شرح پروژه

در این پروژه باید یک سرور برای ذخیره‌سازی محتوامحور فایل‌ها طراحی و پیاده‌سازی کنید که فایل‌ها را به تکه‌های مستقل (chunk) تقسیم می‌کند، هر تکه با هش تأیید می‌شود، و در زمان دانلود نیز راستی‌آزمایی به‌صورت چانکی انجام می‌شود. ساختار کلی مشابه معماری **Merkle-DAG** در **IPFS** است؛ هر فایل با یک Content Identifier(CID) منحصربه‌فرد شناخته می‌شود که از هشِ مانیفست ریشه حاصل می‌شود.
هر عملیات آپلود یا دانلود توسط یک فرآیند مستقل انجام می‌شود تا ایزولیشن پردازه‌ای و هماهنگی بین پردازه‌ها در سناریوهای همزمان مشهود باشد، درحالی‌که پردازش هر تکه‌ی فایل (چانک) توسط یک نخ در هسته‌ی C انجام می‌گیرد. اهداف آموزشی پروژه شامل موارد زیر است:

- یادگیری مدیریت همزمانی با فرآیند (process) و نخ (thread)
- آشنایی با IPC محلی از طریق Unix Domain Socket
- تمرین مدیریت فایل‌ها و ایمنی نوشتن همزمان (atomic write)
- بازسازی ترتیبی فایل از چانک‌های پردازش‌شده به‌صورت موازی

فایل های پروژه

:main.py

بستر سرور HTTP با دو اندپوینت upload/ و download/ که فقط نقش Gateway را دارد، به موتور C از طریق سوکت یونیکس AF_UNIX در مسیر tmp/cengine.sock/ وصل می‌شود و هیچ منطق سنگینی (هش/ذخیره) در پایتون انجام نمی‌دهد. این فایل نیازی به تغییر توسط شما ندارد.

:c_engine

هسته‌ی سرور که روی AF_UNIX bind/listen می‌کند، اتصال‌ها را می‌پذیرد، پیام‌های فریم‌شده را پردازش می‌کند، چانک‌ها را هش و ذخیره می‌کند، DAG/مانیفست می‌سازد، و هنگام دانلود صحت چانک‌ها را بررسی و به ترتیب درست ارسال می‌کند.

پروتکل پیام:

فریم ثابت 1 بایت opcode + طول 4 بایت payload + big-endian؛ دستورات UPLOAD_START/UPLOAD_CHUNK/UPLOAD_FINISH و DOWNLOAD_START، و پاسخ‌های UPLOAD_DONE و DOWNLOAD_CHUNK/DOWNLOAD_DONE برای جریان داده و اعلان پایان استفاده می‌شود.

مفاهیم پروژه

I. معماری محتوامحور (Merkle-DAG)

هر فایل به چند چانک با اندازهٔ پیش‌فرض 256KB تقسیم می‌شود.

برای هر چانک مقدار multi hash از محتوای آن محاسبه می‌شود (الگوریتم پیشنهادی: BLAKE3، قابل تغییر با متغیر محیطی HASH_ALGO).

هر چانک در مسیر<blocks/aa/bb/multihash> ذخیره می‌شود.

پس از پایان آپلود، یک فایل مانیفست ساخته می‌شود و هشِ آن به‌صورت CID محاسبه می‌گردد. این CID معادل Merkle Root در درخت DAG است. مانیفست در مسیر manifests/<cid>.json ذخیره می‌شود.

II. فرمت مانیفست


```
{
"version": 1,
"hash_algo": "blake3",
"chunk_size": 262144,
"total_size": 12345678,
"filename": "example.bin",
"chunks": [
{"index": 0, "size": 262144, "hash": "mhash1"},
{"index": 1, "size": 262144, "hash": "mhash2"}
]}
```


III. تعریف CID

CID(file) = multibase(base32) · multicodec(manifest) · multihash(serialize(manifest))

IV. ایجاد فرآیند

هر درخواست آپلود یا دانلود توسط Gateway در یک فرآیند مستقل اجرا می‌شود تا جداسازی وظایف و رقابت بین کارها در سطح پردازه برای مشاهده و تحلیل OS قابل‌بررسی باشد.
هر فرآیند Gateway با ایجاد اتصال AF_UNIX به c_engine، جریان پیام‌های مربوط به آن درخواست را به‌صورت اتصال‌محور ارسال و دریافت می‌کند تا کانال ارتباطی محلی و کم‌هزینه فراهم شود.

