TRƯỜNG ĐẠI HỌC BÁCH KHOA ĐÀ NẴNG

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN

CƠ SỞ NGÀNH MẠNG

Đề tài 1: *Xây dựng chương trình mô phỏng thuật toán lập lịch multiple-processor của CPU*

Đề tài 2: *: Xây dựng ứng dụng movie server sử dụng sockets*

Giảng viên hướng dẫn: ThS. Lê Trần Đức

Sinh viên thực hiện: Võ Đức Phong

Mã số sinh viên: 102160058

Nhóm học phần: 16.11A

Lớp sinh hoạt: 16T1

Đà Nẵng,11/10/2019

LỜI MỞ ĐẦU

Lập trình mạng là một kiến thức chuyên ngành quan trọng của lĩnh vực công nghệ thông tin. Có thể xem đây là một hướng đi rất rộng mở cho sinh viên và việc nắm bắt những kĩ thuật cơ bản của nó cực kì cần thiết và quan trọng. Sau một loạt các môn học lí thuyết nền tảng như Mạng máy tính, Lập trình Java, Lập trình mạng, thì đồ án lập trình mạng này chính là hội tụ hiện thực các kiến thức học trên sách vở. Nội dung của đồ án chủ yếu thực hiện việc lập trình Socket trên họ giao thức TCP/IP và giao thức UDP trên các ứng dụng viết bằng ngôn ngữ Java, kết hợp các kĩ thuật lập trình đa luồng, lập trình web JSP để thực hiện các hạng mục của học phần.

Thông qua việc tìm hiểu và nghiên cứu đề tài đồ án sinh viên dần thông thạo với công việc lập trình và một phần nào đó làm quen kĩ thuật xây dựng một hệ thống làm việc sao cho hiệu quả. Báo cáo này là kết quả làm việc của cá nhân em trong thời gian vừa qua với sự hướng dẫn của thầy Lê Trần Đức và sự mày mò nghiên cứu thêm ở nhà.

Dù đã kiểm tra nhiều lần nhưng trong báo cáo này có thể sẽ xuất hiện một số lỗi và sai sót, do đó em rất mong đợi sự góp ý từ các thầy cô.

Một lần nữa em xin chân thành cảm ơn.

MỤC LỤC

Đồ án cơ sở nghành mạng

DANH MỤC HÌNH ẢNH

I. Hệ Điều Hành

Đề tài: Xây dựng chương trình mô phỏng thuật toán lập lịch multiple-processor của CPU

1. Cơ sở lý thuyết

a) Các khái niệm cơ bản:

CPU: là viết tắt của chữ Central Processing Unit là bộ xử lý trung tâm của máy tính, gồm nhiều lõi(core).

Process: là một chương trình đang chạy đang thực thi.

Thread: là một luồng thực thi một công việc nào đó, nhiều luồng sử dụng chung không gian địa chỉ ảo của process.

Giờ CPU: là thời gian mà CPU phục vụ cho tiến trình hoạt động. Tại mỗi thời điểm chỉ có một tiến trình được phân phối giờ CPU.

Bộ điều phối: sử dụng bộ lập lịch làm công việc lập lịch biểu cho các tiến trình để thực thi, phải tối ưu được các đại lượng như thời gian hồi đáp, thông lượng và thời gian sử dụng CPU.

Lập lịch CPU: là tổ chức hàng đợi các tiến trình đã sẵn sàng để thực hiện dựa vào độ ưu tiên, thời gian thực hiện, thời gian kết thúc …

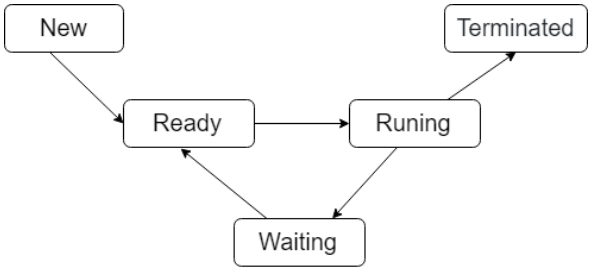
b) Tìm hiểu về tiến trình và lập lịch:

i. Tiến trình:

Trong một hệ điều hành đa chương trình có rất nhiều tiến trình phải thực thi, các tiến trình sẽ được thực thi lần lượt ban đầu chúng được duy trì trong hàng đợi cho đến khi CPU cho phép thực thi.

Tại mỗi thời điểm chỉ có một process được phân bổ CPU để hoạt động.

Các trạng thái chính của một process:



Hình 1: Vòng đời của Process

– New: process mới được tạo

– Ready: process sẵn sàng chạy

– Waiting hay blocked: process đang đợi I/O hoặc tín hiệu nào đó.

– Running: process đang được thực thi/chạy

– Terminated: process kết thúc

Một tiến trình đang trong trạng thái run sẽ chuyển trạng thái bởi một trong các lý do dưới:

– Tiến trình đã thực hiện xong

– Tiến trình tự ngắt: Khi nó đang chờ đợi một sự kiện nào đó, chúng sẽ thực hiện khi sự kiện đó xuất hiện

– Tiến trình sử dụng hết thời gian CPU dành cho nó

Việc chuyển tiến trình sang trạng thái Ready về bản chất là thực hiện vệc phân phối lại giờ CPU.

Khi process chuyển từ trạng thái run qua trạng thái Waiting hay hoàn thành và chuyển sang Terminated thì CPU đang rảnh, lúc đó bộ điều phối sẽ quyết định process nào trong hàng đợi Ready nào sẽ chạy tiếp theo còn bộ điều phối sẽ chuyển đổi ngữ cảnh và chuyển CPU đến cho tiến trình được chọn.

Các đặc điểm của tiến trình:

– Tính hướng nhập xuất: khi một tiến trình nhận được CPU, chủ yếu nó chỉ sử dụng CPU đến khi phát sinh một yêu cầu nhập xuất.

– Tính hướng xử lý: Khi một tiến trình nhận được CPU, nó có khuynh hướng sử dụng CPU đến khi hết thời gian dành cho nó.

– Độ ưu tiên của tiến trình: Các tiến trình có thể được phân cấp theo một số tiêu chuẩn đánh giá nào đó, một cách hợp lý, các tiến trình quan trọng hơn (có độ ưu tiên cao hơn) cần được ưu tiên hơn.

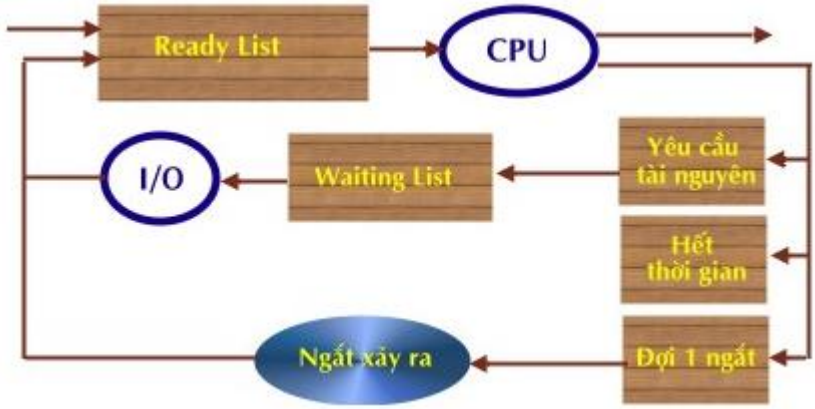
– Thời gian đã sử dụng CPU: Các tiến trình nhận được CPU trong ít thời gian là những tiến trình đã phải chờ lâu nhất, do vậy ưu tiên chọn chúng.

– Thời gian còn lại để hoàn tất tiến trình: Có thể giảm thiểu thời gian chờ đợi trung bình của các tiến trình bằng cách cho các tiến trình cần ít thời gian nhất để hoàn tất được thực hiện trước.

– Tiến trình tương tác hay xử lý theo lô: Các tiến trình của tác vụ được xử lý theo lô nói chung có thể trì hoãn trong một thời gian chấp nhận được.

ii. Lập lịch:

Thuật toán lập lịch cần xem xét và quyết định thời điểm chuyển đổi CPU giữa các tiến trình.



Hình 2: Sơ đồ hoạt động của lập lịch

Lần đầu tiến trình mới được đặt trong danh sách các tiến trình sẵn sàng (ready list), nó sẽ đợi trong danh sách này cho đến khi được chọn để cấp phát CPU và bắt đầu xử lý. Sau đó có thể xảy ra một trong các tình huống sau :

– Tiến trình phát sinh một yêu cầu một tài nguyên mà hệ thống chưa thể đáp ứng, khi đó tiến trình sẽ được chuyển sang danh sách các tiến trình đang chờ tài nguyên tương ứng. Sau đó tiến trình sẽ chuyển từ trạng thái blocked sang ready để tiếp tục thực thi.

