**Đại học Bách khoa Hà Nội**

**Trường CNTT&TT**

**BÁO CÁO BÀI TẬP LỚN**

**MÔN: NGUYÊN LÝ HỆ ĐIỀU HÀNH**

***Chủ đề 04: Xây dựng chương trình minh họa giải thuật lập lịch hồi tiếp đa mức.***

**Họ và tên:** Nguyễn Trung Hiếu - 20204551

Trịnh Quang Quân - 20200511

Nguyễn Thị Khánh Linh - 20204840

**Mã lớp:** 132637

**Giáo viên hướng dẫn:** Đỗ Quốc Huy

**Mục lục**

|  | Trang |
| --- | --- |
| **Phần 1**: Lời mở đầu…………………………………………… | 2 |
| **Phần 2**: Giới thiệu đề tài………………………………………. | 3 |
| 2.1 Giới thiệu chung…………………………………….. |  |
| 2.2 Một số khái niệm…………………………………… |  |
| **Phần 3**: Nội dung chính……………………………………….. | 7 |
| 3.1. Giải thuật điều phối FCFS………………………….. | 8 |
| 3.2. Giải thuật điều phối RR ……………………………. | 8 |
| 3.3 Giải thuật điều phối hàng đợi đa mức………………. | 8 |
| 3.4 Giải thuật điều phối hàng đợi hồi tiếp đa mức……... | 8 |
| 3.5Mô tả bài toán…………………………………………… |  |
| **Phần 4**: Miêu tả chương trình…………………………………. | 10 |
| **Phần 5**: Tài liệu tham khảo …………………………………… | 23 |

**Phần 1: Lời mở đầu**

Nguyên lý hệ điều hành là một môn học bổ ích giúp sinh viên chúng em hiểu biết được cơ cấu tổ chức, cũng như việc quản lý, điều phối các tiến trình của hệ thống máy tính. Qua đó hiểu biết phần nào về phần mềm cơ bản nhất của máy tính là hệ điều hành. Việc nghiên cứu, hoàn thành đồ án nguyên lý hệ điều hành cũng giúp chúng em được hiểu rõ hơn nữa về hệ điều hành cơ bản.

Chúng em xin chân thành cảm ơn sự hướng dẫn của thầy Đỗ Quốc Huy, đã tận tình chỉ dẫn giúp chúng em hoàn thành được đề tài báo cáo này.

**Phần 2:Giới thiệu đề tài**

**2.1 Giới thiệu chung**

Các hệ điều hành đa nhiệm cho phép chạy nhiều chương trình cùng một lúc ,vì vậy đòi hỏi CPU phải thực hiện song song nhiều tiến trình. Điều phối CPU là nền tảng trong thiết kế hệ điều hành. Khi có nhiều tiến trình đồng thời ở trạng thái Ready thì phân phối bộ xử lý cho các tiến trình đảm người dùng luôn có cảm giác các chương trình luôn được thực hiện và các yêu cầu của người dùng luôn được phản hồi nhanh chóng. Điều phối CPU hợp lý sẽ làm tăng khả năng khai thác hệ thống và khiến các tài nguyên trong máy tính luôn bận rộn. Trong khi các giải thuật điều phối CPU như FCFS ,điều phối có ưu tiên hay sử dụng hàng đợi đa mức tồn tại vấn đề “nạn đói” thì điều phối CPU sử dụng hàng đợi hồi tiếp đa mức lại ngăn ngừa “nạn đói” vì tổ chức nhiều hàng đợi và cho phép các tiến trình di chuyển giữa các hàng đợi. Vì vậy, đề tài làm rõ các đặc điểm và cách lựa chọn tiến trình để phân phối CPU của chiến lược điều phối sử dụng hàng đợi hồi tiếp đa mức.

**2.2 Một số khái niệm**

a,Tiến trình

*Process*, hay tiến trình, là sự thực thi của một chương trình và thực hiện các hành động liên quan được chỉ định trong một chương trình, hoặc nó là một đơn vị thực thi nơi chương trình chạy. Hệ điều hành tạo, lên lịch và chấm dứt các tiến trình. Các tiến trình khác được tạo bởi tiến trình chính được gọi là tiến trình con.

Hoạt động của bất kỳ tiến trình nào cũng được kiểm soát bởi khối điều khiển tiến trình (*Process Control Block*, viết tắt là PCB). PCB chứa tất cả những thông tin quan trọng liên quan đến các tiến trình, chẳng hạn như: id tiến trình, ưu tiên, trạng thái, CPU, …

Những hoạt động xen kẽ của các tiến trình giúp nâng cao tốc độ tính toán khi hoạt động I / O trong một tiến trình trùng lặp với hoạt động tính toán trong tiến trình khác.

Tiến trình đơn luồng: Là TT thực hiện chỉ 1 luồng thực thi

Tiến trình đa luồng : Là TT có nhiều luồng thực thi

Khi thực hiện, TT thay đổi trạng thái

• **Khởi tạo (New)** TT đang được khởi tạo

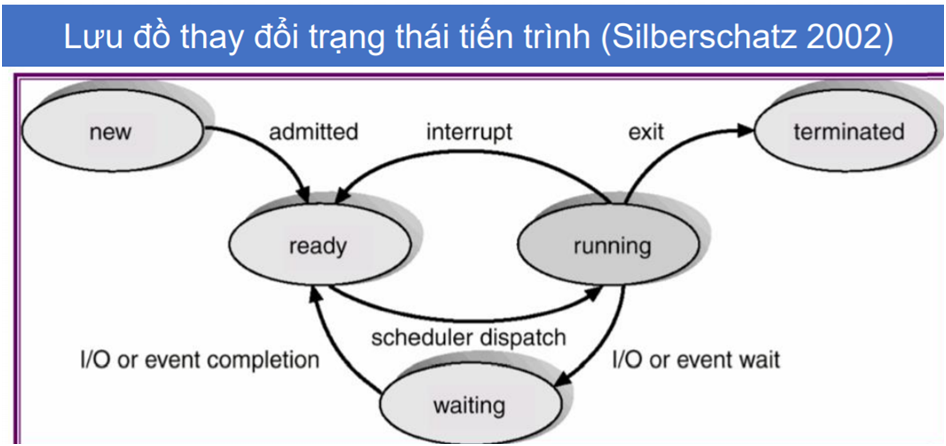
• **Sẵn sàng (Ready)** TT đang đợi sử dụng processor vật lý

• **Thực hiện (Running)** Các câu lệnh của TT đang được thực hiện

• **Chờ đợi (Waiting)** TT đang chờ đợi 1 sự kiện nào đó xuất hiện (sự hoàn thành thao tác vào/ra)

• **Kết thúc (Terminated)** TT thực hiện xong

*Trạng thái của TT là một phần trong hoạt động hiện tại của TT*

**

b, Luồng

Luồng (tiếng Anh: thread) thực thi là một chuỗi các lệnh được lập trình nhỏ nhất để có thể được quản quản lý độc lập bởi một bộ định thời (scheduler), thường là một phần của hệ điều hành. Việc hiện thực các luồng và tiến trình (process) khác nhau giữa các hệ điều hành, nhưng trong hầu hết các trường hợp, luồng là một thành phần của tiến trình. Đa luồng có thể tồn tại trong cùng một tiến trình, thực thi đồng thời và chia sẻ tài nguyên như bộ nhớ, trong khi những tiến trình khác nhau thì không chia sẻ các tài nguyên này. Cụ thể, các luồng trong một tiến trình chia sẻ mã thực thi và giá trị của các biến được phân bổ động và biến toàn cục không phải cục bộ của luồng tại bất kì thời điểm nào.