V. مدیریت فرآیندها

فرآیندهای هم‌زمان دانلود برای یک CID مشترک باید بدون تداخل در مانیفست و بلاک‌ها عمل کنند، بنابراین قفل‌گذاری فایل/منابع مشترک در Engine و استفاده از عملیات اتمیک مانند نوشتن موقت و rename برای مانیفست ضروری است.
در **سناریوی آپلود همزمان محتوای یکسان**، در صورت یافتن بلاک با هش یکسان، Engine به‌جای تکرار ذخیره‌سازی فقط شمارنده مرجع را برای آن بلاک افزایش می‌دهد تا تکرار داده کاهش یابد و سازگاری حفظ شود.

VI. مدیریت رشته‌ها

Engine باید یک Thread Pool برای پردازش چانک‌ها داشته باشد؛ هر چانک به‌عنوان یک وظیفه شامل هش‌کردن/نوشتن یا خواندن/راستی‌آزمایی اجرا می‌شود تا از منابع CPU و I/O به صورت موازی استفاده شود.
یک ادغام‌کنندهٔ ترتیبی در Engine نتایج چانک‌ها را بر اساس index در خروجی دانلود پشت سر هم می‌چیند تا حتی با تکمیل خارج از ترتیب نخ‌ها، ترتیب نهایی فایل همواره درست باشد.

مکانیسم‌های همگام‌سازی

برای منابع مشترک مانند صف کار، جدول بلاک‌ها و مانیفست‌ها، از قفل‌ها و condition variable استفاده کنید تا بن‌بست و شرایط مسابقه پیش نیاید و بیدارباش کارآمد برای آماده‌شدن چانک‌ها وجود داشته باشد.
برای تثبیت مانیفست نهایی از الگوی write-temp + fsync + rename استفاده کنید تا خوانندگان یا نسخه‌ی قبلی یا نسخه‌ی جدید را ببینند و حالت نیمه‌نوشته مشاهده نشود.

VII. ارتباطات‌بین‌فرآیندی

IPC بین main.py و c_engine با Unix Domain Socket در مسیر /tmp/cengine.sock انجام می‌شود که یک فایل سوکت ویژه در فایل‌سیستم است و فقط روی همان میزبان معتبر است، با سربار کمتر نسبت به TCP loopback.
فریمینگ پیام‌ها مطابق قرارداد فوق از چسبندگی پیام‌ها جلوگیری می‌کند و خواندن جزئی را به‌سادگی مدیریت‌پذیر می‌کند؛ سرور باید قبل از bind مسیر قدیمی را unlink کند و در پایان اجرا فایل سوکت را پاک کند.

مواردی که این برنامه باید بتواند انجام دهد به شرح زیر است:

A. **آپلود:** دریافت فایل از upload/، ارسال جریان به Engine، تولید و بازگرداندن CID ریشه پس از تکمیل DAG و ثبت مانیفست نهایی برای دانلودهای بعدی.
B. **دانلود:** دریافت CID در download/، دریافت چانک‌ها از Engine با راستی‌آزمایی هر چانک بر اساس هش، و تحویل جریان خروجی به کلاینت با ترتیب صحیح.

C. **مدیریت همزمانی:** اجرای همزمان چند فرآیند آپلود/دانلود بدون بن‌بست، برخورد درست با منابع مشترک، و ثبت لاگ‌های آموزشی در آغاز عملیات‌ها (UPLOAD_START/DOWNLOAD_START).

D. **ذخیره‌سازی محتوامحور:** نام‌گذاری بلاک‌ها بر اساس هش و نگهداری مانیفست با لیست [{index,size,hash}] و متادیتا برای بازسازی و راستی‌آزمایی.