– Tiến trình có thể bị bắt buộc tạm dừng xử lý do một ngắt xảy ra, khi đó tiến trình được đưa trở lại vào danh sách sẵn sàng để chờ được cấp CPU cho lượt tiếp theo.

Lập lịch có hai loại loại ưu tiên(Preemptive) và không ưu tiên (Non-preemptive)

Lập lịch ưu tiên: Tiến trình đang thực thi độc chiếm CPU đến khi hoàn tất hoặc tự nguyện giải phóng CPU. Nó sẽ giải phóng CPU khi một trong các trường hợp sau xảy ra:

– Khi tiến trình chuyển từ trạng thái đang xử lý(running) sang trạng thái bị khóa blocked (ví dụ: chờ một thao tác nhập xuất hay chờ một tiến trình con kết thúc…).

– Khi tiến trình kết thúc.

– Các giải thuật ưu tiên thường đơn giản và dễ cài đặt. Tuy nhiên chúng thường không thích hợp với các hệ thống tổng quát nhiều người dùng.

Lập lịch Không ưu tiên: cho phép tạm dừng hoạt động của một tiến trình đang sẵn sàng xử lý. Một tiến trình khác có độ ưu tiên cao hơn có thể dành quyền sử dụng CPU của tiến trình ban đầu.

Tiến trình khác được thực thi khi:

– Khi tiến trình chuyển từ trạng thái đang xử lý (running) sang trạng thái bị khóa blocked (ví dụ chờ một thao tác nhập xuất hay chờ một tiến trình con kết thúc…).

– Khi tiến trình chuyển từ trạng thái đang xử lý (running) sang trạng thái ready (ví dụ xảy ra một ngắt).

– Khi tiến trình chuyển từ trạng thái chờ (waiting) sang trạng thái ready (ví dụ một thao tác nhập/xuất hoàn tất).

– Khi tiến trình kết thúc.

– Các thuật toán điều phối theo nguyên tắc không độc quyền ngăn cản được tình trạng một tiến trình độc chiếm CPU, nhưng việc tạm dừng một tiến trình có thể dẫn đến các mâu thuẫn trong truy xuất, đòi hỏi phải sử dụng một phương pháp đồng bộ hóa thích hợp để giải quyết.

– Trong các hệ thống sử dụng nguyên lý điều phối độc quyền có thể xảy ra tình trạng các tác vụ cần thời gian xử lý ngắn phải chờ tác vụ xử lý với thời gian rất dài hoàn tất. Nguyên lý điều phối độc quyền thường chỉ thích hợp với các hệ xử lý theo lô.

c) Các thuật toán lập lịch

Các thuật toán lập lịch chính được áp dụng cho bộ điều phối:

– First Come First Served (FCFS)

Hình 3: Sơ đồ hoạt động của thuật toán First Come First Served

- Tiến trình nào vào Ready list trước thì được giải quyết trước
- CPU chỉ được giải phóng khi tiến trình đó kết thúc hoặc vào trạng thái I/O

Tiến trình	Thứ tự	Thời gian
P0	0	20s
P1	1	12 s
P2	2	5s

- Thứ tự thực hiện P0, P1, P3

P0 (24s)	P1 (12s)	P2 (5s)

- Thời gian chờ P0: 0s , P1: 24s, P2: 24 + 12 = 36s
- Thời gian đợi trung bình: (0+24+36)/3 = 20s

– Shortest Job First

- Tùy thuộc vào thời gian sử dụng CPU của tiến trình
- Tiến trình nào có thời gian sử dụng CPU ngắn hơn được thực hiện trước

- SJF là thuật toán tối ưu thời gian chờ trung bình nhất của một tập các tiến trình
- Nếu hai tiến trình có cùng thời gian sẽ dựa vào FIFO để xác định tiến trình nào chạy trước

Tiến trình	Thứ tự	Thời gian
P0	0	20s
P1	1	12s
P2	2	5s

- Thứ tự thực hiện P2, P1, P0

P2 (5s)	P1 (12s)	P0 (20s)

- Thời gian chờ P2: 0s, P1 : 5s, P0 : 17s
- Thời gian chờ trung bình: (0 + 5 + 17)/3 = 7.333 (s)

– Priority

- Lập lịch ưu tiên là thuật toán được sử dụng phổ biến
- Mỗi quá trình được chỉ định một chỉ số ưu tiên
- Quá trình ưu tiên cao nhất ứng với chỉ số ưu tiên nhỏ nhất thì được thực hiện trước

Tiến trình	Thứ tự	Thời gian	Độ ưu tiên
P0	0	20s	0
P1	1	12s	2
P2	2	5s	1

- Thứ tự thực hiện: P0, P2, P1

P 0 (20s)	P2 (5s)	P1 (12s)

- Thời gian chờ P0: 0s, P2: 20s, P1: 25s
- Thời gian chờ trung bình: (0+20+25)/3 = 15s

– Round Robin

Hình 4: Sơ đồ hoạt động của thuật toán Round Robin

- Các process được xử lý xoay vòng
- Quy định sẵn một giá trị thời gian sử dụng CPU cho tất cả quá trình
- Hết thời gian hệ điều hành sẽ thu hồi CPU và cấp cho process khác tiếp theo trong Ready
- Nếu Process chưa hoàn thành thì được đưa trở lại hàng đợi Ready và chờ được xử lý
- Nếu có n tiến trình trong hàng đợi Ready và thời gian cấp là q thì mỗi tiến trình sẽ nhận 1/n thời gian sử dụng CPU và thời gian chờ không quá (n-1)q (chờ n-1 process trước đó thực thi và mỗi process là q đơn vị thời gian)

Tiến trình	Thứ tự	Thời gian
P0	0	20s
P1	1	12s
P2	2	5s

- Thời gian đợi là q = 6s
- Thứ tự thực hiện: P0, P1, P2, P0, P1, P0, P0

P0 (6s)	P1 (6s)	P2 (5s)	P0 (6s)	P1 (6s)	P0 (6s)	P0 (2s)

- Thời gian chờ P0: 0+17+29, P1: 6+23, P2: 12
- Thời gian chờ trung bình: (17+29+6+23+12)/3 = 29s

d) Đánh giá các thuật toán

Xét các thuật toán dựa vào ví dụ trên:

	FCFS	SJF	Priority	RR
Thời gian đợi trung bình (s)	Lớn (36)	Nhỏ (7.3333)	Trung bình (15)	Tùy vào thời gian có thể chạy của một process
Độ phức tạp khi thực hiện	Thấp (cấp CPU cho tiến trình tiếp theo trong hàng đợi Ready)	Cao (vì khó xác định thời gian của process)	Trung bình(lựa chọn ra process có độ ưu tiên cao)	Trung bình (vì phải đưa các tiến trình chưa hoàn thành vào lại hàng đợi)
Bỏ sót tiến trình	Nếu có tiến trình quá dài dẫn đến các tiến trình sau nó không bao giờ được thực thi	Đa số các tiến trình được thực thi các tiến trình dài thực thi cuối	Những tiến trình có độ ưu tiên thấp có thể không bao giờ đươc thực thi	Tất cả các tiến trình đều được thực thi.

2. Mô tả vấn đề

a) Mục tiêu lập lịch cần đạt được:

Lập lịch CPU cần đạt được 5 mục tiêu sau:

– Sự công bằng: các tiến trình chia sẻ CPU một cách công bằng, không có tiến trình nào phải chờ đợi vô hạn.

– Tính hiệu quả: phải tận dụng tối đa khả năng của CPU, thời gian hoạt động của CPU phải lớn nhất có thể.

– Thời gian đáp ứng hợp lý: cực tiểu hoá thời gian hồi đáp cho các tương tác của người sử dụng.

– Thời gian lưu lại trong hệ thống: cực tiểu hóa thời gian hoàn tất của tiến trình

– Thông lượng tối đa: cực đại hóa số công việc xử lý trong một đơn vị thời gian

b) Vấn đề bài toán:

CPU là tài nguyên quan trọng của máy tính mọi tiến trình muốn hoạt động phải có CPU cung cấp tài nguyên tuy nhiên nó không thể đáp ứng tất cả tiến

trình cùng lúc nên phải nhờ đến bộ điều phối chia CPU cho tiến trình để thực thi.

Bộ điều phối đưa ra các quyết định. Các hệ điều hành xây dựng nhiều chiến lược khác nhau để thực hiện phân phối CPU cho tiến trình.