Trong máy tính có nhiều tiến trình ( process) và luồng (thread). Nhưng CPU thì mỗi máy chỉ có một, và thực tế đa số hệ thống khi hoạt động đều là thực hiện 1 lúc nhiều tiến trình và luồng.Để có thể điều phối tiến trình hợp lí nhất và sử dụng CPU sao cho tận dụng được tối đa năng suất của CPU,cần có 1 cơ chế để điều phối tiến trình và CPU (Scheduler).

c, Job scheduler ( Long-term scheduler)

Bộ điều phối công việc là cơ chế chọn tiến trình nào sẽ được thêm vào hàng đợi sẵn sàng (ready queue).

d, CPU scheduler (Short-term scheduler)

Điều phối CPU là cơ chế chọn tiến trình tiếp theo trong hàng đợi sẵn sàng ( ready queue) để xử lí

Có rất nhiều thuật toán điều phối CPU nhưng thuật toán tổng hợp nhất là Điều phối hàng đợi hồi tiếp đa mức (Multilevel Feedback Queue )

**Phần 3: Nội dung chính**

**3.1 Giải thuật điều phối FCFS (First come - First serve)**

Theo giải thuật FCFS, tiến trình nào tới trước sẽ được sẽ được quyền sử dụng CPU trước cho tới khi kết thúc tiến trình hoặc chờ đợi vào ra mà không có sự trưng dụng CPU. Giải thuật tuy đơn giản, dễ thực hiện nhưng có thể gây ra sự chờ đợi lâu dài nếu các tiến trình ngắn đến sau tiến trình dài.

**3.2 Giải thuật điều phối RR( Round Robin Scheduling )**

Mỗi tiến trình sẽ được cấp một lượng tử thời gian để sử dụng CPU. Sau khi hết thời gian mà tiến trình vẫn chưa thực hiện xong, nó sẽ bị trưng dụng CPU và đưa vào cuối hàng đợi sẵn sàng.Giải thuật mang lại sự cân bằng nhất về thời gian phản hồi giữa các tiến trình vì được luân phiên sử dụng CPU.

**3.3 Giải thuật điều phối hàng đợi đa mức MQL (Multilevel Queue Scheduling )**

Các tiến trình trong hàng đợi sẵn sàng được chia thành các hàng đợi riêng biệt phụ thuộc vào một số đặc điểm của tiến trình ví dụ( memory size, process priority, process type…) mỗi hàng đợi có một 1 thuật toán điều phối CPU riêng. Ví dụ cách chia phổ biến là foreground và background . 2 loại tiến trình này có thời gian phản hồi khác nhau nên cần 2 thuật toán điều phối riêng(VD Foreground dùng RR và background dùng FCFS) . Ngoài ra hàng đợi foreground có độ ưu tiên cao hơn background. Tiến trình chỉ nằm cố định trong 1 hàng đợi. Ví dụ chúng ta có 5 hàng đợi với mức độ ưu tiên từ cao theo STT:

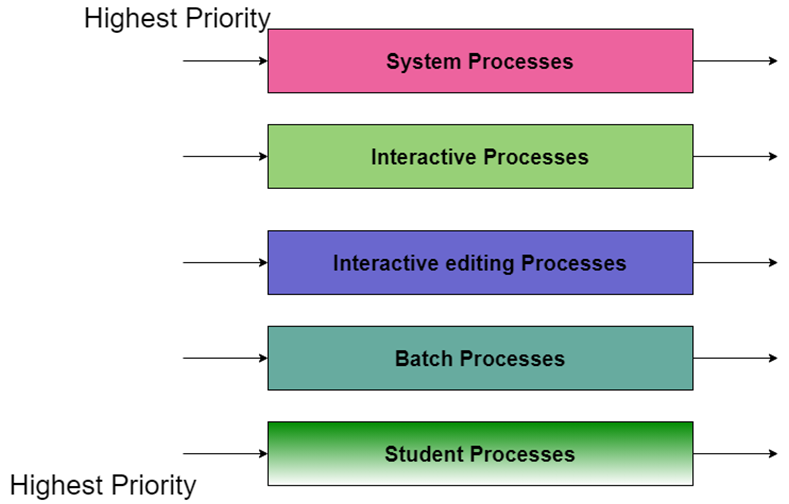
1. System Processes

2. Interactive Processes

3. Interactive Editing Processes

4. Batch Processes

5. Student Processes



Tiến trình trong hàng đợi chỉ được xử lí khi những hàng đợi có độ ưu tiên cao hơn nó đều trống. Ví dụ tiến trình trong hàng đợi Batch process chỉ được xử lí khi 3 hàng đợi System, Interactive, và Interactive editing đều trống. Giả sử tiến trình trong hàng đợi Batch process đang được xử lý mà có một tiến trình vào hàng đợi Interactive editing thì CPU sẽ bị trưng dụng cho tiến trình vừa vào.

Ưu điểm của hàng đợi đa mức (Multilevel Queue Scheduling (MLQ))

Có thể sử dụng đa dạng thuật toán điều phối CPU hợp lý cho mỗi hàng chờ VD:

**For System Processes**: First Come First Serve(FCFS) Scheduling.

**For Interactive Processes**: Shortest Job First (SJF) Scheduling.

**For Batch Processes**: Round Robin(RR) Scheduling

**For Student Processes**: Priority Scheduling

Nhược điểm của hàng đợi đa mức.

Xảy ra hiện tượng “đói CPU” cho các tiến trình ở các hàng đợi có mức độ ưu tiên thấp. Chúng không bao giờ được thực thi hoặc phải đợi 1 khoảng thời gian dài vì các tiến trình ở hàng đợi ưu tiên cao chiếm một khoảng thời gian lớn .

**3.4 Hàng đợi hồi tiếp đa mức (Multilevel Feedback Queue Scheduling)**

Trong thuật toán điều phối hàng đợi đa mức , các tiến trình được chia cố định vào các hàng đợi và không thể di chuyển giữa các hàng đợi. Tuy rằng chi phí lên lịch thấp nhưng nhược điểm là không lịch hoạt. Vì vậy, hàng đợi hồi tiếp đa mức cho phép các tiến trình có thể di chuyển giữa các hàng đợi. Phân chia tiến trình theo đặc điểm xử dụng vi xử lý.