سناریو پروژه

گام۱: Upload

1. Gateway پیام UPLOAD_START می‌فرستد.
2. چانک‌ها با UPLOAD_CHUNK ارسال می‌شوند.
3. Engine هر چانک را هش کرده و در blocks/ ذخیره می‌کند.
4. در پایان، مانیفست ساخته و CID محاسبه می‌شود.
5. CID با UPLOAD_DONE بازگردانده می‌شود.

گام۲: Download

1. Gateway پیام DOWNLOAD_START با CID می‌فرستد.
2. Engine مانیفست را می‌خواند و برای هر چانک وظیفه‌ای ایجاد می‌کند.
3. هر چانک پس از verify موفق با DOWNLOAD_CHUNK بازگردانده می‌شود.
4. پس از آخرین چانک، پیام DOWNLOAD_DONE ارسال می‌شود.

گام۳: دانلود همزمان

- چند فرآیند می‌توانند همزمان از یک CID دانلود کنند.
- Engine از قفل‌های Reader/Writer برای مانیفست مشترک استفاده می‌کند.
- refcount بلاک‌ها در حالت فقط‌خواندنی باقی می‌ماند.

فرمت دیتاست

بلاک‌ها در مسیر blocks/ با نام فایل برابر هش چانک (مثلاً <blocks/ab/cd/hash>) ذخیره شوند تا توزیع دایرکتوری بهبود یابد و برخورد نام کاهش یابد.
مانیفست‌ها در مسیر manifests/<cid>.json ذخیره شوند و شامل آرایهٔ چانک‌ها با index، size و hash و متادیتای فایل (نام، اندازه کل و زمان‌ها) باشند تا بازسازی و راستی‌آزمایی ساده شود.

فرمت ورودی و خروجی

ورودی upload/: هدر X-Filename و Content-Length الزامی است، بدنهٔ باینری فایل به‌صورت جریان ارسال می‌شود و پس از تکمیل، پاسخ JSON شامل {"cid": "..."} بازگردانده می‌شود.
ورودی download/: پارامتر query با نام cid لازم است، خروجی application/octet-stream بدون Content-Length و به‌صورت جریان چانک‌ها تا پایان ارسال می‌شود.

نکات پیاده‌سازی IPFS

- Engine هنگام دریافت OP_UPLOAD_START و OP_DOWNLOAD_START پیام آموزشی چاپ کند تا نگاشت endpoint→operation در لاگ‌ها روشن باشد و با ابزارهای سیستم‌عامل به‌خوبی ردیابی شود.
- اندازه‌ی چانک پیش‌فرض 256KB است و می‌تواند از طریق متغیر محیطی یا هدر قابل‌پیکربندی باشد، اما ترتیب index باید همیشه دقیقاً رعایت شود.

مکانیسم‌های همگام‌سازی

- برای صف کار از mutex + condition variable استفاده کنید، برای جدول بلاک‌ها قفل مجزا قرار دهید، و برای مانیفست از قفل اختصاصی و الگوی نوشتن اتمیک بهره ببرید تا تداخل به حداقل برسد.
- در ادغام خروجی دانلود، از ساختار انتظار روی next_index و بافرهای محدود استفاده کنید تا هم ترتیب تضمین شود و هم مصرف حافظه کنترل گردد.

راهنمای اجرا

- **Engine:** برنامه c_engine را با آرگومان مسیر سوکت اجرا کنید، مثلاً ./c_engine/tmp/cengine.sock و اطمینان حاصل کنید فایل سوکت در /tmp ایجاد شده و در پایان با unlink پاک می‌شود.

- **Gateway:** برنامه main.py را اجرا کنید تا دو اندپوینت HTTP فعال شود و هر درخواست در یک فرآیند مستقل به Engine متصل گردد و جریان پیام‌ها را منتقل کند.

