Assignment2
Student Name: Yi-Ting, Hsieh(a1691807)

Objective:

To gain an understanding of the challenges involved in implementing a distributed system, including synchronization and multi-threading in Java.

Design and Implementation:

1.) Protocol:

1. Implement a news aggregation system in Java. The main entites are as follows:

News aggregator - NA
* responds to a maximum number of requests from all clients (command line parameter), then shuts down
* responds to registration requests from news creators
* receives news from many news creators, and orders them
* delivers news items to any client, upon request
* once the maximum number of requests has been reached, notifies the news creators through a Shut Down Now (SDN) message and shuts down
* the news aggregator must have a dedicated news channel for each of its registered news creators
* writes a log file with these template message (note: the below is a sample log):

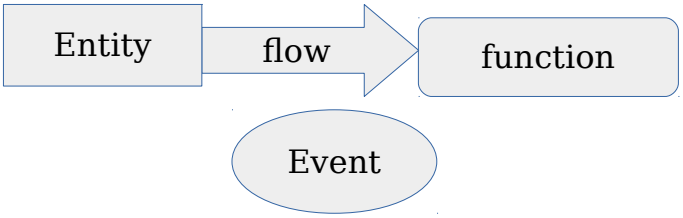
News creator - NC
* created with an ID and a news creation rate (NCR) both command line parameters
* registers its ID to the NA
* based on the rate, the news creator creates a random piece of news and sends it to the NA as plain text, in a file

Client - C
* created with an ID received as command line parameter
* send a request for news from the server
* once news received from the server, prints the news out and shuts down

****The communication between entites can only happen through the use of files.**

2. The communication behaviours between entites are as follows:

* when a client with id CID requests news from the NA, it creates a file called Client_CID
* when the news aggregator delivers news to a client CID, it creates a file called Response_CID
* when a news creator NCID registers with NA, it creates a file called Register_NCID
* the NA responds to the successful registration with a Approve_NCID file: it is only after this file is created that the news creator can send news to the NA; the news aggregator will not respond to requests from unregistered news creators
* when a news creator NCID sends the news to the news aggregator, it creates a file called Creator_NCID
* when the NA has processed the news from a creator NCID, it creates a file called Done_NCID - it is only when this file is created that the creator considers the news as delivered
* when the NA shuts down, it creates a file called SDM



2.) Design & Implementation:

Let's split out the program to four parts:

1.) SetUp:
The main purpose for this entity is cleaning up the previous data and setting up the environment for our program to use. Basically, it will delete all the files and directories under the NA/, NC/ and C/ in current directory.