-Nếu một tiến trình xử lý mất quá nhiều thời gian, nó sẽ được chuyển tới hàng đợi có độ ưu tiên thấp hơn.

-Nếu một tiến trình đợi quá lâu ở hàng đợi ưu tiên thấp, nó sẽ được chuyển lên hàng đợi có mức độ ưu tiên cao hơn.

Định nghĩa của bộ lập lịch hàng đợi phản hồi đa cấp làm cho nó trở thành thuật toán lập lịch CPU chung nhất. Nó có thể được cấu hình để phù hợp với một hệ thống cụ thể đang được thiết kế. Thật không may, nó cũng yêu cầu một số phương tiện chọn giá trị cho tất cả các tham số để xác định bộ lập lịch tốt nhất. Mặc dù hàng đợi phản hồi đa cấp là sơ đồ chung nhất, nhưng nó cũng là sơ đồ phức tạp nhất.

Nói chung, bộ lập lịch hàng đợi phản hồi đa cấp được xác định bởi các tham số sau:

1.Số lượng hàng đợi.

2.Thuật toán lập lịch cho mỗi hàng đợi.

3.Phương pháp được sử dụng để xác định thời điểm chuyển tiến trình lên hàng đợi có mức độ ưu tiên cao hơn.

4.Phương pháp được sử dụng để xác định thời điểm chuyển một tiến trình xuống hàng đợi có mức độ ưu tiên thấp hơn.

5.Phương pháp được sử dụng để xác định hàng đợi mà một tiến trình sẽ vào khi quy trình đó cần được xử lý.

**3.5 Mô tả bài toán:**

-Xây dựng chương trình minh họa giải thuật lập lịch hồi tiếp đa mức((Multilevel Feedback Queue Scheduling). Yêu cầu đặt ra là phân phối CPU cho các tiến trình sao cho:

- Có các tiến trình với thời gian vào và thời gian xử lý riêng vào hệ thống

- Có các hàng đợi và mức độ ưu tiên.

-Chỉ có 1 tiến trình sử dụng CPU tại 1 thời điểm

-Các tiến trình ở hàng đợi ưu tiên thấp không được phân phối CPU trừ khi hàng đợi ưu tiên cao hơn nó trống

-Nếu một tiến trình xử lý mất quá nhiều thời gian thì chuyển nó xuống hàng đợi có mức ưu tiên thấp hơn.

**Phần 4: Miêu tả chương trình**

**Lời giải đã lựa chọn:**

Xây dựng chương trình có 3 hàng đợi với 3 mức độ ưu tiên lần lượt là 1, 2, 3. Các tiến trình trên cùng 1 hàng đợi có cùng mức độ ưu tiên. Khi tiến trình đang chạy nghĩa là nó đang được phân phối CPU.

\* Queue 1 sử dụng chiến lược điều phối RR với lượng tử thời gian là 5.

\* Queue 2 sử dụng chiến lược điều phối RR với lượng tử thời gian là 8.

\* Queue 3 sử dụng chiến lược điều phối FCFS.

\* Khi 1 tiến trình xuất hiện sẽ đi vào hàng đợi Queue 1

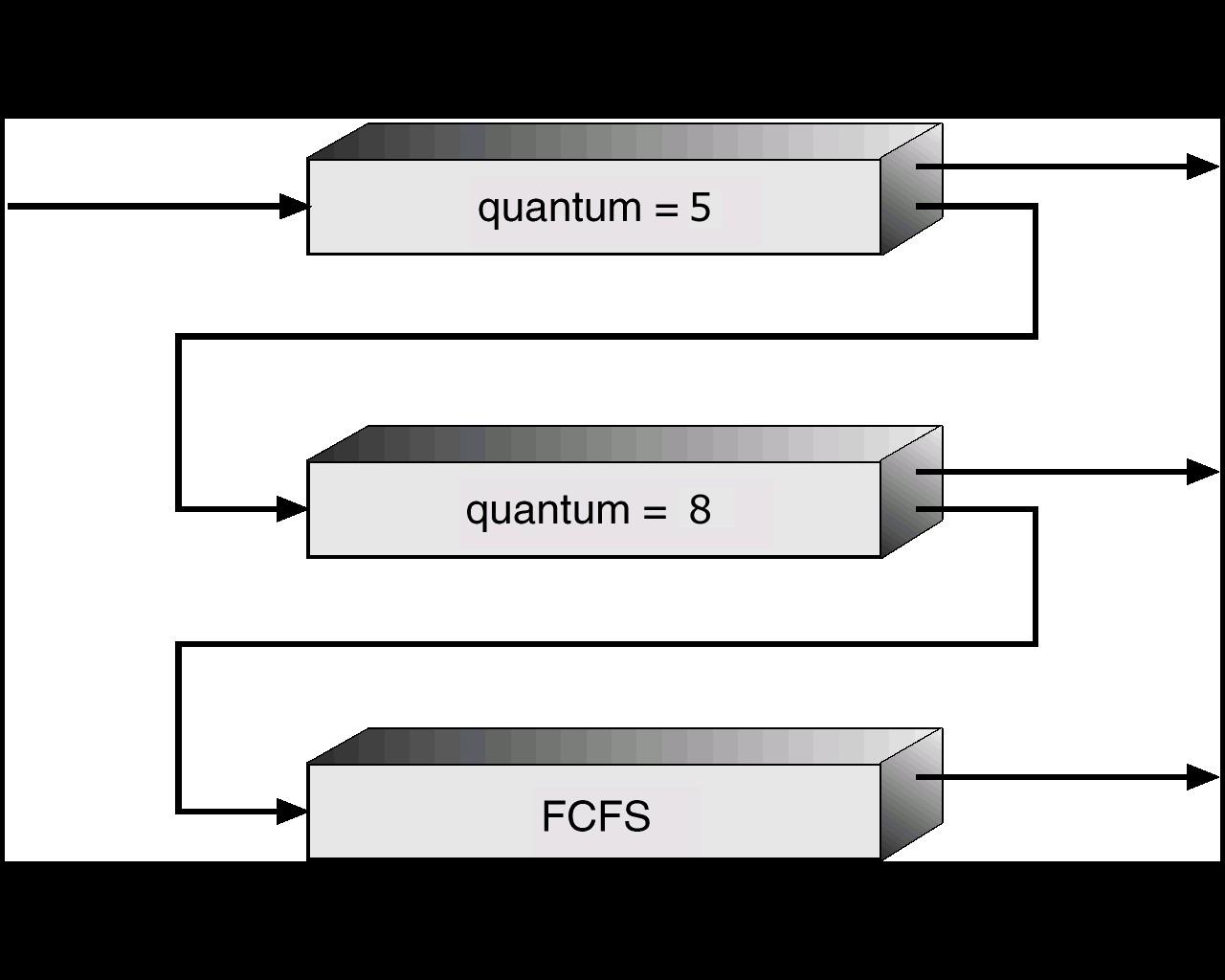
\* Nếu hết lượng tử thời gian dùng CPU ở Queue 1 mà P1 vẫn chưa thực hiện xong thì đẩy nó xuống Queue 2.

