KHOA CÔNG NGHỆ SỐ

BỘ MÔN CÔNG NGHỆ THÔNG TIN

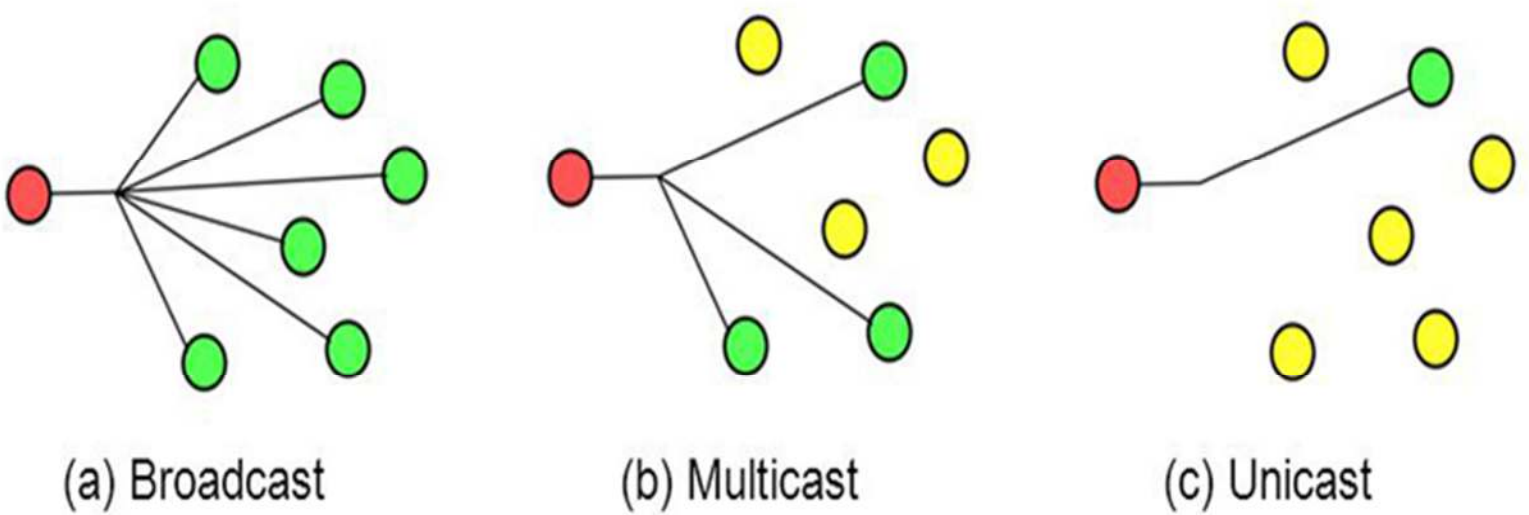
LẬP TRÌNH MẠNG

CHƯƠNG VI
LẬP TRÌNH MULTICAST

Chương VI: Lập trình Multicast

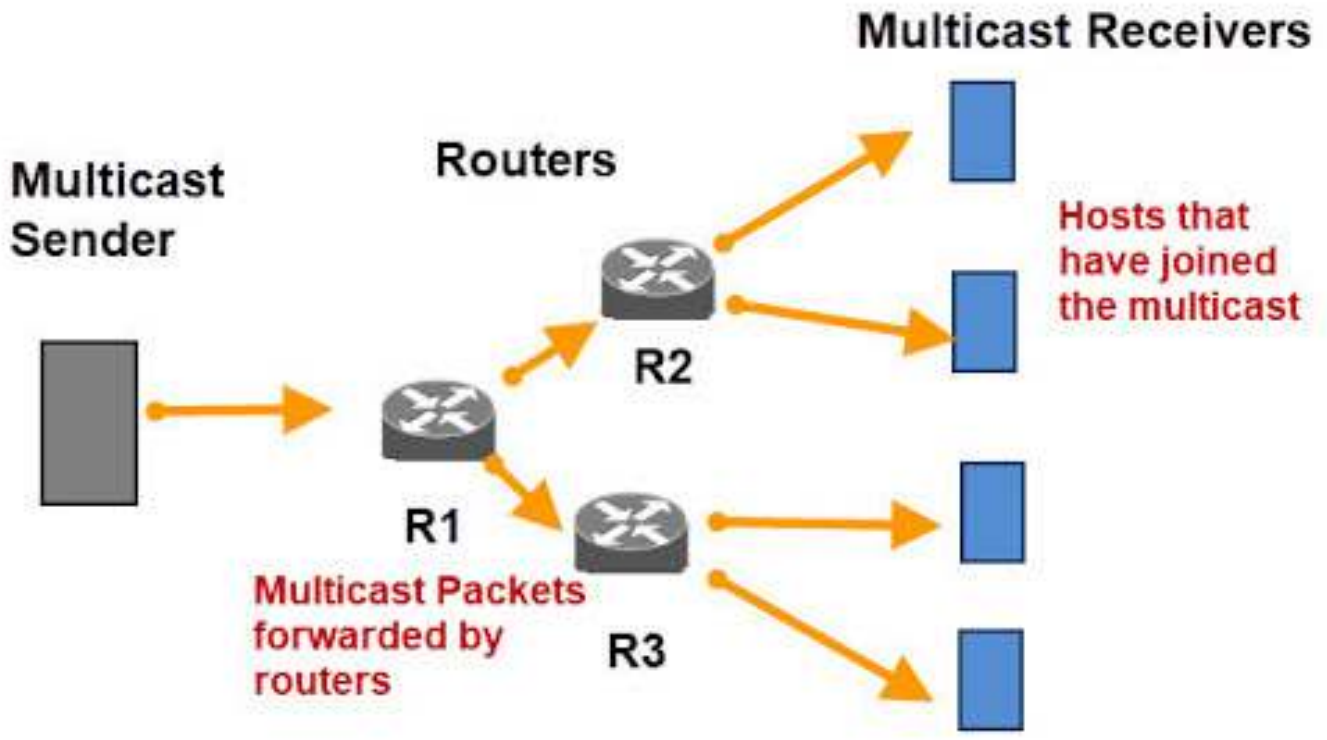
1. Khái niệm
2. Địa chỉ Multicast
3. Lập trình Multicast với Java

Chương V: Lập trình Multicast



(a) Broadcast (b) Multicast (c) Unicast

Chương V: Lập trình Multicast



Multicasting on the Internet

Chương V: Lập trình Multicast

- Unicast: 1 máy tính gửi và chỉ 1 máy tính nhận trong mạng
- Multicast: liên lạc theo nhóm
- Broadcast: gửi đến tất cả các điểm trong một mạng, không dùng để đặt riêng cho một host

Khái niệm

- Một máy có thể gửi message cho nhiều máy theo một địa chỉ.
- Chỉ cần gửi 1 packet, nhiều địa chỉ có thể nhận được
- Lợi ích:
  - Xử lý thông tin nhanh
  - Giảm lưu lượng dữ liệu trên đường truyền
  - Giải quyết bài toán quan hệ 1-nhiều và nhiều nhiều trên mạng.

Khái niệm Multicast Group

- Multicast group:
  - Là một địa chỉ, tương tự như địa chỉ của một host
  - Nhận diện như địa chỉ máy đích một cách logic hoặc một nhóm.
- Các hosts có thể gửi dữ liệu cho nhóm, tham gia hay rời bỏ nhóm.

Ứng dụng Multicast

- Truyền hình trực tiếp
- Game nhiều người chơi
- Phân phối phần mềm, email
- Giải thuật vạch đường (Routing Protocol) khi các router cập nhật thông tin
- …

Địa chỉ Multicast

- Sử dụng IGMP (Internet Group Multicasting Protocol)
- Sử dụng lớp D của địa chỉ IP: từ 224.0.0.0 đến 239.255.255.255
- Mỗi địa chỉ IP trong khoảng này biểu diễn cho một nhóm multicast
- Một địa chỉ IP trong nhóm multicast được sử dụng chung cho tất cả các thành viên của nhóm để gửi và nhận dữ liệu