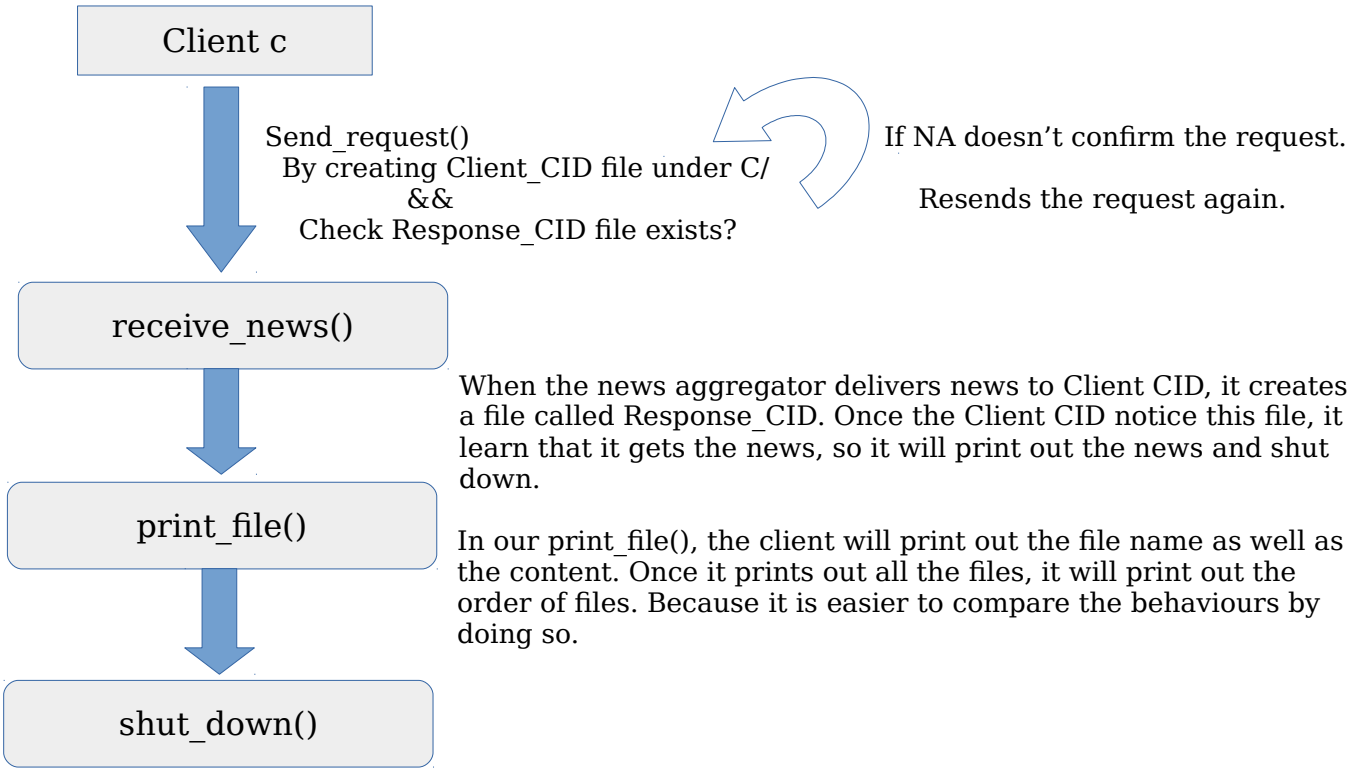
2.) Client:
First, the entity of client should be initialised with an ID received as command line parameter. Because the client would only request for news by creating Client_CID file under C/ directory from the server and receive the news. Once it receives news in it's C/C_ID/ folder, it will print out all the news content before it shuts down.

Client
Static: void main(String[] args);

- ArrayList<File> news; // stores the order of our news
- String path;
- String responsePath = "NA/Response_CID"; // stores the response file path
- Path NA_dir = Paths.get("NA");
- int id;

+Client(int id);
+void send_request(); // sends a request for news from server
+void receive_news();
+void print_file();
+void shut_down();
+int get_C_id();
+void createFile(String path);

When we create our client entity, it will automatically create a directory called C_ID/ under our C folder. By doing so, it allows us easier to distingushed different clients requests.