\* Nếu hết lượng tử thời gian dùng CPU ở Queue 2 mà P1 vẫn chưa thực hiện xong thì đẩy nó xuống Queue 3.

\* Thực hiện các tiến trình theo mức độ ưu tiên và thứ tự trong hàng đợi.

\* Khi một tiến trình ở hàng ưu tiên thấp đang chạy mà có tiến trình đẩy vào hàng đợi ưu tiên cao hơn thì nó sẽ bị ngắt và chuyển CPU cho tiến trình vừa vào.

\* Khi tiến trình ở hàng đợi ưu tiên thấp bị đói thì sẽ được chuyển mức hàng đợi

**Miêu tả chương trình bằng code.**

**Nội dung chính:** Chương trình sử dụng ngôn ngữ lập trình C++, mô hình lập trình hướng đối tượng

**Các lớp đối tượng của chương trình:**Chương trình có 2 class Process và CPU

**Class Process:**

Code:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**class Process{**

**private:**

**char name;**

**int AT; // Arrive time;**

**int BT; // brust time;**

**int Last\_Ptime; // time cuoi duoc chiem dung cpu**

**int RT; // time remain**

**public:**

**Process(const char &Name,const int &aT, const int &bT)**

**: name(Name), AT(aT), BT(bT), Last\_Ptime(aT), RT(bT)**

**{}**

**Process()**

**{}**

**char getName(){**

**return name;**

**}**

**void setRT(){**

**RT--;**

**}**

**int getRT(){**

**return RT;**

**}**

**void setLast\_PT(int x){**

**Last\_Ptime=x;**

**}**

**int getLast\_PT(){**

**return Last\_Ptime;**

**}**

**int getAT(){**

**return AT;**

**}**

**};**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Chi tiết class:

Thuộc tính của lớp Process

* Name: Tên của tiến trình thuộc kiểu char(private)
* AT : Thời điểm tiến trình xuất hiện thuộc kiểu int(private)
* BT: Thời gian cần xử lí tiến trình thuộc kiểu int(private)
* RT: Thời gian còn lại của tiến trình cần được xử lí thuộc kiểu int(private)
* Last\_Ptime: Thời điểm cuối cùng mà tiến trình được cấp CPU thuộc kiểu int(private)

Phương thức chính của lớp Process

* public void setLastP\_time(int x): Chức năng chính là đặt lại thuộc tính Last\_Ptime bằng x
* public void setRT(): Chức năng chính là giảm thời gian cần xử lí tiến trình xuống 1 đơn vị.

**Class CPU:**

Code:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**class CPU{**

**private:**

**Process\* P\_Handle; // tien trinh dang duoc xu li boi CPU;**

**int QUEUE\_P; // hang doi CPU dang lm viec 1/2/3**

**int Time; // Thoi gian tien trinh chiem dung cpu**

**public:**

**CPU(){}**

**void Processing(){**

**P\_Handle->setRT();**

**Time++;**

**}**

**bool check\_Finish(){**

**if(P\_Handle==NULL)**

**return false;**

**if(P\_Handle->getRT()==0)**

**return true;**

**else return false;**

**}**

**bool check\_Time(){ // kiem tra xem tien trinh con duoc quyen chiem dung CPU khong**

**switch (QUEUE\_P)**

**{**

**case 1: { if(Time==5) return false;**

**else return true;}**

**case 2: { if(Time==8) return false;**

**else return true;}**

**default: return true;**

**}**

**}**

**Process\* getP\_Handle(){**

**return P\_Handle;**

**}**

**void setP\_Handle(){**

**P\_Handle=NULL;**

**Time=0;**

**QUEUE\_P=0;**

**}**

**void setP\_Handle(Process\* P , int Q){**

**P\_Handle=P;**

**QUEUE\_P=Q;**

**Time=0;**

**}**

**int getQueue\_P(){**

**return QUEUE\_P;**

**}**

**};**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Chi tiết class:

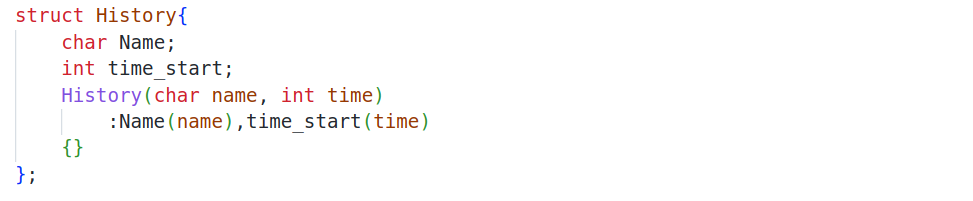
Thuộc tính của lớp CPU

* P\_Handle: Là tiến trình CPU đang xử lý thuộc lớp Process nêu CPU rảnh thì bằng NULL(private)
* QUEUE\_P: là Hàng đợi CPU đang làm việc có giá tri từ 0 ->3 thuộc kiểu int(private)
* Time: Là thời gian CPU xử lý tiến trình P\_Handle thuộc kiểu int(private)

Phương thức chính của lớp Process

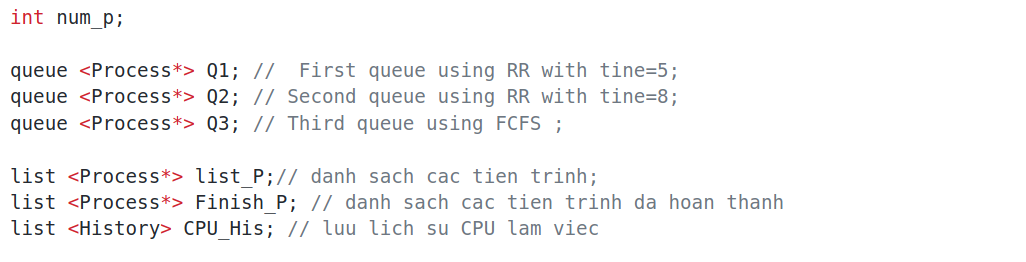
* public void Processing(): Chức năng chính là giảm RT của P\_Handle đi 1 đơn vị và tăng Time(thời gian P\_Handle chiếm dụng CPU) lên 1 đơn vị
* public bool check\_Finish(): Chức năng chính là kiểm tra tiến trình đang chiếm dụng CPU nếu P\_Handle được xử lý xong và có thể kết thúc thì trả về true, ngược lại trả về false.
* public bool check\_Time(): Chức năng chính là kiểm tra tiến trình P\_Handle nếu P\_Handle còn thời gian được quyền chiếm dụng CPU thì trả về true , ngược lại trả về fasle

**Các kiểu cấu trúc và biến toàn cục của chương trình:**

**Struct History**

Là biến cấu trúc dùng để lưu trữ lịch sử làm việc của CPU bao gồm tên tiến trình được xử lý và thời điểm bắt đầu xử lý

**Biến toàn cục:**

* num\_p: là biến lưu số lương tiến trình.
* Q1: là hàng đợi thứ nhất thuộc kiểu queue <Process\*>của thư viện STL C++.
* Q2: là hàng đợi thứ nhất thuộc kiểu queue <Process\*>của thư viện STL C++.
* Q3: là hàng đợi thứ nhất thuộc kiểu queue <Process\*>của thư viện STL C++.
* list\_P: là biến kiểu list của thư viện STL C++ dùng để lưu danh sách các tiến trình đầu vào.
* Finish\_P: là biến kiểu list của thư viện STL C++ dùng để lưu danh sách các tiến trình đã hoàn thành.
* CPU\_His:là biến kiểu list của thư viện STL C++ dùng để lưu lịch sử làm việc của CPU.