Vấn đề bài toán là phải mô phỏng lại những thuật toán bộ điều phối đã sử dụng để phân phối CPU cho các tiến trình thực thi, thứ tự cũng như quá trình thực thi của tiến trình một cách trực quan nhất để người dùng có thể nhìn vào và hiểu được nguyên tắc hoặt động của từng thuật toán, cũng như thấy được ưu nhược điểm của từng thuật toán. Bài toán phải nêu rõ được đầu vào đầu ra và quy trình làm việc của từng thuật toán và thiết kế giao diện dễ hiểu dễ sử dụng thân thiện với người dùng.

c) Hướng giải quyết vấn đề bài toán:

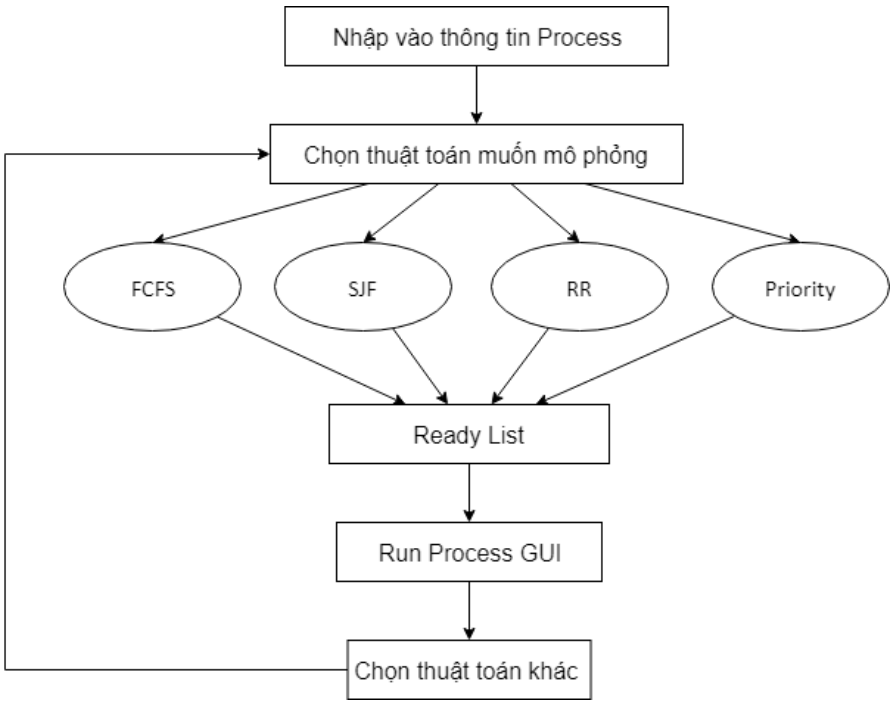
Để mô phỏng lại một cách khái quát và dễ hiểu nhất cho các thuật toán em đã chọn thực hiện trên ngôn ngữ Java, dùng giao diện swing, để mô phỏng quá trình chạy của process và thứ tự process được chọn để thực thi dựa vào yếu tố quy định bởi từng thật toán.

Cụ thể em đã xây dựng giao diện để người dùng nhìn vào có thể hình dung ra được cách hoạt động của thuật toán, để tạo giao diện em sử dụng ProgressBar để tượng trưng cho mỗi process, nó sẽ hiển thị thứ tự cũng như là thông tin của tiến trình và thông tin thuật toán

3. Triển khai và cách giải quyết bài toán

a) Sơ đồ thuật toán:

Sơ đồ thuật toán của chương trình mô phỏng thuật toán lập lịch CPU:



Hình 5: Sơ đồ thuật toán lập lịch CPU

b) Thiết kế hệ thống:

Xây dựng class để giải quyết bài toán

– Class CPU: chứa danh sách Process và thời gian chạy của CPU

– Class Process: chứa tên process, thời gian thực thi, thứ tự, độ ưu tiên, thời gian đợi và ProgressBar.

– Class Alogathim: chứa thuật toán cần mô phỏng và đối tượng CPU

– Class FIFO, SJF, RR, Priority là những class kế thừa từ class Alogthim chứa cách thức điều phối ứng với từng thuật toán.

Một số hàm quan trọng:

– Hàm run() của Class Process:

```
public void run() throws InterruptedException {
    while (i <= 100) {
        jProgressBar.setString(this._Name + "    " + i + " % ");
        Thread.sleep(100);
        int x = (int) (100 / (this._Time * 10)); // vi sleep 100ms
        i += x;
        jProgressBar.setValue(i);
    }
    jProgressBar.setValue(100);
```

```
        jProgressBar.setString(this._Name + "    " + 100 + " % ");
    }
```

– Hàm run() của Class FirstComeFirstServed:

```
@Override
public void run(List<Process> processList) {
    set_nameAlogathim("FirstComeFirstServe");
    _cpu = new CPU(new ArrayList<>(processList));

_cpu.set_runTime(_cpu.get_listProcess().stream().mapToDouble(Process::get_Time).sum());
    double timeTMP = 0;
    for (Process i : _cpu.get_listProcess()) {
        i.set_waitingTime(timeTMP);
        timeTMP += i.get_Time();
    }
}
```

– Hàm run() của Class ShortestJobFirst:

```
@Override
public void run(List<Process> processList) {
    set_nameAlogathim("Shortest Job First");
    _cpu = new CPU(new ArrayList<>(processList));
    _cpu.get_listProcess()
            .sort(((o1, o2) -> o1.get_Time() == o2.get_Time() ? o1.get_order() - o2.get_order() : o1.get_Time() > o2.get_Time() ? 1 : -1 ));

_cpu.set_runTime(_cpu.get_listProcess().stream().mapToDouble(Process::get_Time).sum());
    double timeTMP = 0;
    for (Process i : _cpu.get_listProcess()) {
        i.set_waitingTime(timeTMP);
        timeTMP += i.get_Time();
    }
}
```

– Hàm run của Class Priority:

```
@Override
public void run(List<Process> processList) {
```

```
    set_nameAlogathim("Priority");
    _cpu = new CPU(new ArrayList<>(processList));
    _cpu.get_listProcess().sort((o1, o2) -> o1.get_priority() == o2.get_priority() ?
o1.get_order() - o2.get_order() : o1.get_priority() - o2.get_priority());

_cpu.set_runTime(_cpu.get_listProcess().stream().mapToDouble(Process::get_Ti
me).sum());
    double timeTMP = 0;
    for (Process i : _cpu.get_listProcess()) {
        i.set_waitingTime(timeTMP);
        timeTMP += i.get_Time();
    }
}
```

– Hàm run của Class RoundRobin:

```
@Override
void run(List<Process> processList) {
    set_nameAlogathim("Round Robin");
    _cpu = new CPU(new ArrayList<>(processList));

_cpu.set_runTime(_cpu.get_listProcess().stream().mapToDouble(Process::get_Ti
me).sum());
    for (int i = 0; i < _cpu.get_listProcess().size(); i++) {
        double wait = 0.0;
        if (_cpu.get_listProcess().get(i).get_Time() <= _time_one_process) {
            for (int j = 0; j < i; j++) {
                wait += Math.min(_time_one_process,
_cpu.get_listProcess().get(j).get_Time());
            }
        } else {
            double tmp = _cpu.get_listProcess().get(i).get_Time();
            int k = 0;
            for (int j = 0; j < i; j++) {
                wait += Math.min(_time_one_process,
_cpu.get_listProcess().get(j).get_Time());
            }
            tmp -= _time_one_process;
```

```
                while (tmp > 0) {
                    for (int j = i + 1; j < _cpu.get_listProcess().size(); j++) {
                        if (_cpu.get_listProcess().get(j).get_Time() - k * _time_one_process
> 0)
                            wait += Math.min(_time_one_process,
_cpu.get_listProcess().get(j).get_Time() - k * _time_one_process);
                    }
                    k++;
                    for (int j = 0; j < i; j++) {
                        if (_cpu.get_listProcess().get(j).get_Time() - k * _time_one_process
> 0)
                            wait += Math.min(_time_one_process,
_cpu.get_listProcess().get(j).get_Time() - k * _time_one_process);
                    }
                    tmp -= _time_one_process;
                }
            }
            _cpu.get_listProcess().get(i).set_waitingTime(wait);
        }
    }
```

c) Các chức năng đạt được:

– Có giao diện dễ sử dụng cho người dùng

– Cho phép người dụng lựa chọn thuật toán để xem mô phỏng

– Cho phép người dùng xem thông tin của chương trình và người phát triển

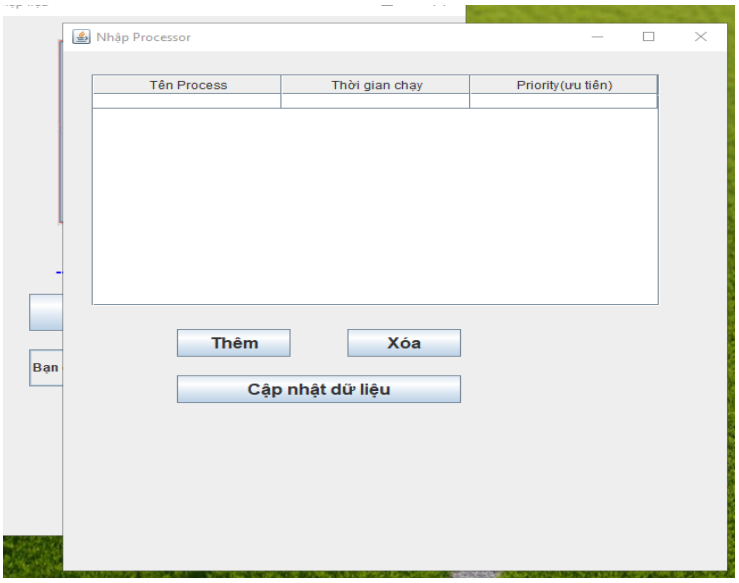
– Sau khi chọn thuật toán người dùng có thể nhập vào các process để test nếu không nhập chương trình sẽ lấy process do em đã cài đặt sẵn