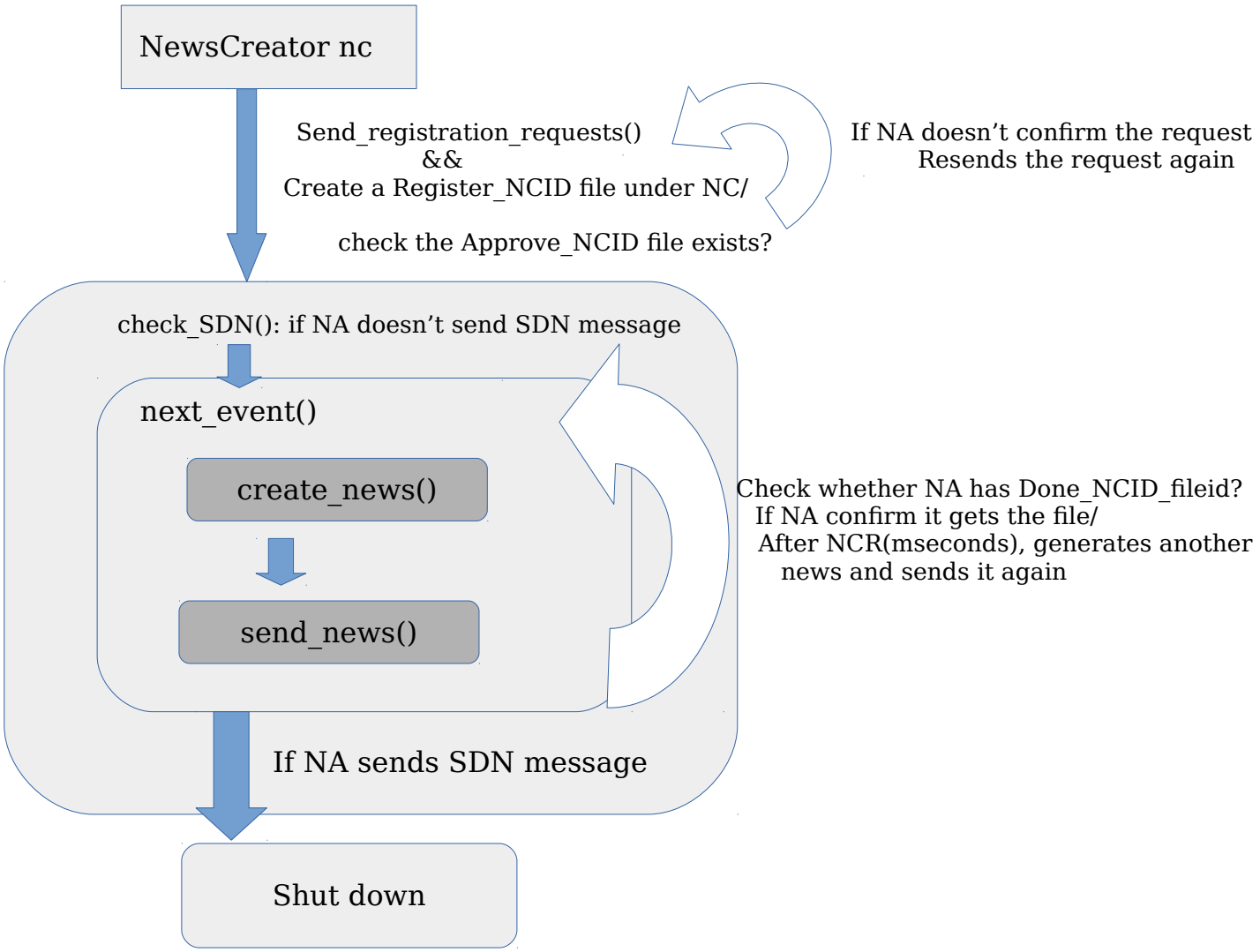


When the news aggregator delivers news to Client CID, it creates a file called Response_CID. Once the Client CID notice this file, it learn that it gets the news, so it will print out the news and shut down.

In our print_file(), the client will print out the file name as well as the content. Once it prints out all the files, it will print out the order of files. Because it is easier to compare the behaviours by doing so.

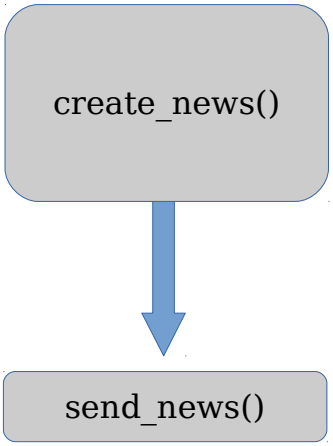
3.) NewsCreator:
The entity of our NewsCreator should be created with an ID and a news creation rate(NCR) both command line parameters. The newscreator should always register it's ID to the NA by creating Register_NCID file before it creates and sends the news. Once the newscreator gets approved from NA by Approve_NCID file, it will create a random piece of news called NCID_fileid.txt under NC/NC_ID/ folder and sends it to the NA as plain text based on the NCR.

NewsCreator
Static: void main(String[] args);
- String path; - Path NA_dir = Paths.get("NA"); - int id; - int fileID; - int news_creation_rate;
+NewsCreator(int id, int ncr); +boolean check_SDN() +void next_event(); +void send_registration_requests(); +void create_news(); +void send_news(); +int get_NC_id(); +int get_file_id(); +int get_NC_rate(); +void createFile(String path);



So what do we do in our next_event()?
We use *WatchService* to check the *NA/* directory. If we get the Done_NCID_fileid, the news creator will learn that it's news has been successfully received by the NewsAggregator and it will wait for NCR(mseconds), then it creates and sends another news again until it get the *SDN* message.

