國立成功大學 111 學年度 第二學期 計算機系統管理
National Cheng Kung University
Computer System Administration 2023 Spring

last update: 2023/5/24 16:00

Homework 4 - RAID and web API implementation

# Description

1. Deadline: 2023-06-15 (Thu.) 23:59 (UTC+8)
2. Operating system: FreeBSD 13.1-RELEASE
3. Online Judge: https://sa.imslab.org
4. TA's email: nasa@imslab.org
5. Total: 100 points.

# General Goals

- Setup a nginx server.
- Learn how to sign a SSL CA.
- Write a web to provide RAID CRUD interface.

# Precautions

- **DO NOT ATTACK JUDGE SYSTEM OTHERWISE YOU WILL FAIL THIS COURSE!!!**
- You can use other linux distro, but we only guarantee you can get 100% points on FreeBSD 13.1-RELEASE.
- You can submit multiple judge requests. However, OJ will cool down for several minutes after each judge.
- Late submissions will not be accepted.
- **BACKUP** or **SNAPSHOT** your server before judging **EVERY TIME**.
- Make sure everything is fine after reboot.

# Tasks & Requirements

General

- Create a directory under `/var` named `raid` to simulate disks.
- Install nginx package using pkg.
- Create a reverse proxy host to the next section server.
- Using the `sacertbot` to download your CA certificate and CA key. (Please download `sacertbot` from judge server)
- Nginx Configuration
    - The domain name is `<your_judge_username>.sa.` .
    - Setup a reverse proxy web Interface from 443 port to Web Interface API in localhost 8000 port.
    - Create a SSL certificate with provided CAs and install it on your web host.
    - Setup HTTP 301 redirection to redirect http request to https.
    - Disable request size limit.

Web Interface API

- We offer a simple web template written in python that can help you build your service, but you are still free to write your service in any language of your choice.
- Run your service on `localhost:8000` by uWSGI, WSGI or other web server.
- Create a FreeBSD service named `hw4`. Provide `service hw4 [start | restart | stop]` commands to control your service.
    - Your service should run as `root`.
    - You should write an rc.d script to support the hw4 service.
    - Judge will use `sysrc` to write the variables into the environment. Make sure your service is capable of retrieving the settings from the environment variables.
    - Judge will restart your service to assist with loading the variables. The following are some variables that you should support.

Name	Type	Example	Comment
NUM_DISKS	int	5	How many disks should simulate, the value should be between 3 to 10.

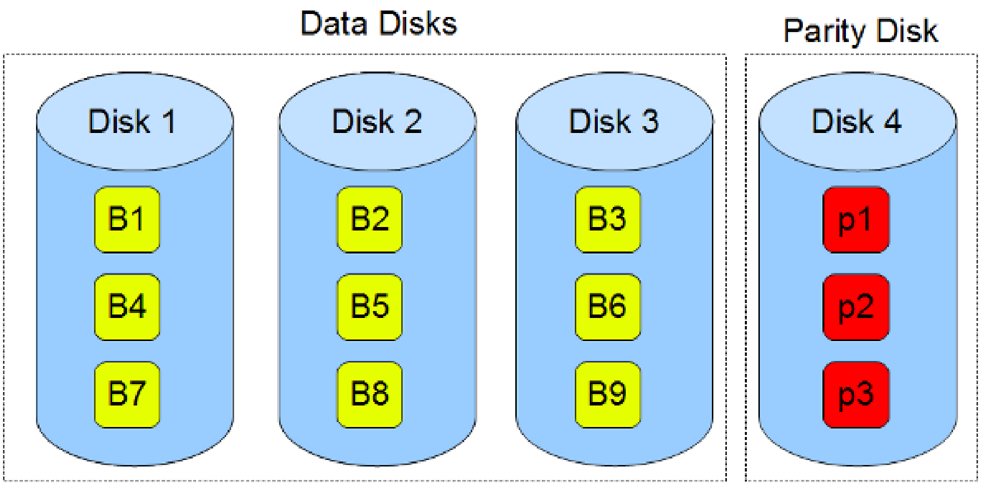
`MAX_SIZE`	int	104857600	The maximum file size that can be uploaded, specified in **BYTES**.

RAID API Implementation

- As a web interface of RAID, you should store files like a RAID-3 system. For the parity block can be calculated using XOR operation as the following formula.

  $$P = D_0 \otimes D_1 \otimes D_2 \otimes \cdots D_n$$

  where D_i denotes the data block and P denotes the parity block, all blocks are equally sized.
- The overall system architecture looks like the following graph.



