CS 6210 Project2 Report

Kailun Li
903417169

1. Library APIs

There are 3 APIs provided by the library: (1) *call_service()*, which can be used for synchronous call; (2) *initiate_service()*, which can be used to send a request for memory segments to the service and doesn't return until there is a response from the service saying that some shared memory segments have been allocated to the calling client; (3) *call_service_async()*, which is used to asynchronously interact with the service and get the result of the compression.

The asynchronous call implements the *poll* idea given by the project instruction. The *initiate_service()* returns a struct that has a flag *has_result*, indicating whether the service has completed the compression and returned the compressed result in a buffer, which can be check later when the client process is idle. This API also starts a helper thread to do the later interactions (setting up the shared memory for this client, transmitting data through shared memory, etc.) with the service.

The synchronous *call_service()* API can also be viewed as a combination of the other two APIs but executed by a single process (i.e. single thread in this case). The actual call of this API lies in the *test_service.c* source file, in *sync_call()* function (which is kind of like wrapped the API with some file writing and other few lines of code). This should be put into the library API but I didn't get time to do this.

2. Service Internal

The client and server use 6 message queues for interaction:

request queue, response queue, original data message queue, server acknowledgment queue for having read the original data from shared memory, result queue holding message for compressed data, and client acknowledgement queue for having read the compressed data from shared memory.

(1) the client will calculate the number of segments needed to the *request queue*, and then wait for reply from *response queue*;

(2) the server will check for *request queue*, pull the request from the queue, and response the number of segments it is able to provide.

3. Get the Program Running

(1) In *TinyFile* directory, type :

touch requestq responseq origq resultq serverackq clientackq

These are files used to generate System V keys for the 6 message queues used,

(2) In *TinyFile* directory, type:

make; make service; make test_service

to compile the library, compile the service program, and compile the test program respectively.

(3) From one command window, within *TinyFile* directory, type:

./bin/service NUM_OF_SEGMENTS SIZE_OF_SEGMENT file1 file2 …

to start the service first.

(4) From another command window, within *TinyFile* directory, type:

./bin/test_service *CALL_TYPE SIZE_OF_SEGMENT file1 file2 …*

to start the test program

In step (3) and (4), the *NUM_OF_SEGMENTS and SIZE_OF_SEGMENT* are arguments to configure the shared memory cap of the service and the size per shared memory segment, *CALL_TYPE* has 2 usable values: 0 specifying synchronous call and 1 standing for asynchronous call. (in the test_service.c file, I didn't comprehensively wrote test cases, but test cases can be extended using more fork()).

On accomplishment of the execution of the service, the compressed files will appear in *TinyFile* directory with the suffix ".snp".