So, how do we generate a random news and send it to NA?
Let's take a closer look to our create_news() and send_news():



Inside our create_news(), it will create a new news with the file path "NC/NCID/NCID_fileid.txt". And once it create a new empty file, it will call a **RandomArticleProducer** class to generate a random news(strings) for around 400 lines, which would only take 5mseconds for each file. The usage is:

```
RandomArticleProducer.genFile("filePath");
```

You can use the following code to test the time:

```
long time = System.currentTimeMillis();
RandomArticleProducer.genFile("filePath");
System.out.println(System.currentTimeMillis() - time);
```

Once the NC create the news, then it calls the send_news(). In send_news(), it will create another file called *CREATOR_NCID.* On the other hand, our NA will build a channel between *NC/NCID/* and *NA/News* if it checks the news creator has already registered its ID. What this channel for is that it will have another *WatchService* check the *NC/NCID/* directory. If it has new files, then it will be sent to *NA/News*. The idea and behaviour is quite similar as TCP protocol. Once the channel has been built up, we don't need to build up the connection every time when we send a new file. I think this design is not only easy for implementation, but also matching the real world scenario. But the security issue should be taken into account as well.

4.) NewsAggregator:
The entity of NewsAggregator should be initialised with an maximum number of requests from all clients(command line parameter). It should responds to registration requests from news creators before it receives news from that news creator, which means it only accept news sent from registrated ID from news creator. On the other hand, the news aggregator must deliver news items to any client, upon request. Once the maximum number of requests has been reached, it would notify all news creators through a Shut Down Now(SDN) message and shuts down. In addition, the news aggregator must have a dedicated news channel for each of its registered news creators, and it should support to write a log file to record the whole information.

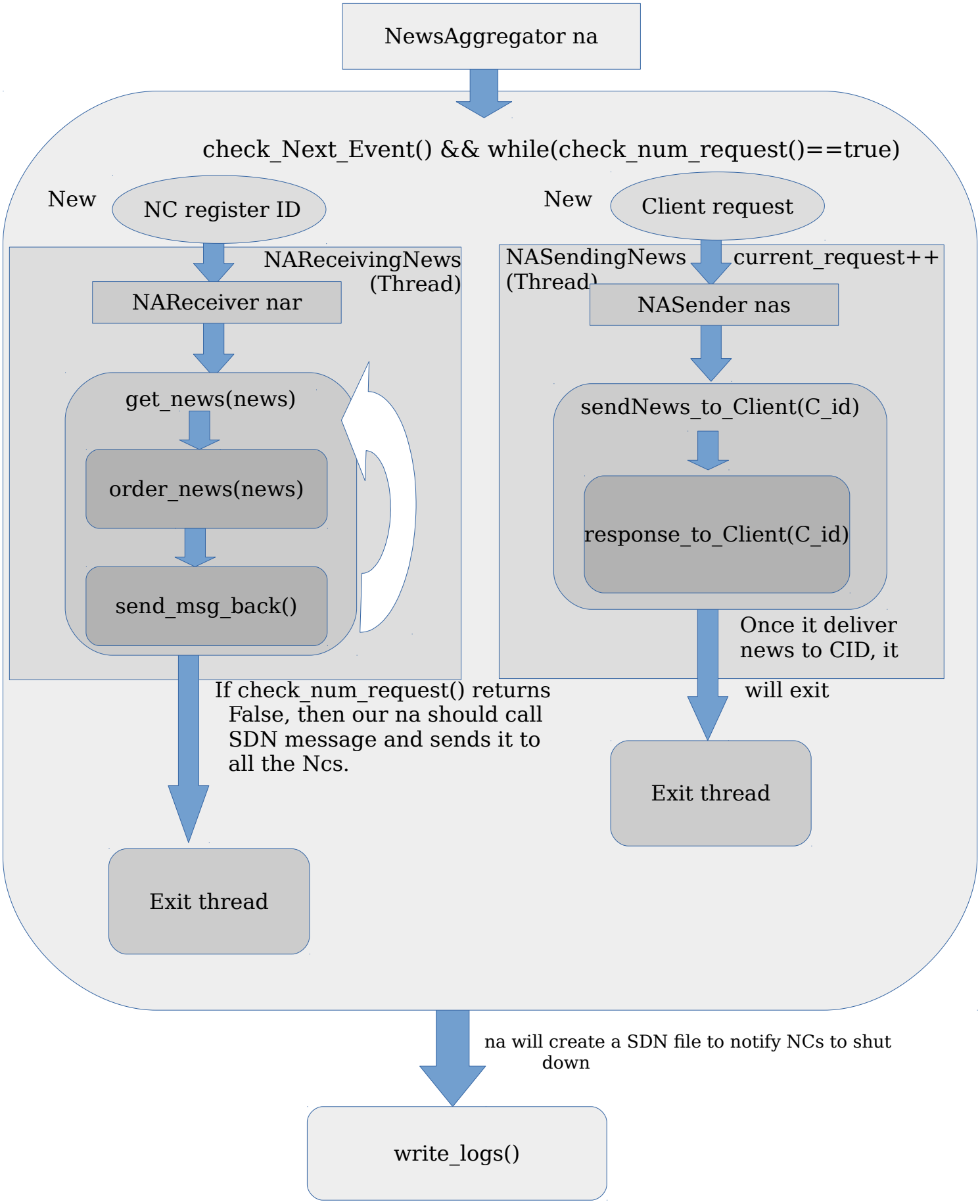
NewsAggregator
Static: ArrayList<Integer> NewsCreatorID; ArrayList<Integer> ClientID; ArrayList<File> news; String log_buffer; Path nc_dir = Paths.get(“NC”); Path c_dir = Paths.get(“C”); Path na_news = Paths.get(“NA/News”); int maximum_reqeust; int current_requests void main(String[] args);
+NewsAggregator(int max_requests); +void check_Next_event(); +void response_NC_requests(int id); +void check_if_NC_registrated(int id); +void write_logs(); +void createFile(String path); +boolean check_num_requests()