RAID 3 – Bytes Striped. (and Dedicated Parity Disk)

- There should be $NUM\_DISKS - 1$ (hereinafter N) data blocks, and 1 parity block.
- The format for block paths should be `/var/raid/block-{block_id}/`, where the range of `block_id` is from $0 \sim N$ (in total NUM_DISKS folders).
- The top N disks should be used to store data blocks, while the last disk should be used to store the parity block.
- For a file of size L, the size of the first $L \% N$ blocks should be $\lfloor L/N \rfloor + 1$, and the remaining block's size should be $\lfloor L/N \rfloor$.
- If each of the data blocks are not of equal size, you should pad the shortest block(s) with null byte (`\x00`) until all blocks are equally sized.
- If the file fails to satisfy any of the below conditions, it should be considered as non-existent and must be deleted. Following are the rules for file integrity.

    - All data blocks and the parity block must exist in their designated locations.
    - All data blocks must be of equal size.
    - Parity-check must succeed.
- Before using the GET, POST, or PUT methods on the `/api/file` endpoints, you should check the integrity of the file.
- The following specifications are all http endpoints you should implement yourself.

GET /api/health

- You should provide an endpoint that allows the judge to check the health of the service.

```
Request:

GET /api/health HTTP/1.1
Host: <username>.sa
Accept: application/json


Response:

HTTP/1.1 200 OK
Server: nginx
Content-Type: application/json

{"detail":"Service healthy"}
```

POST /api/file

- This endpoint should allow the judge to upload a file, and you should use the method mentioned previously to save the file. Below are the explanations of the fields in the response body.
    - name: the uploaded file's name.
    - size: the size of the uploaded file, in bytes.
    - checksum: the content md5 hash of the uploaded file.
    - content: the content base64 encode of the uploaded file.
    - content_type: content_type of the uploaded file

```
Request:

POST /api/file/ HTTP/1.1
Host: <username>.sa
Accept: application/json
Content-Type: multipart/form-data
```

```
file: (binary; filename=m3ow87.txt)


Response:

HTTP/1.1 201 Created
Server: nginx
Content-Type: application/json

{
    "name": "m3ow87.txt",
    "size": 26,
    "checksum": "d44d11c472f88a15737ae8eee2247231",
    "content": "RG8gVSBXYW50IFRvIE1lb3cgV2l0aCBNZT8=",
    "content_type": "text/plain"
}
```

- Here are some instances where you should provide different responses.

```
If file already exists:

HTTP/1.1 409 Conflict
Server: nginx
Content-Type: application/json

{"detail":"File already exists"}


If file too large:

HTTP/1.1 413 Request Entity Too Large
Server: nginx
Content-Type: application/json

{"detail":"File too large"}
```

GET /api/file

- The endpoint should allow the judge to retrieve previously uploaded files. To achieve this, remove null bytes and concatenate data scattered across different data blocks.
    - The judge should be able to download files directly from this endpoint, and the downloaded file should have the same filename as the one uploaded by the judge. The filename can be any Unicode character.

```
Request:

GET /api/file/?filename=m3ow.txt HTTP/1.1
Host: <username>.sa
Accept: application/json
```

```

Response:

HTTP/1.1 200 OK
Server: nginx
Content-Type: application/octet-stream

(binary)
```

- Here are some instances where you should provide a different response.

```
If file not found:

HTTP/1.1 404 Not Found
Server: nginx
Content-Type: application/json

{"detail":"File not found"}
```

PUT /api/file

- The endpoint should allow the judge to update previously uploaded files. To achieve this, cover the data with upload content.

```
Request:

PUT /api/file/ HTTP/1.1
Host: <username>.sa
Accept: application/json
Content-Type: multipart/form-data

file: (binary; filename=m3ow87.txt)


Response:

HTTP/1.1 200 OK
Server: nginx
Content-Type: application/json

{
    "name": "m3ow87.txt",
    "size": 26,
    "checksum": "d44d11c472f88a15737ae8eee2247231",
    "content": "RG8gVSBXYW50IFRvIE1lb3cgV2l0aCBNZT8=",
    "content_type": "text/plain"
}
```

- Here are some instances where you should provide a different response.

```
If file not found:

HTTP/1.1 404 Not Found
```

```
Server: nginx
Content-Type: application/json

{"detail":"File not found"}

If file too large:

HTTP/1.1 413 Request Entity Too Large
Server: nginx
Content-Type: application/json

{"detail":"File too large"}
```

DELETE /api/file

- To enable the judge to delete previously uploaded files, remove the data and parity associated with the given filename.

```
Request:

DELETE /api/file/?filename=m3ow.txt HTTP/1.1
Host: <username>.sa
Accept: application/json


Response:

HTTP/1.1 200 OK
Server: nginx
Content-Type: application/json

{"detail":"File deleted"}
```

