برنامه نویسی وب

تمرین اول

نویسنده: حسام سرخوش

شماره دانشجویی: ۹۷۱۳۰۲۰

استاد : پرهام الوانی

سوال ۱

بخش اول

- **نسخه اولیه:**
  - در ابتدا نامی نداشت و بعد ها برای تمییز از سایر نسخه ها ان را ورژن ۰.۹ نامیدند.
  - بسیار ساده و ابتدایی طراحی شده بود:
  - درخواست ها تک خطی بودند که در آن ها تنها نام فایل و Method موزد استفاده ذکر میشد.
    - مثال : `GET /index.html`
  - تنها Method موجود در آن **GET** نام داشت.
  - طبیعتا با این اوصاف پورت، پروتوکل، آدرس دامنه میزبان و ... در بدنه درخواست دیده نمیشد.
  - پاسخ نیز بسیار ساده بود و بدنه تنها شامل یک فایل میشد.
    - مثال : index.html
  - بدیهی است که با این اوصاف در درخواست ها هیچگونه Header ای وجود نداشت و هیچ گونه Response code ای در بدنه پاسخ ها تعریف نمیشد. پس اگر خطایی به وجود می آمد سرور یک فایل که در آن توضیحات ایراد به وجود آمده قرار داشت را ارسال میکرد.
- **نسخه ۱**

- بدیهی است با به وجود آمدن نسخه جدید، باید ورژن یک بودن آن در بدنه درخواست ذکر میشد تا تفاوت آن مشخص بشود.
  - مثال : `GET /itdex.html HTTP/1.0`
- قابلیت تنظیم Header هم برای درخواست ها و هم برای پاسخ ها اضافه گردید:
  - مثال :

```
GET /mypage.html HTTP/1.0
User-Agent: NCSA_Mosaic/2.0 (Windows 3.1)
```

- قابلیت Status code به پاسخ ها اضافه گردید.
- با وجود افزوده شدن قابلیت Header و وجود Header ای با نام content-type ، قابلیت ارسال بدنه های دیگری افزون بر فایل HTML مهیا میشد.
  - مثال:

```
200 OK
Date: Tue, 15 Nov 1994 08:12:31 GMT
Server: CERN/3.0 libwww/2.17
Content-Type: text/gif
(image content)
```

- نسخه ۱.۱

- بر روی نسخه قبل پیاده گردید.
- یک connection قابلیت استفاده مجدد را پیدا کرد.
- قابلیت pipeline افزوده شد که به Request ها اجازه میداد بدون آن که منتظر پاسخ قبلی بمانند، ارسال بشوند.
- پاسخ ها قابلیت Chunk شدن پیدا کردند.
- یک مکانیزم برای کش کردن افزوده شد.
- با افزوده شدن Header ای با نام host قابلیت پیاده سازی و مدیریت چندین دامنه بر روی یک IP address محیا گردید.

- هدر هایی مانند Content-Encoding، Accept-Language و Accept اضافی گردید که مدیریت محتوا را آسان تر میساخت.
  - مثال :

```
GET /en-US/docs/Glossary/Simple_header HTTP/1.1
Host: developer.mozilla.org
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:50.0) Gecko/20100101 Firefox/50.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: https://developer.mozilla.org/en-US/docs/Glossary/Simple_header
```

```
200 OK
Connection: Keep-Alive
Content-Encoding: gzip
Content-Type: text/html; charset=utf-8
Date: Wed, 20 Jul 2016 10:55:30 GMT
Etag: "547fa7e369ef56031dd3bff2ace9fc0832eb251a"
Keep-Alive: timeout=5, max=1000
Last-Modified: Tue, 19 Jul 2016 00:59:33 GMT
Server: Apache
Transfer-Encoding: chunked
Vary: Cookie, Accept-Encoding
```

- **نسخه ۲ یا** SPDY
  - توسط گوگل و با هدف بهبود عملکرد اتصالات تحت وب توسعه داده شد.
  - یک پروتکل Binary است و از Text استفاده نمیکند. این موضوع یعنی پیاده سازی بخش های عضیمی را به توسعه دهنده واگذار میکند تا قابلیت شخصی سازی بالا داشته باشد و در عین حال ساده و تمیز باقی بماند.
  - درخواست ها در این پروتکل میتوانند به صورت همزمان و موازی بر روی یک ارتباط ارسال بشوند
  - هدر ها را فشرده میسازد که تا حد زیادی حجم دوباره نویسی ها را میکاهد.
- **نسخه ۳ بر بستر QUIQ**

- توسط گوگل و با هدف رسیدن به بهره وری هر چه بیشتر توسعه داده شد و پایه آن بر اساس نسخه ۲ بنا گردید تا از همان قابلیت ها پشتیبانی کند اما این بار به جای استفاده از TCP از UDP استفاده شد.