– Hiển thị giao diện mô phỏng thuật toán, click vào nút run để xem demo của thuật toán, nếu muốn demo tiếp thì có thể chọn lại thuật toán khác hoặc click nút x ở màn hình chính để thoát

4. Xây dựng chương trình

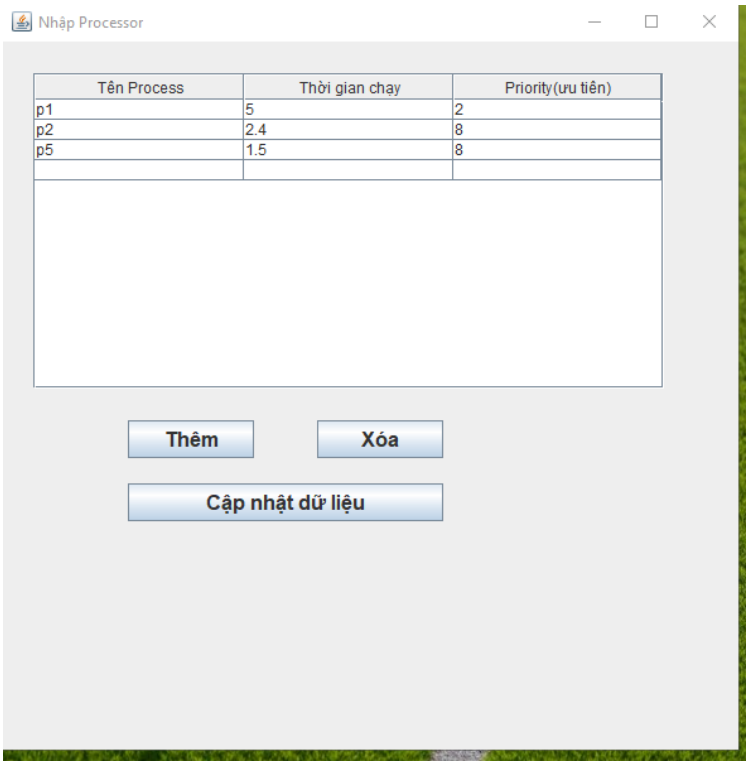
Hình 6: Trang chủ GUI của hệ thống mô phỏng lập lịch CPU

Màn hình trang chủ của chương trình người dùng có thể nhập mới Process bằng button Nhập mới nếu click vào nhập mới sẽ hiện ra bảng nhập bên dưới nếu không thì hệ chương trình mặc định sử dụng những process mà em đã tạo sẵn.



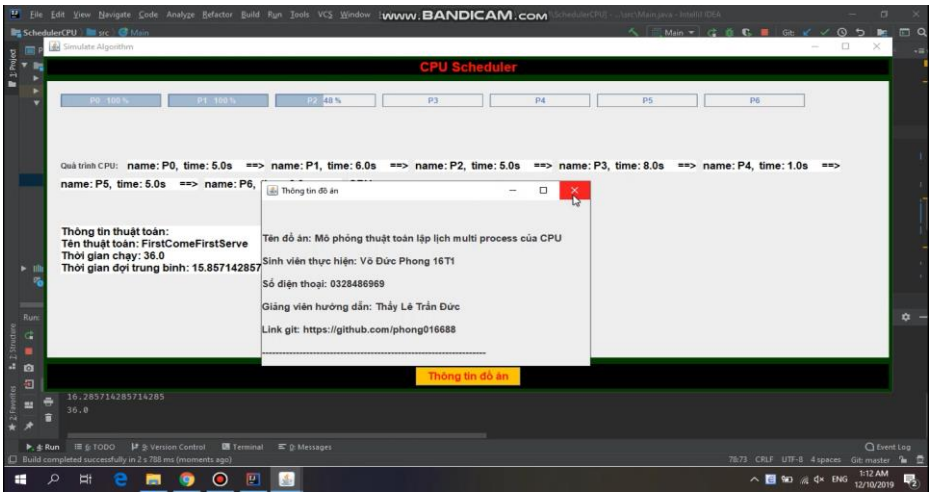
Hình 7: Màn hình nhập thông tin Process

Màn nhập dữ liệu vào gồm có tên process, thời gian chạy và độ ưu tiên của mỗi process có thể thêm hàng bằng cách nhấp vào button thêm hoặc xóa hàng bằng button xóa sau đó nhấn vào button cập nhật dữ liệu để lưu dữ liệu.

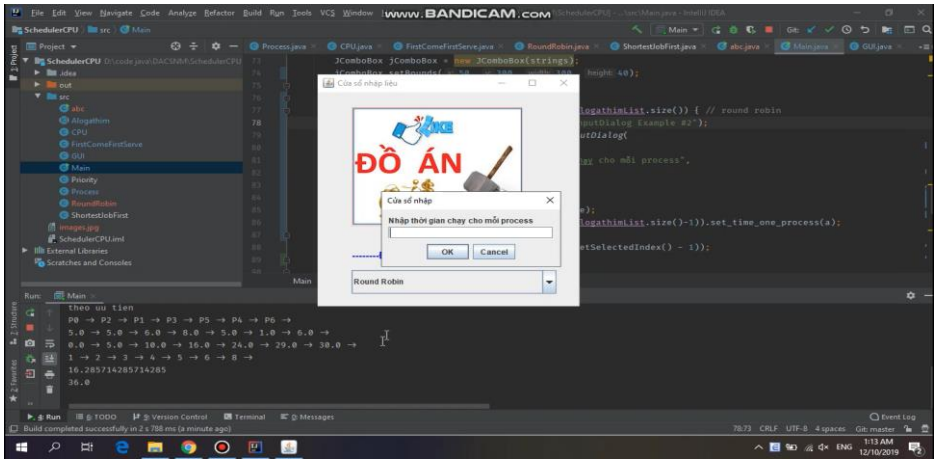


Hình 8: Màn hình nhập thông tin Process

Sau đó người dùng có thể chọn thuật toán và test sau khi chọn nó hiện lên màn hình mô phỏng thuật toán như bên dưới click vào button RUN để bắt đầu



Hình 9: Màn hình mô phỏng thuật toán của hệ thống mô phỏng lập lịch



Hình 10: Nhập thời gian định mức cho thuật toán RR

5. Kết luận và hướng phát triển

a) Kết luận:

Sau khi nghiên cứu và thực hiện em nhận thấy:

– Giải thuật First Come First Served (FCFS) là giải thuật định thời đơn giản nhất, nhưng nó có thể gây các quá trình ngắn chờ các quá trình quá trình quá dài.

– Giải thuật ShortestJobFirst(SJF) có thể tối ưu, cung cấp thời gian chờ đợi trung bình ngắn nhất nhưng độ phức tạp của SJF sẽ lớn vì đoán trước chiều dài của chu kỳ CPU kế tiếp là khó.

– Giải thuật Priority là trường hợp đặc biệt của giải thuật FCFS. Nó đơn giản cấp phát CPU tới quá trình có độ ưu tiên cao nhất.

– Cả hai định thời độ ưu tiên và SJF có thể gặp phải trở ngại của việc đói tài nguyên.

– Giải thuật RoundRobin (RR) là hợp lý hơn cho hệ thống chia sẻ thời gian. Vấn đề quan trọng là chọn định mức thời gian. Nếu định mức quá lớn, thì định thời RR giảm hơn định thời FCFS ; nếu định mức quá nhỏ thì chi phí định thời trong dạng thời gian chuyển ngữ cảnh trở nên thừa.

Trong thực tế để áp dụng người ta phối hợp nhiều thuật toán và tiêu chuẩn khác nhau để phân tài nguyên CPU và đương nhiên không có giải thuật nào tối ưu hoàn toàn nó chỉ dung hòa ở một mức độ nào đó và đáp ứng một phần

mục tiêu đặt ra của bài toán vì đơn giản những mục tiêu đã có phần mâu thuẩn lẫn nhau.

b) <u>Hướng phát triển:</u>

– Xây dựng giao diện đẹp hơn dễ sử dụng hơn nữa.

– Tối ưu hóa mã nguồn giảm số lượng code và code một cách clear nhất có thể để một người khi đọc xong có thể hiểu và làm lại bài toán này.

– Áp dụng nhiều thuật toán lẫn nhau cho CPU.

– Phát triển trên nhiều ngôn ngữ khác nhau.