- Here are some instances where you should provide a different response.

```
If file not found:

HTTP/1.1 404 Not Found
Server: nginx
Content-Type: application/json

{"detail":"File not found"}
```

POST /api/fix/<block_id:int>

- Before accessing this endpoint, the judge will remove a block by giving block_id to simulate a broken data scenario. Once the error is induced, the judge will request the fix endpoint to restore the data, and the judge will ensure that the rest of the data blocks are intact.

```
Request:
```

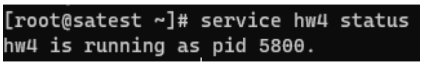
```
POST /api/file/<block_id:int> HTTP/1.1
Host: <username>.sa
Accept: application/json


Response:

HTTP/1.1 200 OK
Server: nginx
Content-Type: application/json

{"detail":"Block fixed"}
```

# Grading

Tasks	Check Condictions	Testing Commands	Score
General (30%)			
Check all mock disks folder	service hw4 start correctly	ls -d /var/raid/block*/ //The number of directory count must same as NUM_DISKS that setting with sysrc command.	5
Nginx host is configured.		curl --connect-timeout 1 -s -I -k -L http://<username>.sa | grep -i server | tail -n 1 | grep -i nginx	10
Https redirection		curl --connect-timeout 1 -s -w "%{redirect_url}" -o /dev/null http://<username>.sa	5
Check https is valid and Intermediate certificate CN is your username		python3 -c "import requests; print(requests.get('https://<username>.sa').text)" echo | openssl s_client -connect <username>.sa:443 2>&1 1>/dev/null	10
Web interface (70%)			
service hw4 start, service hw4 restart and service hw4 stop can actually control your service and provide sysrc <variables name>=<number> command to control your service environment variables		service hw4 status  [root@satest ~]# service hw4 status hw4 is running as pid 5800.	10

[GET /api/health] 200 response correct		curl -k https://<username>.sa/api/health	4
[POST /api/file] 201 response correct	Check all mock disks folder	test with python requests and create or delete files directly in /var/raid	4
[POST /api/file] 409 response correct No file uploaded successfully	Check all mock disks folder	test with python requests and create or delete files directly in /var/raid	4
[POST /api/file] 413 response correct No file uploaded successfully	Check all mock disks folder	test with python requests and create or delete files directly in /var/raid	4
[GET /api/file] 200 response correct The file was downloaded successfully with the correct filename, and the checksum matches.	Check all mock disks folder	test with python requests and create or delete files directly in /var/raid	4
[GET /api/file] 404 response correct	Check all mock disks folder	test with python requests and create or delete files directly in /var/raid	4
[PUT /api/file] 200 response correct Data updated successfully	Check all mock disks folder	test with python requests and create or delete files directly in /var/raid	4
[PUT /api/file] 404 response correct No data is updated	Check all mock disks folder	test with python requests and create or delete files directly in /var/raid	4
[PUT /api/file] 413 response correct No data is updated	Check all mock disks folder	test with python requests and create or delete files directly in /var/raid	4
[DELETE /api/file] 200 response correct All data and parity belonging to the file should be removed	Check all mock disks folder	test with python requests and create or delete files directly in /var/raid	4
[DELETE /api/file]	Check all mock disks folder	test with python requests and	4

404 response correct No data is deleteed		create or delete files directly in /var/raid	
[POST /api/fix] 200 response correct Corrupted data blocks should be repaired	Check all mock disks folder	test with python requests and create or delete files directly in /var/raid	4
Random 50 testcase	All task of Web interface	pressure test	12
Total			100

# File Segmentation Example

- Suppose the `NUM_DISK` is 4 and we have a file named `m3ow.txt` with the content "`Let's meow all day and all night!!`" In that case, the data should be scattered as follows.

```
> xxd /var/raid/block-0/m3ow.txt
00000000: 4c65 7427 7320 6d65 6f77 2061            Let's meow a

> xxd /var/raid/block-1/m3ow.txt
00000000: 6c6c 2064 6179 2061 6e64 2000            ll day and .

> xxd /var/raid/block-2/m3ow.txt
00000000: 616c 6c20 6e69 6768 7421 2100            all night!!.

> xxd /var/raid/block-3/m3ow.txt
00000000: 4165 3863 7c30 2a6c 7532 2161            Ae8c|0*lu2!a
```

# Useful Resources

- Node for OpenSSL, PKI and SSL/TLS
- FreeBSD Handbook | FreeBSD Documentation Portal
- NGINX Reverse Proxy | NGINX Plus
- Configuring HTTPS servers (nginx.org)
- Module ngx_http_core_module (nginx.org)
- FreeBSD Service Configuration: A primer & example (jacquesheunis.com)