**Các hàm chính của chương trình:**

**Hàm void input()**

**void input(){**

**cout<<" Nhap so luong tien trinh:";cin>>num\_p;**

**char c='A';**

**for(int i=0;i<num\_p;i++,c++){**

**int AT,BT;**

**cout<<"Nhap arrive time cua trinh "<<c<<": ";cin>>AT;**

**cout<<"Nhap Brust time cua trinh "<<c<<": ";cin>>BT;**

**Process\* temp= new Process(c,AT,BT);**

**list\_P.push\_back(temp);**

**}**

Chức năng chính : Nhận dữ liệu từ người dùng

**Hàm Main:**

**Các Module chính của hàm main:**

* Biến int iTime :dùng để làm bộ đếm thời gian cho chương trình
* Đầu tiên sắp xếp các danh sách tiến trình theo thứ tự thời gian xuất hiện của tiến trình

**list\_P.sort([](Process\* A,Process \*B){ // sap sep theo thu tu thoi gian den**

**return A->getAT()<B->getAT();**

**});**

* Tiếp theo là vòng lặp while()- lặp cho tới khi số lượng tiến trình đã hoàn thành bằng số lượng tiến trình đã khai báo đầu vào

**while (!(Finish\_P.size()==num\_p))**

Vòng lặp while() trên sẽ thực hiện lần lượt các công việc sau:

* **Kiểm tra các tiến trình trong danh sách các tiến trình, nếu tiến trình có thời gian xuất hiện bằng thời điểm hiện tại thì đưa vào hàng đợi 1**

**while(list\_P.size()!=0 &&list\_P.front()->getAT()==iTime){**

**Q1.push(list\_P.front());**

**out\_put=out\_put+"Arrive :"+list\_P.front()->getName()+" to Q1\n";**

**list\_P.pop\_front();**

**}**

* **Kiểm tra và chuyển mức cho các tiến trình có độ ưu tiên thấp đang phải chờ đợi quá lâu(**nếu hàng đợi thứ 2 có tiến trình và tiến trình đó có khoảng thời gian từ lần cuối chiếm dụng CPU đến thời điểm hiện tại lớn hơn thời gian chờ đợi tối đa thì chuyển mức , tương tự cho hàng đợi thứ 3**)**

**while (Q2.size()!=0 && (iTime - Q2.front()->getLast\_PT())>=MAX\_WAITTIME)**

**{**

**Q2.front()->setLast\_PT(iTime);**

**out\_put=out\_put+"up level :"+Q2.front()->getName()+"\n";**

**Q1.push(Q2.front());**

**Q2.pop();**

**}  
while (Q3.size()!=0 && (iTime - Q3.front()->getLast\_PT()) >=MAX\_WAITTIME)**

**{**

**Q3.front()->setLast\_PT(iTime);**

**out\_put=out\_put+"up level :"+Q3.front()->getName()+"\n";**

**Q2.push(Q3.front());**

**Q3.pop();**

**}**

* **Kiểm tra tiến trình đang sử dụng CPU(**nếu tiến trình đang sử dụng CPU được hoàn thành thì chuyển tiến trình đó vào danh sách Finish\_P và đặt thuộc tính P\_Handle của CPU bằng NULL, hoặc nếu trình đang sử dụng CPU quá thời gian sử dụng CPU thì chuyển tiến trình xuống hàng hàng đợi thấp hơn và đặt thuộc tính P\_Handle của CPU bằng NULL, các trường hợp khác không thực hiện gì **)**

**if(cpu->check\_Finish()){**

**out\_put=out\_put+"Finish :"+ cpu->getP\_Handle()->getName()+"\n";**

**Finish\_P.push\_back(cpu->getP\_Handle());**

**cpu->setP\_Handle();**

**}else if(!cpu->check\_Time()){**

**int temp=cpu->getQueue\_P();**

**switch (temp){**

**case 1: {**

**out\_put+=out\_put+"Break "+cpu->getP\_Handle()->getName()+" out CPU to Q2\n";**

**cpu->getP\_Handle()->setLast\_PT(iTime);**

**Q2.push(cpu->getP\_Handle());cpu->setP\_Handle();**

**break;**

**}**

**case 2:{**

**out\_put=out\_put+"Break "+cpu->getP\_Handle()->getName()+" out CPU to Q3\n";**

**cpu->getP\_Handle()->setLast\_PT(iTime);**

**Q3.push(cpu->getP\_Handle());cpu->setP\_Handle();**

**break;**

**}**

**}**

**}**

* **Tiếp theo xử lý tiến trình** 
  + Nếu CPU->P\_Handle==NULL thì ta tìm kiếm tiến trình từ các hàng đợi để đưa vào, và xử lý (CPU->Processing).

**if(cpu->getP\_Handle()==NULL){ // neu cpu ranh**

**if(Q1.size()!=0){**

**out\_put=out\_put+ "CPU processing "+Q1.front()->getName()+"\n";**

**cpu->setP\_Handle(Q1.front(),1);**

**Q1.pop();**

**}else if(Q2.size()!=0){**

**out\_put=out\_put+"CPU processing "+Q2.front()->getName()+"\n";**

**cpu->setP\_Handle(Q2.front(),2);**

**Q2.pop();**

**}else if(Q3.size()!=0){**

**out\_put=out\_put+"CPU processing "+Q3.front()->getName()+"\n";**

**cpu->setP\_Handle(Q3.front(),3);**

**Q3.pop();**

**}**

**if(cpu->getP\_Handle()!=NULL){**

**cpu->Processing();**

**CPU\_His.push\_back(History(cpu->getP\_Handle()->getName(),iTime));**

* Nếu CPU->P\_Handle!=NULL thì ta kiểm tra tiến trình đang chiếm dụng CPU có bị trưng dụng bởi tiến tiến trình khác có độ ưu tiên cao hơn , nếu có thì nó sẽ bị thay thế , sau đó xử lý tiến trình (CPU->Processing)