II. Lập trình mạng

Đề tài: Xây dựng ứng dụng movie server sử dụng sockets

1. Cơ sở lý thuyết

a) Mô hình Client-Server:

Mô hình được phổ biến nhất và được chấp nhận rộng rãi trong các hệ thống phân tán là mô hình client/server. Trong mô hình này sẽ có một tập các tiến trình mà mỗi tiến trình đóng vai trò như là một trình quản lý tài nguyên cho một tập hợp các tài nguyên cho trước và một tập hợp các tiến trình client trong đó mỗi tiến trình thực hiện một tác vụ nào đó cần truy xuất tới tài nguyên phần cứng hoặc phần mềm dùng chung. Bản thân các trình quản lý tài nguyên cần phải truy xuất tới các tài nguyên dùng chung được quản lý bởi một tiến trình khác, vì vậy một số tiến trình vừa là tiến trình client vừa là tiến trình server. Các tiến trình phát ra các yêu cầu tới các server bất kỳ khi nào chúng cần truy xuất tới một trong các tài nguyên của các server. Nếu yêu cầu là đúng đắn thì server sẽ thực hiện hành động được yêu cầu và gửi một đáp ứng trả lời tới tiến trình client.

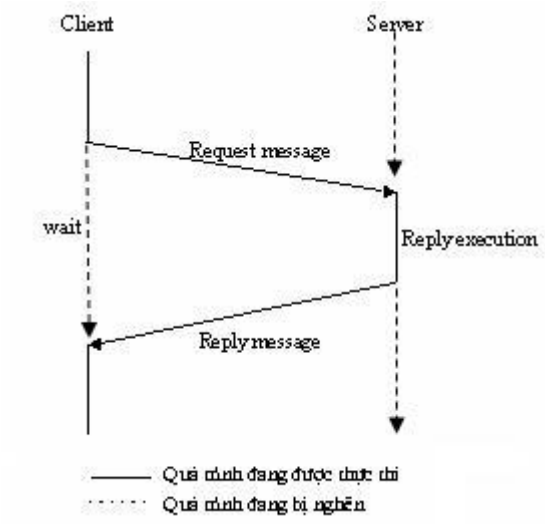
Mô hình client/server cung cấp một cách tiếp cận tổng quát để chia sẻ tài nguyên trong các hệ thống phân tán. Mô hình này có thể được cài đặt bằng rất nhiều môi trường phần cứng và phần mềm khác nhau. Các máy tính được sử dụng để chạy các tiến trình client/server có nhiều kiểu khác nhau và không cần thiết phải phân biệt giữa chúng; cả tiến trình client và tiến trình server đều có thể chạy trên cùng một máy tính. Một tiến trình server có thể sử dụng dịch vụ của một server khác.

Mô hình truyền tin client/server hướng tới việc cung cấp dịch vụ. Quá trình trao đổi dữ liệu bao gồm:

- Truyền một yêu cầu từ tiến trình client tới tiến trình server
- Yêu cầu được server xử lý
- Truyền đáp ứng cho client

Mô hình truyền tin này liên quan đến việc truyền hai thông điệp và một dạng đồng bộ hóa cụ thể giữa client và server. Tiến trình server phải nhận thức được thông điệp được yêu cầu ở bước một ngay khi nó đến và hành động phát ra yêu cầu trong client phải được tạm dừng (bị phong tỏa) và buộc tiến trình client ở trạng thái chờ cho tớ khi nó nhận được đáp ứng do server gửi về ở bước ba.

Mô hình client/server thường được cài đặt dựa trên các thao tác cơ bản là gửi (send) và nhận(receive).



Quá trình giao tiếp client và server có thể diễn ra theo một trong hai chế độ: bị phong tỏa (blocked) và không bị phong tỏa (non-blocked).

Chế độ bị phong tỏa (blocked):

Trong chế độ bị phong tỏa, khi tiến trình client hoặc server phát ra lệnh gửi dữ liệu (send), việc thực thi của tiến trình sẽ bị tạm ngừng cho tới khi tiến trình nhận phát ra lệnh nhận dữ liệu (receive).

Tương tự đối với tiến trình nhận dữ liệu, nếu tiến trình nào đó (client hoặc server) phát ra lệnh nhận dữ liệu, mà tại thời điểm đó chưa có dữ liệu gửi tới thì việc thực thi của tiến trình cũng sẽ bị tạm ngừng cho tới khi có dữ liệu gửi tới.

Chế độ không bị phong tỏa (non-blocked)

Trong chế độ này, khi tiến trình client hay server phát ra lệnh gửi dữ liệu thực sự, việc thực thi của tiến trình vẫn được tiến hành mà không quan tâm đến việc có tiến trình nào phát ra lệnh nhận dữ liệu đó hay không.

Tương tự cho trường hợp nhận dữ liệu, khi tiến trình phát ra lệnh nhận dữ liệu, nó sẽ nhận dữ liệu hiện có, việc thực thi của tiến trình vẫn được tiến hành mà không quan tâm đến việc có tiến trình nào phát ra lệnh gửi dữ liệu tiếp theo hay không.

b) Giao thức TCP:

Là một trong các giao thức cốt lõi của bộ giao thức TCP/IP. Sử dụng TCP, các ứng dụng trên các máy chủ được nối mạng có thể tạo các "kết nối" với nhau, mà qua đó chúng có thể trao đổi dữ liệu hoặc các gói tin. Giao thức này đảm bảo chuyển giao dữ liệu tới nơi nhận một cách đáng tin cậy và đúng thứ tự. TCP còn phân biệt giữa dữ liệu của nhiều ứng dụng (chẳng hạn, dịch vụ Web và dịch vụ thư điện tử) đồng thời chạy trên cùng một máy chủ.

TCP hỗ trợ nhiều giao thức ứng dụng phổ biến nhất trên Internet và các ứng dụng kết quả, trong đó có WWW, thư điện tử và Secure Shell.

Trong bộ giao thức TCP/IP, TCP là tầng trung gian giữa giao thức IP bên dưới và một ứng dụng bên trên. Các ứng dụng thường cần các kết nối đáng tin cậy kiểu đường ống để liên lạc với nhau, trong khi đó, giao thức IP không cung cấp những dòng kiểu đó, mà chỉ cung cấp dịch vụ chuyển gói tin không đáng tin cậy. TCP làm nhiệm vụ của tầng giao vận trong mô hình OSI đơn giản của các mạng máy tính.

Các ứng dụng gửi các dòng gồm các byte 8-bit tới TCP để chuyển qua mạng. TCP phân chia dòng byte này thành các đoạn (segment) có kích thước thích hợp (thường được quyết định dựa theo kích thước của đơn vị truyền dẫn tối đa (MTU) của tầng liên kết dữ liệu của mạng mà máy tính đang nằm trong đó). Sau đó, TCP chuyển các gói tin thu được tới giao thức IP để gửi nó qua một liên mạng tới mô đun TCP tại máy tính đích. TCP kiểm tra để đảm bảo không có gói tin nào bị thất lạc bằng cách gán cho mỗi gói tin một "số thứ tự" (sequence number). Số thứ tự này còn được sử dụng để đảm bảo dữ liệu được trao cho ứng dụng đích theo đúng thứ tự. Mô đun TCP tại đầu kia gửi lại "tin báo nhận" (acknowledgement) cho các gói tin đã nhận được thành công một "đồng hồ" (timer) tại nơi gửi sẽ báo time-out nếu không nhận được tin báo nhận trong khoảng thời gian bằng một round-trip time (RTT), và dữ liệu

(được coi là bị thất lạc) sẽ được gửi lại. TCP sử dụng checksum (giá trị kiểm tra) để xem có byte nào bị hỏng trong quá trình truyền hay không; giá trị này được tính toán cho mỗi khối dữ liệu tại nơi gửi trước khi nó được gửi, và được kiểm tra tại nơi nhận.

c) Socket trong java:

i. Khái quát về socket:

Như chúng ta đã biết kết nối URLs và URL cung cấp cho chúng ta một cơ cấu để truy xuất vào các tài nguyên trên Internet ở một mức tương đối cao, nhưng đôi khi chương trình của chúng ta lại yêu cầu một giao tiếp ở tầng mạng mức thấp.Ví dụ khi chúng ta viết một ứng dụng client-server.