Lập trình Multicast

- Sử dụng giao thức UDP
- Các bước lập trình multicast:
  - Tạo một UDP socket
  - Tham gia một nhóm multicast chỉ ra bởi một địa chỉ IP lớp D
  - Nhận các packet gửi đến cho nhóm đó
  - Gửi các packet đến các máy trong nhóm
  - Rời bỏ khỏi nhóm multicast
  - Đóng socket

Các phương thức Multicast

- Lớp java.net.MulticastSocket
- public MulticastSocket(int port): tạo socket với port
- public void joinGroup(InetAddress group): tham gia nhóm multicast tại địa chỉ xác định
- public void leaveGroup(InetAddress group): rời nhóm multicast
- public void send(DatagramPacket dp): gởi datagramPacket đi
- public synchronized void receive(Datagram Packet dp): nhận một datagramPacket

Các phương thức Multicast(tt)

VD: gia nhập nhóm multicast

- INET_ADDR = “224.0.0.3”
- PORT =8888

```
InetAddress address =InetAddress.getByName(INET_ADDR);
MulticastSocket clientSocket = new MulticastSocket(PORT);
clientSocket.joinGroup(address);
```

Các phương thức Multicast(tt)

VD: gửi datagramPacket đến địa chỉ multicast, port

- String msg = "*Sent message no* " + i

- DatagramPacket msgPacket = new DatagramPacket(msg.getBytes(),
msg.getBytes().length, addr, PORT);

- serverSocket.send(msgPacket);

Các phương thức Multicast(tt)

VD: nhận DatagramPacket trên MulticastSocket

- MulticastSocket clientSocket = new MulticastSocket(PORT);
- DatagramPacket msgPacket = new DatagramPacket(buf, buf.length);
- clientSocket.receive(msgPacket);

CHƯƠNG VI
LẬP TRÌNH ĐA TUYẾN

Lập trình đa tuyến

1. Đa nhiệm, tiến trình và luồng
2. Xử lý đa luồng trong Java
3. Mức ưu tiên của luồng
4. Vấn đề đồng bộ hóa và bài toán tắc nghẽn

Đa nhiệm, tiến trình và luồng

- Tiến trình: là một chương trình chạy trên hệ điều hành và được quản lý thông qua các thẻ
- Tiểu trình: là một đơn vị xử lý cơ bản của hệ thống. Một tiến trình sở hữu nhiều tiểu trình
- Đơn nhiệm: tại một thời điểm chỉ có duy nhất một tiến trình.
- Đa nhiệm: ở cùng một thời điểm có nhiều hơn một tiến trình thực hiện đồng thời trên cùng một máy tính. Có hai kỹ thuật đa nhiệm:
    - Đa nhiệm dựa trên các tiến trình
    - Đa nhiệm dựa trên các luồng

Đa nhiệm, tiến trình và luồng(tt)

- Một tiến trình có thể bao gồm nhiều luồng: Các luồng của một tiến trình có thể chia sẻ với nhau về không gian địa chỉ chương trình, các đoạn dữ liệu và môi trường xử lý, đồng thời cũng có vùng dữ liệu riêng để thao tác.
- Kỹ thuật đa nhiệm cho phép tận dụng được những thời gian rỗi của CPU để thực hiện những tác vụ khác.

Đa luồng (đa tuyến)

- Là khả năng làm việc với nhiều luồng
- Đa luồng chuyên sử dụng cho việc thực thi nhiều công việc đồng thời.
- Đa luồng giảm thời gian rỗi của hệ thống đến mức thấp nhất.

Tạo và quản lý luồng

- Khi chương trình Java thực thi hàm main() tức là luồng main được thực thi. Tuyến này được tạo ra một cách tự động, tại đây:
  - Các luồng con sẽ được tạo ra từ đó
  - Nó là luồng cuối cùng kết thúc việc thực thi. Ngay khi luồng main() ngừng thực thi, chương trình bị chấm dứt.