بخش دوم

دلیل این انتخاب آن بود که UDP به دلیل پیاده سازی ساده تر و abstract تر، قابلیت شخصی سازی و بهینه سازی بیشتر را به توسعه دهندگان میدهد و در عین حال چون یک پروتکل قدیمی است که در تمام دستگاه های دیروزی و امروزی پشتیبانی میشود، طبیعتا برای استفاده از ان نیازی به باز سازی تمام لایه های زیرین شبکه نیست.

سوال ۲

۱.

- پروتکل‌های بدون حالت نوعی از پروتکل‌های شبکه هستند که در آن سیستم کاربر درخواستی را به سرور ارسال می‌کند و پاسخ سرور را مطابق با وضعیت فعلی درخواست برمی‌گرداند. چه کاربر پاسخی دریافت کند چه نکند اطلاعات بعدی پشت سر هم ارسال میشوند چون وضعیت ارسال های قبلی ذخیره نمیشوند و وضعیت هر درخواست مستقل است. این پیاده سازی بهینه و ساده است اما ممکن است اطلاعات در آن از دست بروند.
- پروتکل‌های دارای حالت نوعی از پروتکل‌های شبکه هستند که در آن زمانی که سیستم کاربر درخواستی را به سرور ارسال می‌کند انتظار پاسخ دارد و تا زمانی که پاسخی دریافت نکند مجددا درخواست را ارسال میکند. این پیاده سازی سرور و کلاینت را به شدت به یکدیگر وابسته میسازد.

II. پروتکل HTTP در تمامی نسخه ها Stateless میباشد که به دلیل سادگی و سرعت ترجیح بر آن بوده است. به همین دلیل سایر برتری های پروتکل های stateful که در آن وجود ندارد به واسطه لایه application و توسط توسعه دهندگان نرم افزار های پیاده سازی میگردد.

سوال۳

- چالش گفته شده به طور خلاصه برخواسته از این واقعیت هست که در HTTP ارتباط ها یک طرفه هستند و برای اینکه یک node مثلا کاربر وضعیت سیستم را جویا شود باید برای دریافت پاسخ درخواست ارسال کند اما نمیداند که این درخواست را باید چه زمانی ارسال کند.
- از راهکار های ارائه شده برای این مشکل میتوان به موارد زیر اشاره کرد:

I. ۲ ارتباط HTTP ایجاد کنیم. این راهکار تنها در مواردی جواب میدهد که ارتباط ها ۱ به N باشد. در عین حال N برای ما یک عدد محدود و مشخص باشد به عنوان مثال اتصال یک میکروسرویس به ۵ میکروسرویس دیگر که لازمه آن پیاده سازی اتصالات در هر ۶ سیستم و دانستن آدرس مقصد در هر مبدا است. این روش برای حالاتی که ارتباط از نوع کلاینت/سرور است بسیار ساده لوحانه است زیرا ذخیره سازی آدرس های هر کاربر به صورت دستی و چک کردن متصل بودن کاربران در قالب heartbeat request ضروری است. تا در صورت قطع شدن هر کاربر نام ان کاربر را از لیست مقصد ها خارج کنیم. که این کار بار ترافیکی را به شدت افزایش میدهد.

II. راهکار بعدی که به ذهن میرسد استفاده از polling است. به این صورت که یک client برای با خبر شدن از وضعیت سرور دائما و در یک چرخه زمانی مشخص درخواست ارسال میکند. این روش با وجود اینکه بار ترافیکی زیادی ایجاد میکند معمولا اولین روشی هست که به ذهن توسئه دهندگان مبتدی میرسد و به طرز عجیبی در بسیاری از سیستم های شناخته شده نیز استفاده میشود! به عنوان مثال سایت nubitex.ir که بزرگترین صرافی کریپتو ایرانی است از همین روش برای دریافت اطلاعات معاملات استفاده میکرد.