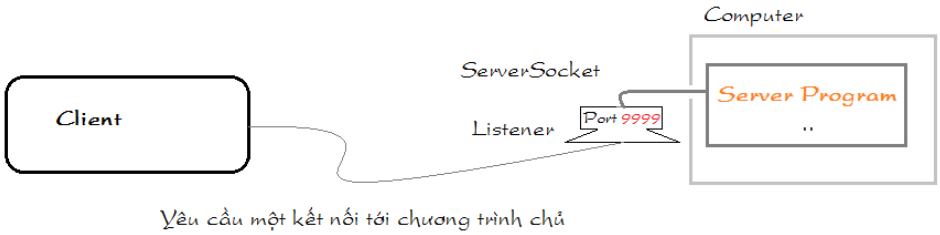
Trong một ứng dụng client-server thì phía server sẽ cung cấp một số dịch vụ, như: xử lí cơ sở dữ liệu, các yêu cầu bên phía client đưa ra, sau đó sẽ gửi lại cho phía client. Sự giao tiếp như vậy gọi là tin cậy bởi vì dữ liệu sẽ không bị mất mát, sai lệch trong quá trình truyền, server gửi cho client thông điệp gì thì phía client sẽ nhận được thông điệp nguyên như vậy. Giao thức TCP sẽ cung cấp cho chúng ta một cách thức truyền tin cậy. Để có thể nói chuyện được trên TCP thì chương trình client và chương trình server phải thiếp lập một đường truyền, và mỗi chương trình sẽ phải kết nối lại với socket là điểm cuối để kết nối, client và server muốn nói chuyện với nhau thì sẽ phải thông qua socket, mọi thông điệp sẽ phải đi qua socket. Chúng ta cứ mường tượng socket ở đây là một cái cửa mọi người muốn đi ra hay đi vào đều phải thông qua cái cửa này.

ii. Cơ chế socket:

Một socket là một điểm cuối của thông tin hai chiều liên kết giữa hai chương trình đang chạy trên mạng. Những lớp socket được dùng để đại diện cho kết nối giữa một chương trình client và một chương trình server. Trong Java gói Java.net cung cấp hai lớp Socket và ServerSocket để thực hiện kết nối giữa client và server.

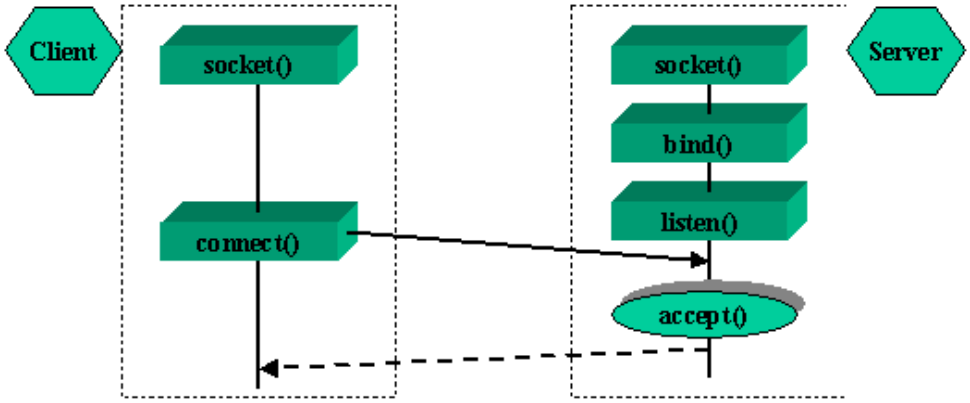
Thông thường thì server sẽ chạy trên một máy đặc biệt và có một socket giới hạn trong 1 Portnumber đặc biệt.

Phía client: client được biết hostname của máy mà server đang chạy và port number mà server đang lắng nghe. Để tạo một yêu cầu kết nối client sẽ thử hẹn gặp server ở trên máy của server thông qua port number. Client cũng cần xác định chính nó với server thông qua local port number.



Hình 11: Mô hình client-serve sử dụng socketr

Nếu mọi thứ tốt đẹp thì server sẽ đồng ý kết nối. khi đồng ý kết nối thì server sẽ tạo ra một socket mới để nói chuyện với client và cũng tạo ra một socket khác để tiếp tục lắng nghe.



Hình 12: Các thao tác của socket

Một socket có thể thực hiện bảy thao tác cơ bản:

- Kết nối với một máy ở xa (ví dụ, chuẩn bị để gửi và nhận dữ liệu)
- Gửi dữ liệu
- Nhận dữ liệu
- Ngắt liên kết
- Gán cổng
- Nghe dữ liệu đến
- Chấp nhận liên kết từ các máy ở xa trên cổng đã được gán

iii. Sử dụng socket trong java:

Class mô tả về socket

– Tạo một socket:

- Socket(InetAddress address, int port)
- Socket(String host, int port)
- Socket(InetAddress address, int port, InetAddress, localAddr, int
- localPort)
- Socket(String host, int port, InetAddress, localAddr, int localPort)
- Socket()

– Lấy thông tin về một socket

- InetAddress getInetAddress() : trả về địa chỉ mà socket kết nối đến
- int getPort() : trả về địa chỉ mà socket kết nối đến.
- InetAddress getLocalAddress() : trả về địa chỉ cục bộ.
- int getLocalPort() : trả về địa chỉ cục bộ.

– Sử dụng Streams:

- getOutputStream() throws IOException: Trả về một output stream cho việc viết các byte đến socket này.
- getInputStream() throws IOException : Trả về một input stream cho việc đọc các byte từ socket này.

ServerSocket class

Class mô tả ServerSocket

– Tạo một ServerSocket

- ServerSocket(intport) throws IOException ServerSocket(intport, intbacklog) throws IOException ServerSocket(intport, intbacklog, InetAddressbindAddr) throws IOException

– Các phương thức trong ServerSocket

- Socket accept() throws IOException: lắng nghe một kết nối đến socket này và có chấp nhận nó hay không
- void close() throws IOException: Đóng socket.
- InetAddress getInetAddress() : trả về địa chỉ cục bộ của socket
- int getLocalPort() : trả về port mà server đang lắng nghe

2. Mô tả vấn đề

a) Mục tiêu chương trình cần đạt được:

Xây dựng được ứng dụng theo mô hình client Server sử dụng để xem như web xem phim. Client và Server phải kết nối được với nhau thông qua giao thức TCP, ở hai máy khác nhau.

Phía Server:

– Server phải đáp ứng kết nối của nhiều client cùng lúc và xử lý yêu cầu nhiều client đồng thời. Xây dựng một chương trình Server đa tuyến (Threaded Server) để cho phép nhiều Client kết nối tới Server.Mỗi tuyến đảm nhận liên lạc với Client.

– Chờ và lắng nghe yêu cầu kết nối từ Client, chấp nhận kết nối và nhận Socket tương ứng. Truyền nhận thông tin qua các luồng nhận/gửi dữ liệu của socket.Tối đa hóa tốc độ xử lý của server đối với từng client.

– Gửi Video mà client yêu cầu nhanh và tỷ lệ thành công cao.

Phía Client:

– Có giao diện client dễ sử dụng và hiệu quả cao.

– Tạo một TCP Socket với địa chỉ IP và số cổng mà chương trình Server đang chạy để thiết lập kết nối tới Server, trao đổi dữ liệu với Server.

– Danh sách video được lưu trên server.

– Khi nhận dữ liệu từ Server tạo file lưu lại và mở video ra để xem, hoặc có thể dowload video về máy để xem office

b) Vấn đề bài toán:

– Tổ chức dữ liệu sao cho dễ lấy và lấy theo từng phần dữ liệu và đáp ứng được với số lượng data lớn.

– Xây dựng mô hình sao cho tối ưu thời gian đọc dữ liệu và thời gian gửi dữ liệu cho client.

– Khó quản lý kết nối tới server nên dễ dẫn đến việc server không đáp ứng được hay gọi là sập.

– Khó kiểm soát thời gian sống của thread riêng khi client không còn kết nối, dẫn đến tình trạng hao tốn tài nguyên server.

c) Hướng giải quyết vấn đề bài toán:

– Để tổ chức dữ liệu em đã lưu từng file riêng với video và hình anh mô tả của video vào từng thư mục riêng lẽ.

– Khi server được run thì thực hiện đọc dữ liệu nếu đọc xong chờ kết nối từu client có client kết nối thì cho nó vào thread riêng để xử lý biệt lập.

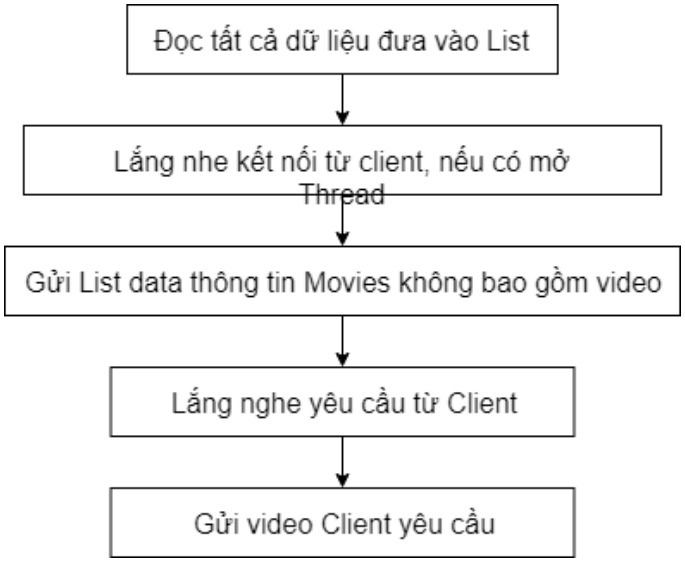
– Để quản lý thì em đã đặt HOST và PORT riêng biệt để cho những ai có thông tin kết nối vào HOST và PORT em đã mở.