توضیح کوتاه درباره /tmp/cengine.sock

- /tmp/cengine.sock یک سوکت دامنه‌ی یونیکس از نوع فایل ویژه است که آدرس محلی سرور Engine برای اتصال Gateway محسوب می‌شود و فقط روی همان میزبان معتبر است، نه یک فایل معمولی داده.
- سرور قبل از bind باید نسخه‌ی قدیمی این مسیر را پاک کند و پس از خاتمه اجرا آن را حذف کند تا bind بعدی بدون خطا انجام شود و وضعیت فایل‌سیستم تمیز بماند.

دستورات و پاسخ‌ها

Opcode	نوع	Payload	توضیح
0x01	UPLOAD_START	`filename, total_size,}` `{chunk_size, hash_algo`	شروع آپلود
0x02	UPLOAD_CHUNK	`{...index, bytes}`	ارسال تکهٔ فایل
0x03	UPLOAD_FINISH	`{}`	پایان آپلود
0x10	DOWNLOAD_START	`{cid}`	شروع دانلود
0x81	UPLOAD_DONE	`{cid}`	پایان موفق آپلود
0x90	DOWNLOAD_CHUNK	`{...index, size, bytes}`	ارسال چانک دانلود
0x91	DOWNLOAD_DONE	`{}`	پایان دانلود

0xFF	ERROR	{code, message}	پیام خطا

کدهای خطا

- E_BAD_CID
- E_NOT_FOUND
- E_HASH_MISMATCH
- E_BUSY
- E_PROTO

مستندسازی

برای پروژه خود، فایلی با نام Documentation تهیه نمایید که به‌عنوان مستندسازی عمل کند. در این سند، مراحل پیاده‌سازی، روش اجرای برنامه، و سایر توضیحات ضروری را به‌طور کامل توصیف کنید. این فایل جزء الزامی تحویل پروژه محسوب می‌شود و باید در Git شما نیز قرار گیرد.

شما ملزم هستید بخش ارتباط فایل c_engine.c با سرور (main.py) را با بهره‌گیری از هوش مصنوعی به‌عنوان راهنما پیاده‌سازی کنید. همچنین، استفاده از چت‌بات‌ها در کل پروژه مجاز است. با این حال، برای پروژه‌ای که پیاده‌سازی کردید، باید مستنداتی تدوین کنید که اقدامات انجام‌گرفته را توصیف کند، به ویژه بخش ارتباطی.

این مستندات نباید توسط هوش مصنوعی نگاشته شود و باید بازتاب‌دهنده دانش و تلاش شما باشد، به گونه‌ای که ارزیابی عملکردتان صرفاً بر اساس مطالعه آن امکان‌پذیر گردد.

نمره اضافه

اگر کاربر از هرگونه رابط گرافیکی جدیدی استفاده کند که دریافت مسیر و نمایش خروجی را در محیطی خارج از محیط terminal انجام دهد، شامل نمره اضافه خواهد بود. می‌تواند این پیاده سازی سمت سرور باشد که یک کلاینت ساخته شود و همچنین می‌تواند نمایشگر هسته سرور باشد.

توضیحات تکمیلی

1) پروژه به صورت **انفرادی** یا در *گروه‌های* **2نفره** قابل انجام است (آپلود توسط هر دو عضو الزامی‌ست.)
گروه‌ها **می‌بایست** از گیت استفاده کنند.

2) زبان پیاده‌سازی پروژه، فقط **زبان برنامه‌نویسی C** می‌باشد (برای پیاده‌سازی رابط گرافیکی، محدودیت زبانی ندارید.)

3) هنگام تحویل، هر *دو* عضو گروه باید **تسلط کامل** داشته باشند.

4) فایل نهایی (شامل کد و فایل Documentation) را به‌فرمت
"IPFS<Student.IDs>_<Student.names>.zip" در Vu بارگذاری کنید.

6) **در صورت مشاهده هرگونه شباهت میان گروه‌ها، اگر درصد شباهت برابر یا بیشتر از پنجاه درصد (±50%) باشد؛ نمره طرفین به صورت** $100 - 1.5a$ **خواهد بود (a به‌معنای درصد تشابه است).**

مهلت تحویل: جمعه 14 آذر 1404 خورشیدی؛ ساعت 23:59

موفق باشید