NAReceiver class for handling NCID requests:
Once our NA gets the registration request from the NC, it creates a **NAReceiver** entity which is inherited from NewsAggregator class, running under a thread called **NAReceivingNews** to receive the news sent from this NC. It will only exit when it gets the SDN mesesage.

NAReceiver extends NewsAggregatos
- Path nc_path; - int nc_id; - int nc_file_id;
+NAReceiver(int max, int nc_id); +void get_news(File file); +void order_news(File file); +void send_msg_back(); // create DONE_NCID_fileID file

NASender class for handling Clients requests:
Once our NA gets the request from the Client, it creates a new **NASender** entity, which is inherited from NewsAggregator class, running under a thread called **NASendingNews** to send the news to the client. Once it sends all the news it has so far, it will exit.

NASender extends NewsAggregatos
- Path c_path; - int c_id; - ArrayList<File> news_NAS;
+NASender(int max, int c_id); +void sendNews_to_Client(int C_ID); +void response_to_Client(int id); // create Response_CID file


NewsAggregator na
check_Next_Event() && while(check_num_request()==true)
New
NC register ID
NAReceivingNews
(Thread)
NAReceiver nar
get_news(news)
order_news(news)
send_msg_back()
If check_num_request() returns
False, then our na should call
SDN message and sends it to
all the Ncs.
Exit thread
New
Client request
NASendingNews
(Thread)
current_request++
NASender nas
sendNews_to_Client(C_id)
response_to_Client(C_id)
Once it deliver
news to CID, it
will exit
Exit thread
na will create a SDN file to notify NCs to shut
down
write_logs()

Example output of our logfile:

[Notes]
Below is sample output of our logfile. There is one thing should be noticed, that is for our NewsAggregator, it will only write the log file and shut down until the accumulative client requests reaches to the maximum requests number. If you try to kill or interrupt the NA process, it will not create log file. To be honest, I do think about implementing a functionality that allows NA could read instructions from terminal so that we can ask it to shut down or write the log file. But it is not in the assignment specification. Moreover, it is quite normal if a program does not finish and exit correctly, it would lose the information. Even for those log files in Linux, like systemctl or journalctl, if we just kill it's process, it may lose information. So if you want to test the log file, the easiest way is setting the maximum requests number to 1 and call one Client sending the request.