**- Cuối cùng tăng bộ đếm lên 1 đơn vị**

***Mô tả dữ liệu đầu vào*:**

* Dòng đầu tiên là số tiến trình n
* Tiếp theo n dòng, mỗi dòng gồm 2 số là thời gian đến và thời gian xử lí (AT và BT).
* Ví dụ

3

0 23

4 22

5 90

***Mô tả dữ liệu đầu ra*:**

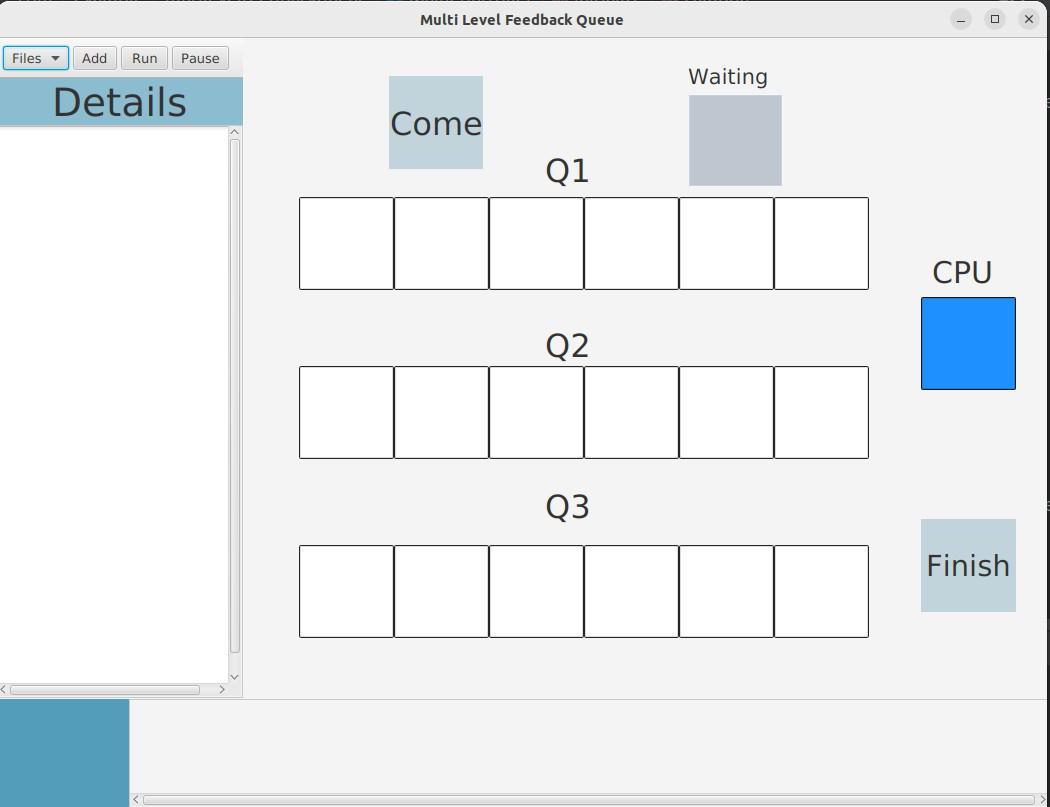
* Thời điểm và các sự kiện xảy ra ở thời điểm đó
* Biểu đồ Gantt

**Miêu tả chương trình bằng giao diện.**

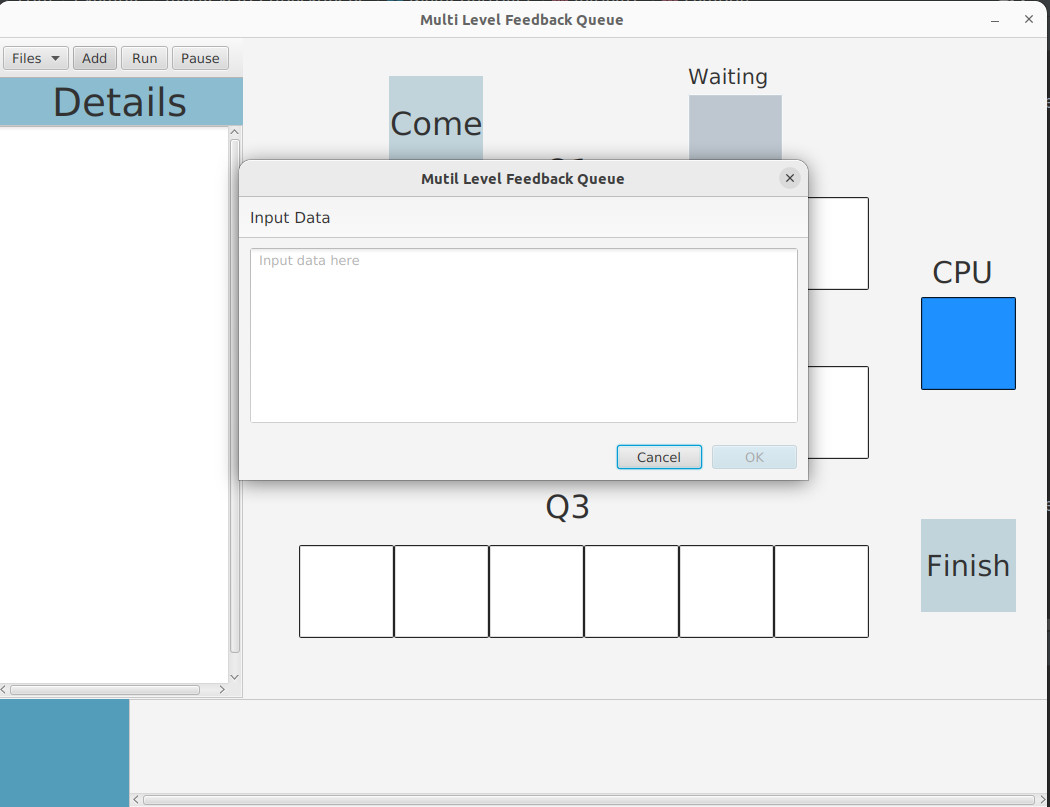
Dưới đây là giao diện mở đầu của chương trình.