– Để kiểm soát thời gian sống của thread thì em đã check nếu client ngắt kết nối thì em ngừng lắng nghe từ phía server và kết thúc thread đó để giảm tiêu tốn tài nguyên.

3. Triển khai và giải quyết bài toán

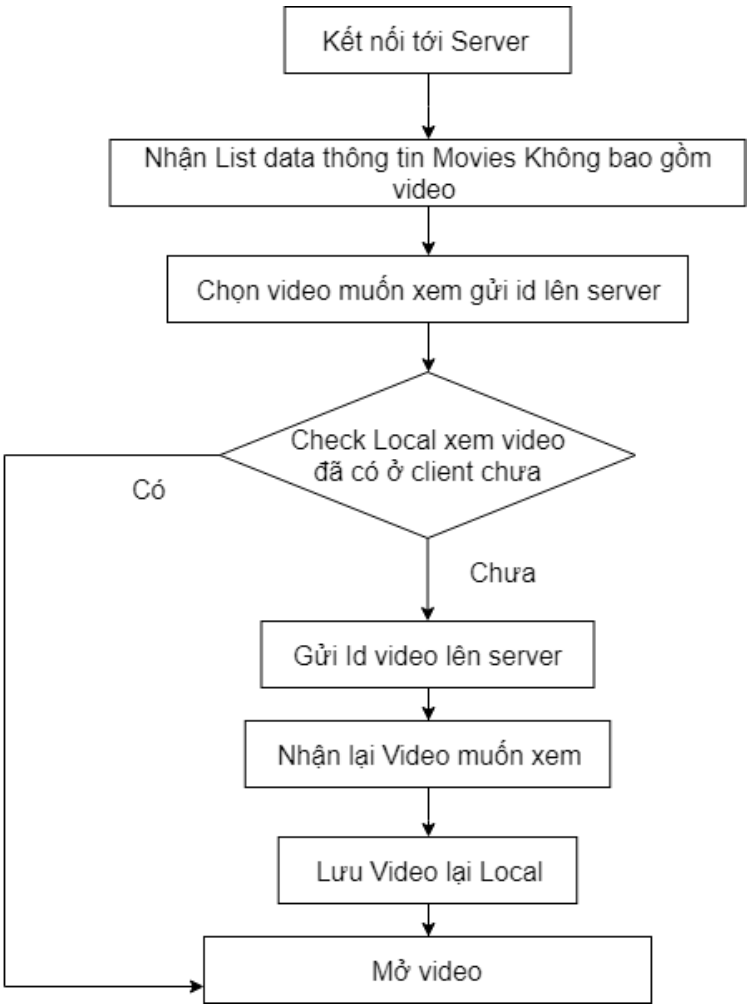
a) Sơ đồ thuật toán:

Sơ đồ thuật toán xử lý của Server:



Hình 13:Sơ đồ thuật toán của Server

Sơ đồ thuật toán của Client:



Hình 14: Sơ đồ thuật toán client

b) Thiết kế hệ thống:

Xây dựng class để gải quyết bài toán:

– Class MoviesModel chứa hình ảnh, tên, mô tả của video và video

– Class GUI: là giao diện người dùng để tương tác với chương trình

– Class Server: là class server sử dụng lớp Server Socket để mở kết nối cho client kết nối đến lấy thông tin.

– Class Client: là lớp dành cho người dùng chứa lớp GUI để hiển thị giao diện và sử dụng lớp socket để gửi yêu cầu và nhận dữ liệu từ server.

Một số hàm quan trọng:

– Hàm run() của Class Server:

```
@Override
public void run() {
    super.run();
```

```
        try {
            System.out.println("ip Address!" +
socket.getInetAddress().getHostAddress());
            System.out.println("port!" + socket.getPort());
            in = socket.getInputStream();
            os = socket.getOutputStream();
            ObjectOutputStream objectOutputStream = new ObjectOutputStream(os);
            List<MoviesModel> listMovies = new ArrayList<>();
            for (MoviesModel i : moviesModelList) {
                listMovies.add(new MoviesModel(i.getId(), i.getImager(), null,
i.getNamevideo(), i.getDetail()));
            }
            objectOutputStream.writeObject(listMovies);
            objectOutputStream.flush();


            while (true) {
                try {
                    DataInputStream dataInputStream = new DataInputStream(in);
                    String id = dataInputStream.readUTF();
                    if (!id.isEmpty()) {
                        System.out.println(id);
                        for (MoviesModel i : moviesModelList) {
                            if (i.getId().equals(id)) {
                                objectOutputStream.writeObject(i.getVideo());
                                objectOutputStream.flush();
                                break;
                            }
                        }
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                    break;
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
```

```
        }
    }
```

