuộc tính luồng (pthread\_attr\_init).
     + Tạo một luồng mới với hàm runner bằng pthread\_create.
     + Gọi pthread\_join để đợi luồng mới kết thúc trước khi tiếp tục thực thi.
     + Sau khi luồng mới kết thúc, cập nhật value += 15.
     + In tại **Line B**.
   * **Luồng mới (runner)**:
     + Cập nhật value += 20.
     + In tại **Line A**.
     + Kết thúc luồng bằng pthread\_exit(0).
3. **Thứ tự thực thi**:
   * pthread\_join đảm bảo rằng luồng chính sẽ đợi luồng mới (runner) hoàn thành trước khi tiếp tục. Do đó:
     + Đầu tiên, luồng runner chạy:
       - Ban đầu, value = 10.
       - value += 20, nên value trở thành 10 + 20 = 30.
       - In tại **Line A**: "A. Value = 30\n".
     + Sau khi luồng runner kết thúc, luồng chính tiếp tục:
       - value += 15, nên value trở thành 30 + 15 = 45.
       - In tại **Line B**: "B. Value = 45\n".
4. **Kết quả in**:
   * **Line A** (luồng runner): A. Value = 30.
   * **Line B** (luồng chính): B. Value = 45.

**Kết luận phần b**

* **Line A**: A. Value = 30.
* **Line B**: B. Value = 45.

**Tóm tắt kết quả**

* **Phần a (fork)**:
  + **Line A**: A. Value = 30.
  + **Line B**: B. Value = 25.
* **Phần b (pthread)**:
  + **Line A**: A. Value = 30.
  + **Line B**: B. Value = 45.

A white paper with black text

AI-generated content may be incorrect.

A screenshot of a white sheet with black text

AI-generated content may be incorrect. A screenshot of a math problem

AI-generated content may be incorrect.

A screenshot of a math test

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A white paper with black text

AI-generated content may be incorrect.

A screenshot of a white page

AI-generated content may be incorrect.

A screenshot of a math test

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.