Giao diện của chương trình bao gồm:

* 3 button chính(góc trái màn hình):
  + Button Add: Có chức năng nhận dữ liệu từ người dùng. Khi người dùng click chuột sẽ xuất hiện hộp thoại yêu cầu người dùng nhập dữ liệu
  + Button Run: Có chức năng chạy thuật toán mô phòng từ dữ liệu đã nhận từ người dùng
  + Button Pause : Chức năng dừng chương trình đang chạy;
* 1 khung Details: Chức năng chính là hiển thị chi tiết time line của các tiến trình vào cpu
* Khung giao diện bên trái: Nơi thực hiện mô phỏng của thuật toán
  + Come: Vị trí mặc định nơi tiến trình xuất hiện
  + Q1,Q2,Q3: Biểu diễn hàng đợi với các mức khác nhau
  + CPU: Vị trí của CPU
  + Finish: Vị trí của các tiến trình khi đã hoàn thành
  + Waitting: Vị trí các chương trình bị block/ wait



**-Hộp hội thoại yêu cầu người dùng nhập dữ liệu**



* **Một vài hình ảnh demo trong quá trình chạy chương trình**
  + Mỗi tiến trình được biểu diễn bởi 1 hình vuông, cùng với tên chương trình
  + Khi tiến trình đang đợi xư lí sẽ có trạng thái màu xanh
  + Khi tiến trình đang được CPU xử lí sẽ có trạng thái màu đỏ
  + Khi tiến trình kết thúc sẽ có trạng thái màu xám

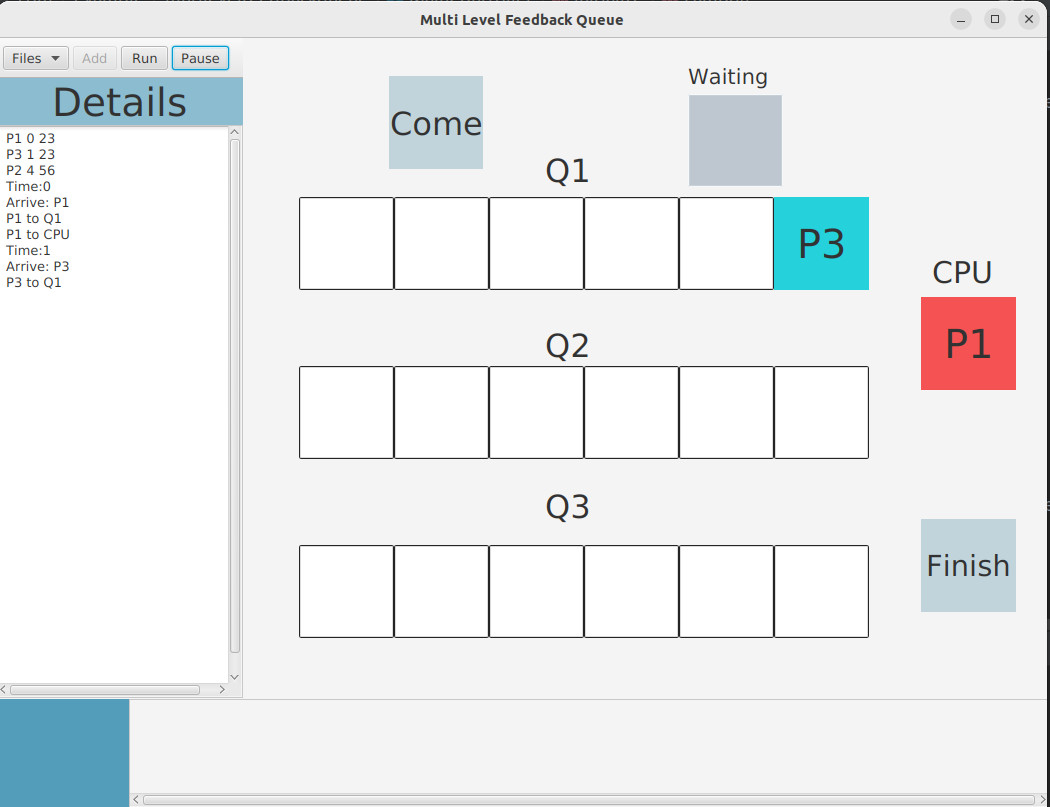
Chương trình đang hiển thị dưới đây có đầu vào là :

P1 0 23

P2 4 56

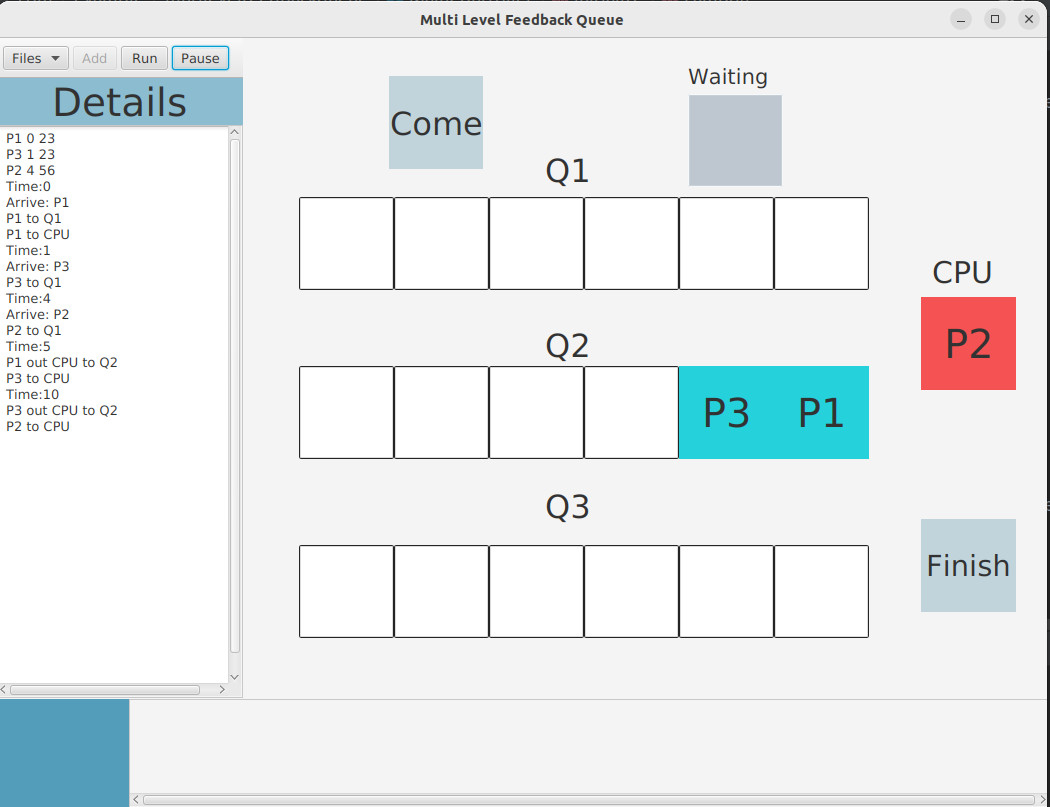
P3 1 23

Thứ tự nhập là thời gian xuất hiện và thực hiện của tiến trình.