– Hàm xử lý Onlick vào video:

```
@Override
public void onClick(String id) {
    int index = getIndexOfList(id);
    if (index == -1) return;
    if (moviesModelList.get(index).getVideo() != null) {
        try {
            FileOutputStream fileOutputStream = new
FileOutputStream("tmp.mp4");
            fileOutputStream.write(moviesModelList.get(index).getVideo());
            fileOutputStream.close();
            Desktop.getDesktop().open(new File("tmp.mp4"));
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    } else {
        try {
            DataOutputStream dataOutputStream = new DataOutputStream(os);
            dataOutputStream.writeUTF(id);
            dataOutputStream.flush();
            while (true) {
                byte[] byteVideo = (byte[]) (objectInputStream.readObject());
                if (byteVideo.length != 0) {
                    System.out.println(byteVideo.length);
                    moviesModelList.get(index).setVideo(byteVideo);
                    try {
                        FileOutputStream fileOutputStream = new
FileOutputStream("tmp.mp4");
                        fileOutputStream.write(byteVideo);
                        fileOutputStream.close();
                        Desktop.getDesktop().open(new File("tmp.mp4"));
                    } catch (IOException ex) {
                        ex.printStackTrace();
                    }
```

```
                    break;
                }
            }
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

c) Các chức năng đạt được:

– Xây dựng được giao diện đơn giản và dễ sử dụng.

– Xây dựng được server cho kết nối đồng thời nhiều client, có thể nhận yêu cầu và gửi dữ liệu đến client.

– Mô phỏng được việc sử dụng giao thức và sử dụng socket trong java hiểu được cách thức hoạt động của giao thức và socket.

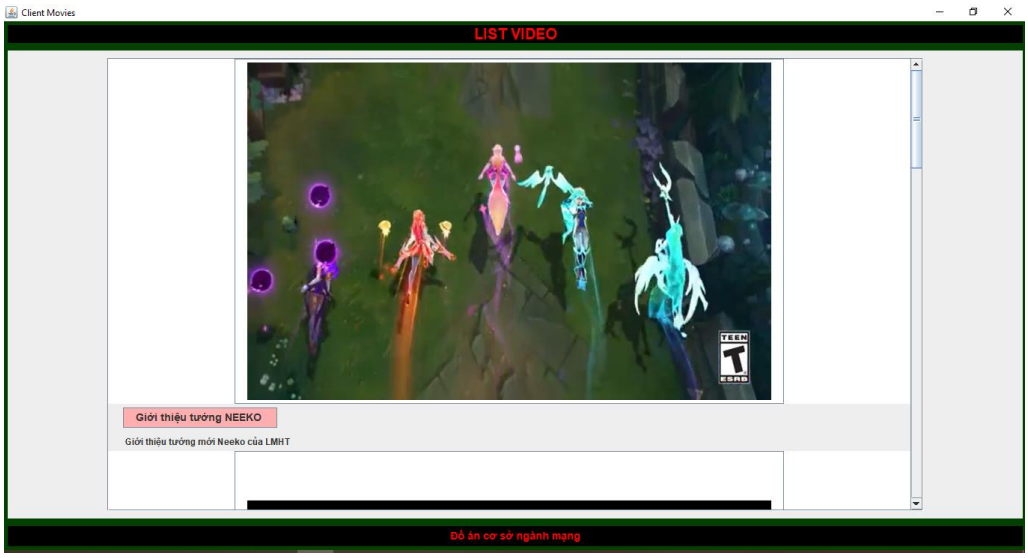
– Tối ưu hóa được thời gian gửi nhận và thời gian đọc dữ liệu.

– Có chức năng dowload để người dùng có thể dowload nhiều video về máy và xem dần.

– Có thể ứng dụng để phục vụ nhu cầu cho 1 phòng có nhiều người muốn xem movies trực tuyến với tốc độ và chất lượng cao.

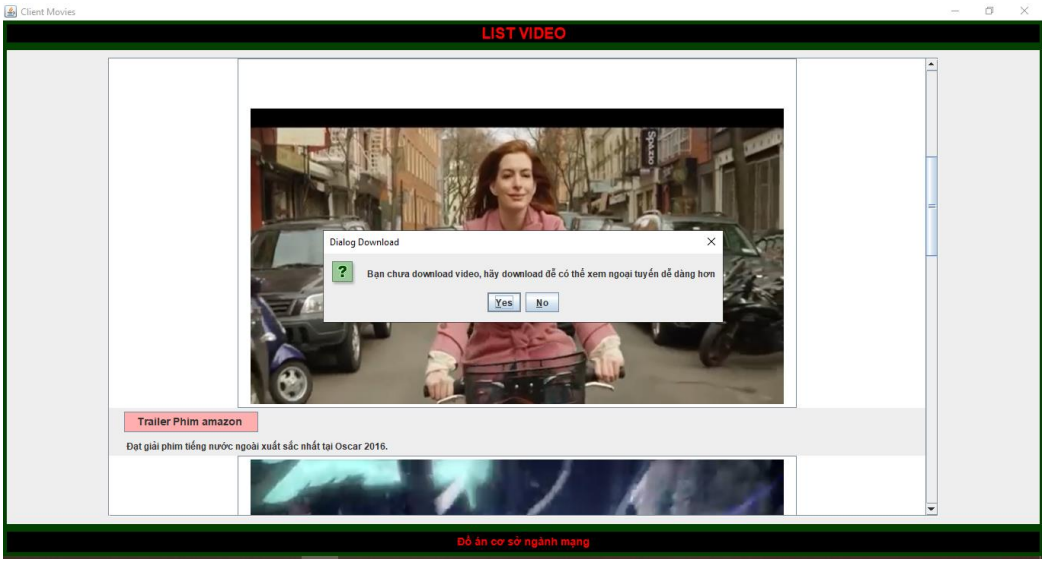
4. Xây dựng chương trình:

Khi chạy chương trình màn hình trang chủ hiện ra với nhiều video có trên server như hình 13.



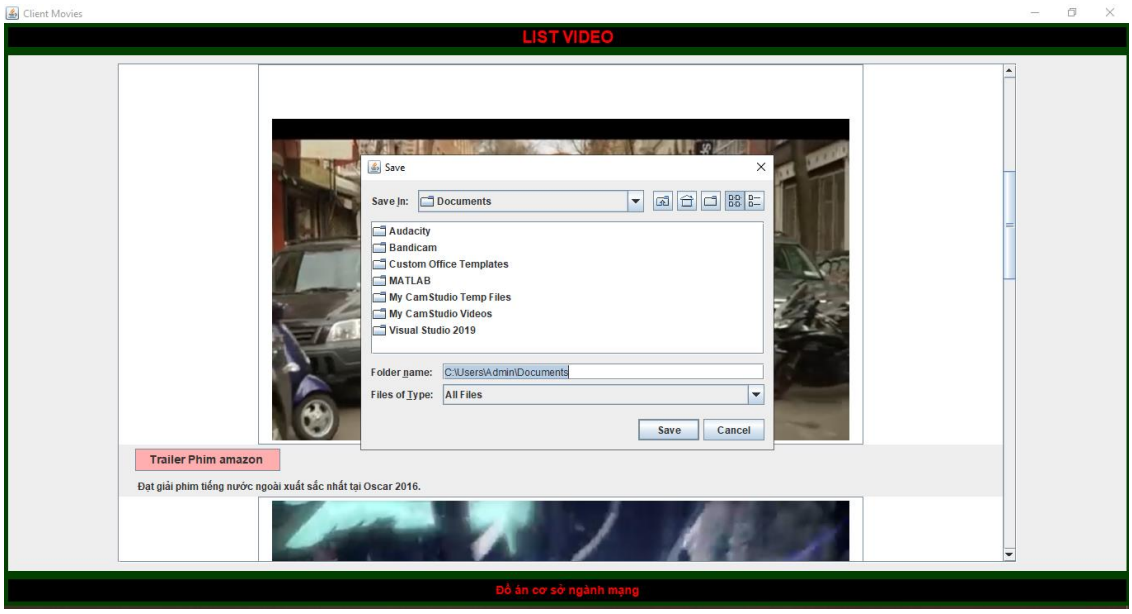
Hình 15: Trang chủ ừng dụng Movies Server

Sau khi bạn chọn video muốn xem nếu video bạn chưa dowload thì hệ thống sẽ hỏi bạn xem có muốn dowload video không.



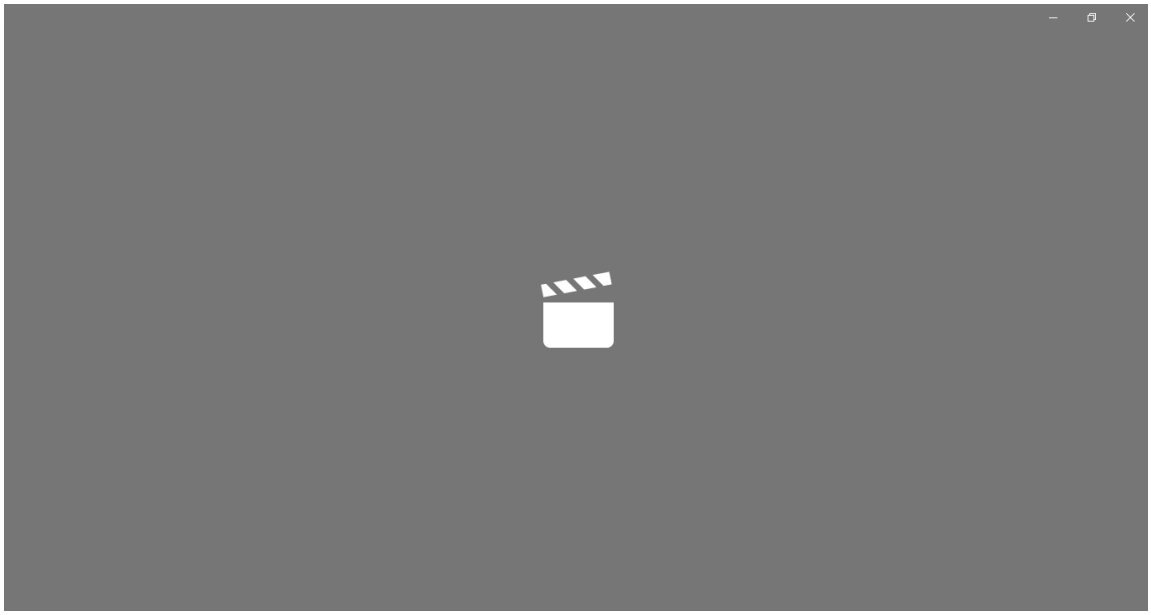
Hình 16: Xác nhận dowload video của ứng dụng Movies Server

Nếu chọn không thì bạn hệ thống sẽ mở video lên để xem hình 16 còn nếu có thì hiện JfileChoise để chọn nơi lưu file hình 15.



Hình 17: Chọn nơi lưu video của ứng dụng Movies Server

Nếu mở file hệ thống sẽ chọn trình mở file mặc định của pc bạn và mở video hình 16.



Hình 18: Mở video để xem của ứng dụng Movies Server

5. Kết luận và hướng phát triển

a. Kết luận:

Mô hình client-server là mô hình tối ưu để tổ chức hệ thống có thể áp dụng cho nhiều ứng dụng khác nhau.

Giao thức TCP là giao thức tốt để truyền dữ liệu có thể lắng nghe xem dữ liệu thành công hay thất bại và đơn giản dễ sử dụng cho hệ thống.

Chương trình đã giải quyết hầu hết yêu cầu đặt ra của bài toán.

Tính ứng dụng của chương trình chưa cao và tính thực tế thấp vì còn nhiều hạn chế như tốc độ, độ phức tạp dữ liệu, số lượng người dùng số lượng kết nối và nếu có một client đen lập trình sẵn để kết nối nhằm tăng số lượng xử lý của server thì có thể dẫn đến quá tải và sập server.

b. Hướng phát triển:

Xây dựng dữ liệu chứa đươc nhiều thông tin về movies hơn có hệ thống đánh giá bình luận video, tương tác giữa các client…

Xây dựng giao diện đẹp hơn, code clear hơn và dễ hiểu hơn.

Xây dựng hướng dẫn chi tiết để người khác có thể đọc và thiết kế lại hệ thống tương tự.

TÀI LIỆU THAM KHẢO

- https://vi.wikipedia.org/wiki/Client-server
- https://it.die.vn/s/socket/
- https://voer.edu.vn/c/dieu-phoi-tien-trinh/a039fa79/ba6e1a94
- Giáo trình giảng dạy ngôn ngữ Java của Thầy Mai Văn Hà – Trường Đại Học Bách Khoa Đà Nẵng
- https://voer.edu.vn/m/mo-hinh-client-server/eedbe7c9