III. راهکار بعدی که اولین راهکار مناسب است استفاده از پروتکل WS یا WebSocket است که از شناخته شده ترین پروتکل های مبتنی بر رویداد است. این ارتباط که یک ارتباط دو طرفه است هم کار event handling را آسان میکند و هم Broad casting را ممکن میسازد. پیاده سازی به این صورت است که درخواست ها برای گروهی از نود های متصل یا تمام آن ها رو صورت رخ دادن یک رویداد در نود مبدا ارسال میگردد. این سیستم در بسیاری از پیام رسان های امروزی استفاده شده است. از پیاده سازی های مشهور این پروتکل میتوان به GraphQL subscribers و Socket.io اشاره کرد.

IV. راهکار بعدی ارائه شده استفاده از Webhook ها است. این روش به طرز عجیبی به روش اول شباهت دارد اما تفاوت آن این است که سیستمی در آن پیاده سازی شده است که هر کاربر به هنگام اتصال به سرور webhook ، اعلام میکند که به چه event هایی علاقه مند است و همچنین ادرسی را ارائه میکند که وب هوک به آن متصل شده و در صورت رخ دادن هر یک از آن رویداد ها درخواستی را به آن لینک ارسال کرده و از آن طریق کاربر

را با خبر سازد. این پیاده سازی نیز برای ارتباطاتی همچون ربات های تلگرامی و mailing system ها کاربرد دارد و منظور از کاربران در آن سرویس هایی هستند که خود از آن webhook به خصوص استفاده میکنند و نه کاربر غیر متخصصی که به یک صفحه وب درخواست ارسال کرده است.

۷. روش دیگر HTTP-streaming است که اگر بخواهیم مختصرا آن را بیان کنیم همان اتصال HTTP است با این تفاوت که زمانی که کاربر یک درخواست اولیه را به سرور ارسال میکند سرور به صورت زنجیره ای متداوم به سمت کاربر پاسخ ارسال میکند که اکثر این پاسخ ها بی مفهوم و صرفا برای حفظ ارتباط است. سرور این ارتباط را قطع نمیکند تا زمانی که event مورد نظر اتفاق بیافتد و کاربر در یکی از همین پاسخ ها از آن با خبر بشود.

سوال ۴

---

ss://asghar:1234!!@ss.myproxy.com:1234\#shadowSocks1

---

ss IS The protocol or scheme

asghar:1234!!@ss IS Hostname or Subdomain

myproxy IS Domain name

com IS Top-level domain (TLD)

1234 IS Port

#shadowSocks1 IS Fragment

سوال۵

I. با وجود آنکه ممکن است بنظر برسد خطای **401 Unauthorized** رخ داده است اما چون ایراد از سمت سرور است و نه کاربر، بهتر است خطای **500 Internal server error** نمایش داده شود. (لزومی ندارد کاربر از دلیل خطا با خبر بشود.)

II. **401 Unauthorized** رخ داده است.

III. **301 Moved Permanently** باید نمایش داده شود.

IV. **429 Too Many Requests** باید نمایش داده شود.

V. **200 OK** میتواند مناسب باشد.

VI. **403 Forbidden**

سوال۶

I. Forward proxy سرور های دانشگاه نقش یک سرور پراکسی را بازی میکنند که client ها درخواست دریافت پکیج را به آن ارسال میکنند و سرور دانشگاه میتواند خود پکیج ها را از منبع اصلی دریافت کرده و cache کند. (یا میتواند نذارد پکیج هایی که نمیخواهد را دانلود کنیم!)

II. Reverse proxy در واقع هر سیستم بزرگی همچون Instagram یا google از چنین سیستمی استفاده میکند که درخواست های ما را برای server های نزدیک تر و در دسترس تر جغرافیایی ارسال میکند. همچنین میتواند برای load balancing مورد استفاده قرار گیرد.

III. Reverse proxy به عنوان مثال یکپارچه سازی login در تمام سیستم های دانشگاه امیرکبیر با اضافه کردن سامانه accounts.aut.ac.ir.

پایان