Tại thời điểm time = 0: P1 xuất hiện được cho vào hàng đợi Q1 sau đó liền được cho vào CPU

Tại thời điểm time = 1: P3 xuất hiện được cho vào hàng đợi Q1 và đang chờ CPU xử lý P1

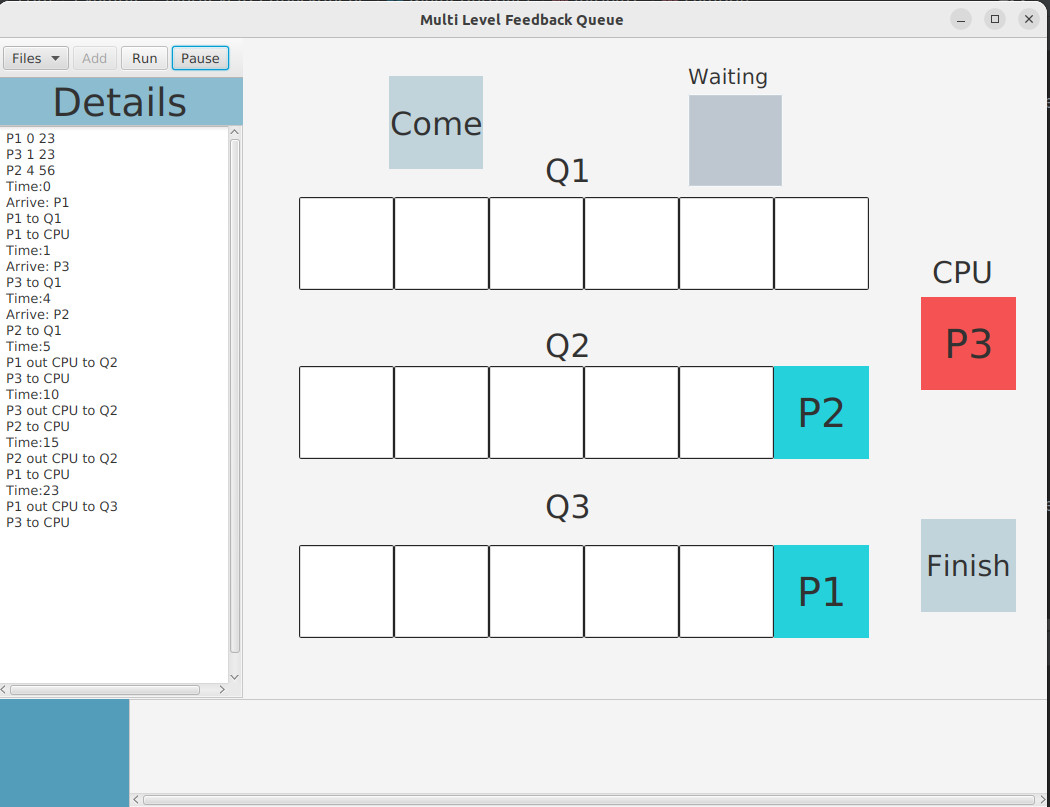


Tại thời điểm time = 4: P2 xuất hiện được cho vào hàng đợi Q1

Tại thời điểm time = 5: P1 tới Q2 (hết lượng tử thời gian dùng CPU ở Queue 1 nhưng chưa xong) và P3 tới CPU

Tại thời điểm time = 10: P3 tới Q2 và P2 tới CPU

Tại thời điểm time = 15: P2 tới Q2 và P1 tới CPU

Tại thời điểm time = 23: P1 tới Q3(hết lượng tử thời gian dùng CPU ở Queue 2) và P3 tới CPU

Tại thời điểm time = 31: P3 tới Q3 và P2 tới CPU

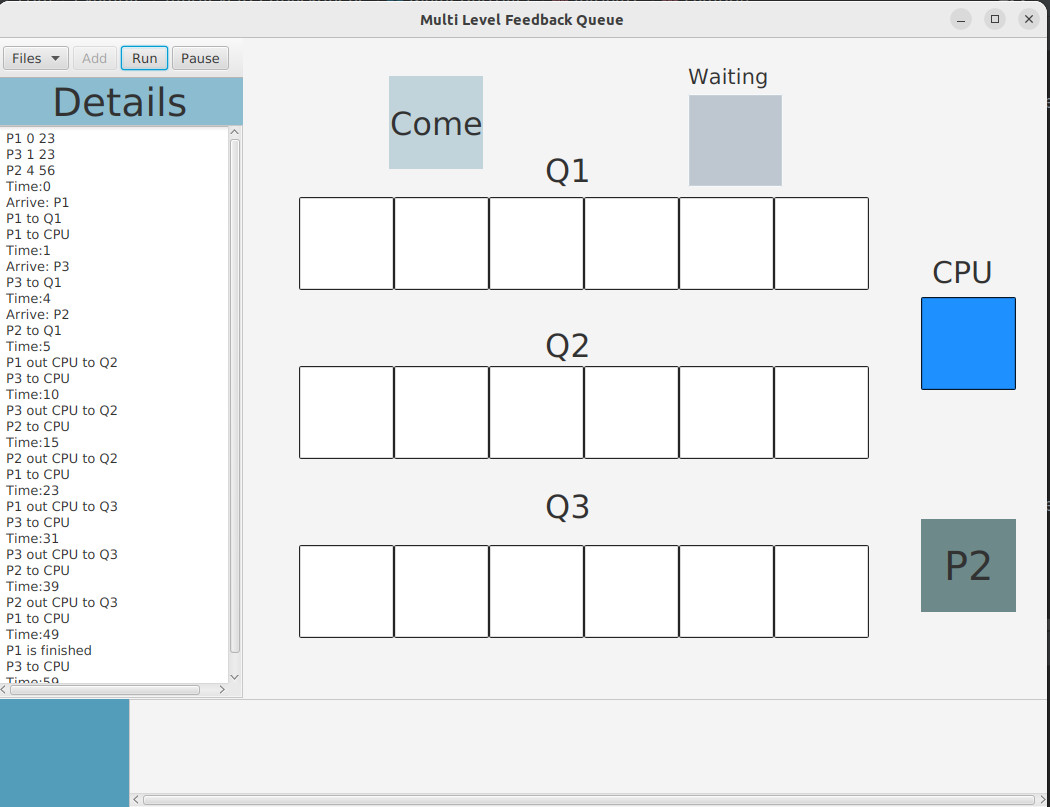
Tại thời điểm time = 39: P2 tới Q3 và P1 tới CPU

Tại thời điểm time = 49: P1 đã được xử lý xong và P1 tới Finish, P3 tới CPU

Tại thời điểm time = 59: P3 đã được xử lý xong và P3 tới Finish, P2 tới CPU

Tại thời điểm time = 102:P2 đã được xử lý xong và P2 tới Finish

Tất cả tiến trình được xử lý xong.

**Phần 6: Tài liệu tham khảo** 

1. Slide bài giảng Hệ Điều Hành - Đại học Bách khoa Hà Nội
2. [Multilevel feedback queue - Wikipedia](https://en.wikipedia.org/wiki/Multilevel_feedback_queue)
3. [Multilevel Feedback Queue Scheduling (MLFQ) CPU Scheduling - GeeksforGeeks](https://www.geeksforgeeks.org/multilevel-feedback-queue-scheduling-mlfq-cpu-scheduling/)