Lập trình đa luồng Java

- Java cung cấp hai giải pháp tạo lập luồng:
  1. Thiết lập lớp con của Thread
  2. Cài đặt lớp xử lý luồng từ giao diện Runnable

Lập trình đa luồng Java (tt)

Cách thứ 1: Tạo ra một lớp kế thừa từ lớp **Thread** và ghi đè phương thức run của lớp Thread:

- Lớp mới tạo có đầy đủ cơ chế của một thread
- Cần phải nạp chồng phương thức Run của lớp Thread.
- Khi chương trình chạy nó sẽ gọi một hàm đặc biệt đã được khai báo trong Thread đó là start() để bắt đầu một luồng đã được tạo ra.

Lập trình đa luồng Java (tt)

Cách thứ 2: Tạo ra một lớp triển khai từ giao diện Runnable

```
class MyClass implements Runnable
{
        //các thuộc tính
        // Nạp chồng hay viết đè một số hàm
        public void run()
        {
        …..
        }
}
```

Trạng thái và các phương thức của lớp Thread

Một luồng có thể ở một trong các trạng thái sau:

- New: Khi một luồng mới được tạo ra với toán tử **new()** và sẵn sàng hoạt động.
- Runnable: Trạng thái mà luồng đang chiếm CPU để thực hiện, khi bắt đầu thì nó gọi hàm start().
- Bộ lập lịch phân luồng của hệ điều hành sẽ quyết định luồng nào sẽ được chuyển về trạng thái Runnable và hoạt động.
- Cũng cần lưu ý rằng ở một thời điểm, một luồng ở trạng thái Runnable có thể hoặc không thể thực hiện.

Trạng thái và các phương thức của lớp Thread

Một luồng có thể ở một trong các trạng thái sau (tt):

- Non runnable (blocked): Từ trạng thái runnable chuyển sang trạng thái ngừng thực hiện (“bị chặn”) khi gọi một trong các hàm: sleep(), suspend(), wait(), hay bị chặn lại ở Input/output
- Waiting: khi ở trạng thái Runnable, một luồng thực hiện hàm wait() thì nó sẽ chuyển sang trạng thái chờ đợi (Waiting).
- Sleeping: khi ở trạng thái Runnable, một luồng thực hiện hàm sleep() thì nó sẽ chuyển sang trạng thái ngủ (Sleeping).
- Blocked: khi ở trạng thái Runnable, một luồng bị chặn lại bởi những yêu cầu về tài nguyên, như yêu cầu vào/ra (I/O), thì nó sẽ chuyển sang trạng bị chặn (Blocked).

Trạng thái và các phương thức của lớp Thread

Mỗi luồng phải thoát ra khỏi trạng thái Blocked để quay về trạng thái unnable, khi:

+ Nếu một luồng đã được cho đi "ngủ" (sleep) sau khoảng thời gian bằng số micro giây n đã được truyền vào tham số của hàm sleep(n);
+ Nếu một luồng bị chặn lại vì vào/ra và quá trình này đã kết thúc;
+ Nếu luồng bị chặn lại khi gọi hàm wait(), sau đó được thông báo tiếp tục bằng cách gọi hàm notify() hoặc notifyAll();
+ Nếu một luồng bị chặn lại để chờ monitor của đối tượng đang bị chiếm giữ bởi luồng khác, khi monitor đó được giải phóng thì luồng bị chặn này có thể tiếp tục thực hiện (khái niệm monitor được đề cập ở phần sau).

Trạng thái và các phương thức của lớp Thread

+ Nếu một luồng bị chặn lại bởi lời gọi hàm suspend(), muốn thực hiện thì trước đó phải gọi hàm resume().

+ Hàm suspend() có tác dụng tạm ngừng tuyến, ít được dùng do không nhả tài nguyên của hệ thống, dễ dẫn đến deaklock.

Nếu ta gọi các hàm không phù hợp đối với luồng th JVM sẽ phát sinh ra ngoại lệ IllegalThreadStateException.

+ Dead: Luồng chuyển sang trạng thái "chết" khi nó kết thúc hoạt động bình thường, hoặc gặp phải ngoại lệ không thực hiện tiếp được. Trong trường hợp đặc biệt, bạn có thể gọi hàm stop() để kết thúc ("giết chết") một luồng.

Trạng thái và các phương thức của lớp Thread