Log file:

NA -NEW CREATOR- 15	<---------- a new NC with ID:15 requests for registration
NA -CONFIRM- 15	<---------- NA confirms
NA -RECEIVED- news from NC_15	<---------- NA receives news from NC_15
NA -RECEIVED- news from NC_15	
NA -RECEIVED- news from NC_15	
NA -RECEIVED- news from NC_15	
NA -RECEIVED- news from NC_15	
NA -RECEIVED- news from NC_15	
NA -RECEIVED- news from NC_15	
NA -NEW CREATOR- 20	<---------- a new NC with ID:20 requests for registration
NA -CONFIRM- 20	<---------- NA confirms
NA -RECEIVED- news from NC_15	<---------- NA receives news from NC_15
NA -RECEIVED- news from NC_20	<---------- NA receives news from NC_20
NA -REQUEST- from Client_1	<---------- a Client with ID:1 requests for news
NA -SENT- to Client_1	<---------- NA sends the news to Client_1
NA -SDN-	<---------- NA sends Shut Down Now(SDN) message

Analyze:

1.) The basic functionality:

- all entites behave as described from assignment specification
- log file captures the behaviour of the system
- all entites communicate as described from assignment specification

From the above discussion and diagrams, I believe it already shows that our program satisfies all the basic requirement. All entites behave as described. And the entites do use files to communicate. More than that, entites use *WatchService* to check new events, like file created or modified, rather than just using busy waiting. However, as I have pointed out the problem we have for log file. You can only obtain the log file if the NA gets the accumulative Clients requests.

2.) The advanced functionality:

- no or minimum busy waiting
- the news aggregator shuts down only once all news creators have shut down
- retry on fail on clients and news creators, and mechanism on NA for duplicate requests
- news aggregator remembers clients, so when a client with the same id connects, it sends only the most recent news since the client last requested

As we have already discussed above, all entites have *WatchService* to check the new event for specific paths. So for our program, it does have minimum busy waiting. I only check whether the file exists for Shut Down Now(SDN) messages or few NA's confirmed files for NC and Client.

For the second one, as shown in our NewsCreator, all the NCs will shut down immediately once they receive the SDN message. On the other hand, because our NA still needs to send the news to our latest client, so it will shut down only when the latest client gets all the news. Moreover, the main thread will exit and write the log file if and only if all other threads exit in theoritically. From the diagram above for our NA, we know that the NAReceiving will exit once if it see the SDN message. For our NASending, it will exit right after it delivers all the news to the client. Because we have already knew that each news requried around 5mseconds to write into a file. In our main thread of NA, once it sends SDN message, I call the main thread to wait for 6 times number of news of miliseconds(6*number_of_news). So it should be sufficient enough to deliver all the news to client before it writes the log file and exits. However, it is not the best approach. I have also done some reading on this topic. There are some techniques we can use, such as ExecutorService, which can manage pools of threads. But I am running out of time to implement the code and test it, our program do not have this method.

I do not have that functionality for the last one. The reason is that once our NASender deliver all the news to the client, it will just exit directly. I did not create another functionality for our NA to store all the information for client. But, I do implement the retry on fail for our news creator. If the news creator does not get any respond from NA for it's registration request, it will send another registration request again. But for our NA, it won't answer the duplicate requests. I think next time I could add ENTRY_MODIFY event for watch service, so that the WatchService will notice the duplicate request.

Result:

Discussion whether or not you think the specified mehtod of communcation using files is the most appropriate for such a system: justify answer by comparison with at least one other possible method

Inter-process communication(IPC) is about the mechanisms that operating system provides to allow the processes to manage shared data. Typically, applications can use IPC, categorized as clients and servers, where the client requests data and the server responds to client requests. Many applications are both clients and servers, as commonly seen in distributed computing. There are multiple approaches for IPC, such as file, socket, shared memory, pipe, message passing, etc. For this assignment, I do agree with using file for IPC is better approach among another. The reason is strongly related to the primary purpose of this distributed system, it is a news aggregator system which collects lots of news from creators and delivers the news to clients when they are all simulating under the local computer. It is the best way to communicate with different processes through files as well as intuitively way to receiving the news and sending the news. If we use other approach such as message passing, because our NA does not know what is the size for the news, it may pass lots of information into the buffer, at the same time, there is no other way to store those large information but in files. Moreover, if NA is still handling the current buffer, but there is another request coming, it may lose the information in buffer or cause a series delay in the system. That is why I think using file is a better approach for simulating this distributed program under the same local machine.

However, if the news creators and clients are from different machines, it is better to use socket to communicate between processes. Just like what we did for computer network and application, we can communicate different process under specific protocol, like TCP, UDP, etc.