Shared Memory and Message Queue

Software Design Document

Henry Ho

Xiaomei Ji

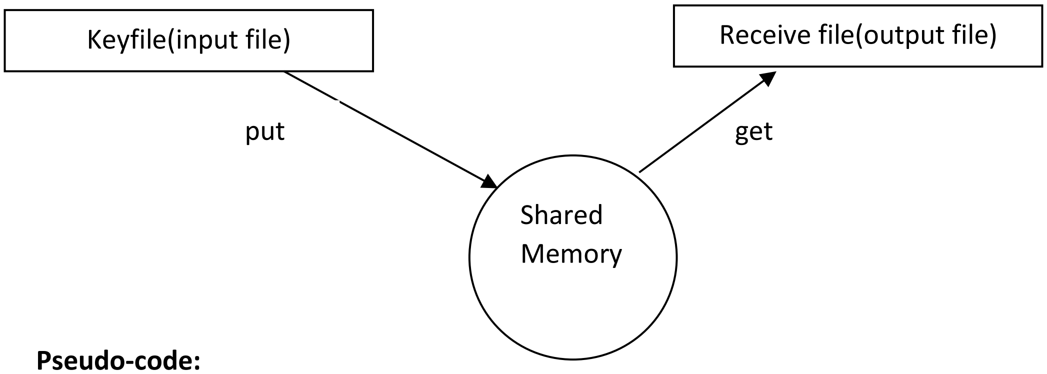
David Luong

June 15, 2018

I.Introduction

In this assignment our group use shared memory and message queues to implement an application which synchronously transfers files between two processes: sender and receiver.

II.Data Flow Diagram



III. Pseudo-code:

sender: this program shall implement the process that sends files to the receiver process.

```
Function init:

generate a key for shared memory

allocate a piece of shard memory

use a pointer to attach the shared memory which is set up by the receiver

create a message queue
```

```
Function send:


While (not end of the file)
       Read a number of bytes and store it in the shared memory


       Send a message to the receiver telling him the data is ready

       (using a message queue)


       Wait until the receiver sends a message telling he finished saving the
```

```
        memory chunk.

Once reach the end of the file, send a message to the receiver with the size
field set to 0. This signals to the receiver that no more message to send.
```

```
Function cleanup:
Close the file, detach shared memory, and exit
```

receiver: this program shall implement the process that receives files from the sender process. It shall perform the following sequence of steps:

```
function signal:

install a signal handler
```

```
function: init

generate a key for shared memory
allocate a piece of shard memory
use a pointer to attach the shared memory
create a message queue
```

```
Function mainloop:

Open the file for writing

Receive the message and get the message size

While (the message size is not 0)
        Read the number of bytes from the shared and save them to the file
        Send message to the sender that successful reception and saving of data
```

```
Function cleanup:



Close the file, detach shared memory, and exit
```

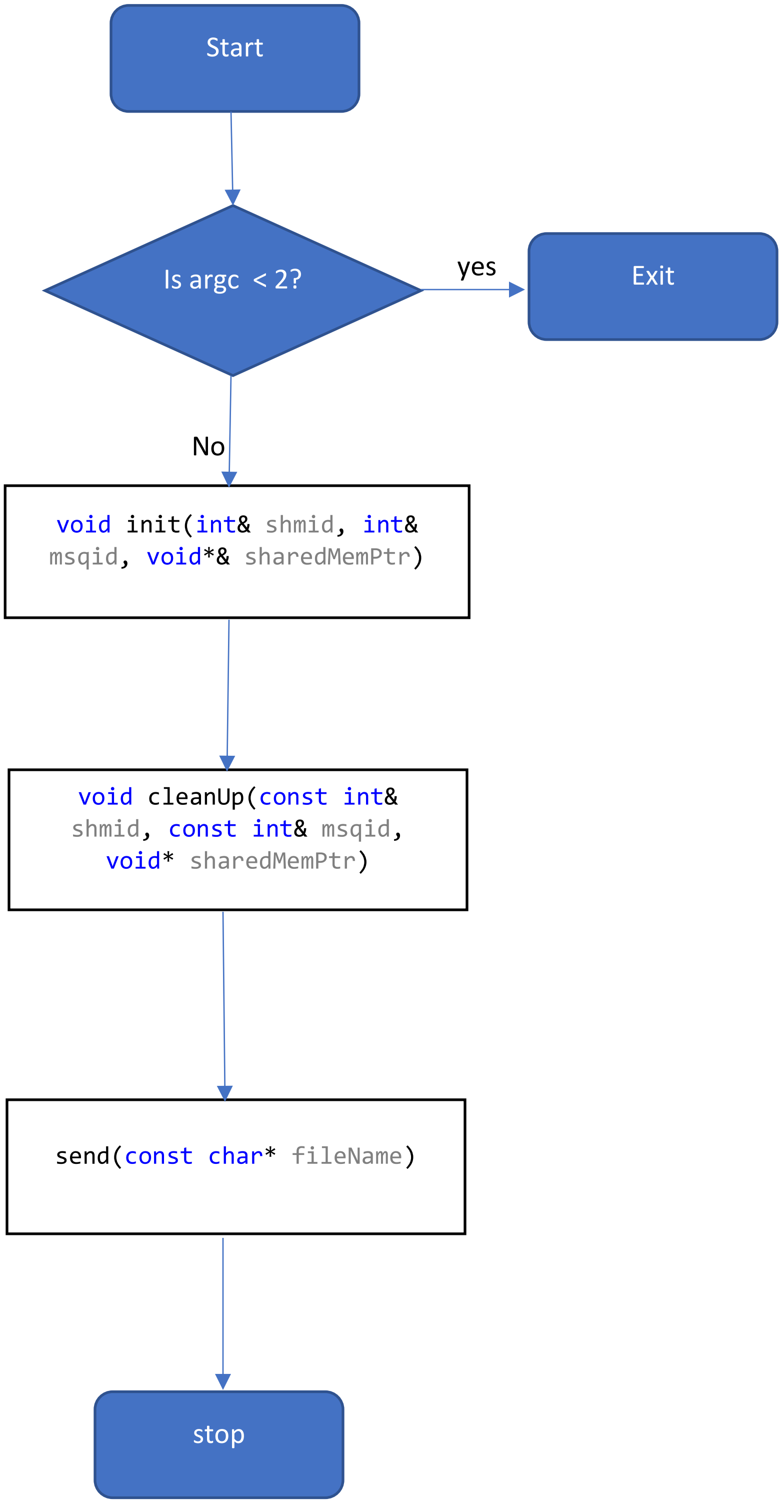
IV. Conclusion:

This assignment helps our group to get a better understanding of IPC principles, different IPC mechanisms using shared memory and using message queues, and gain hands-on experience using signals.

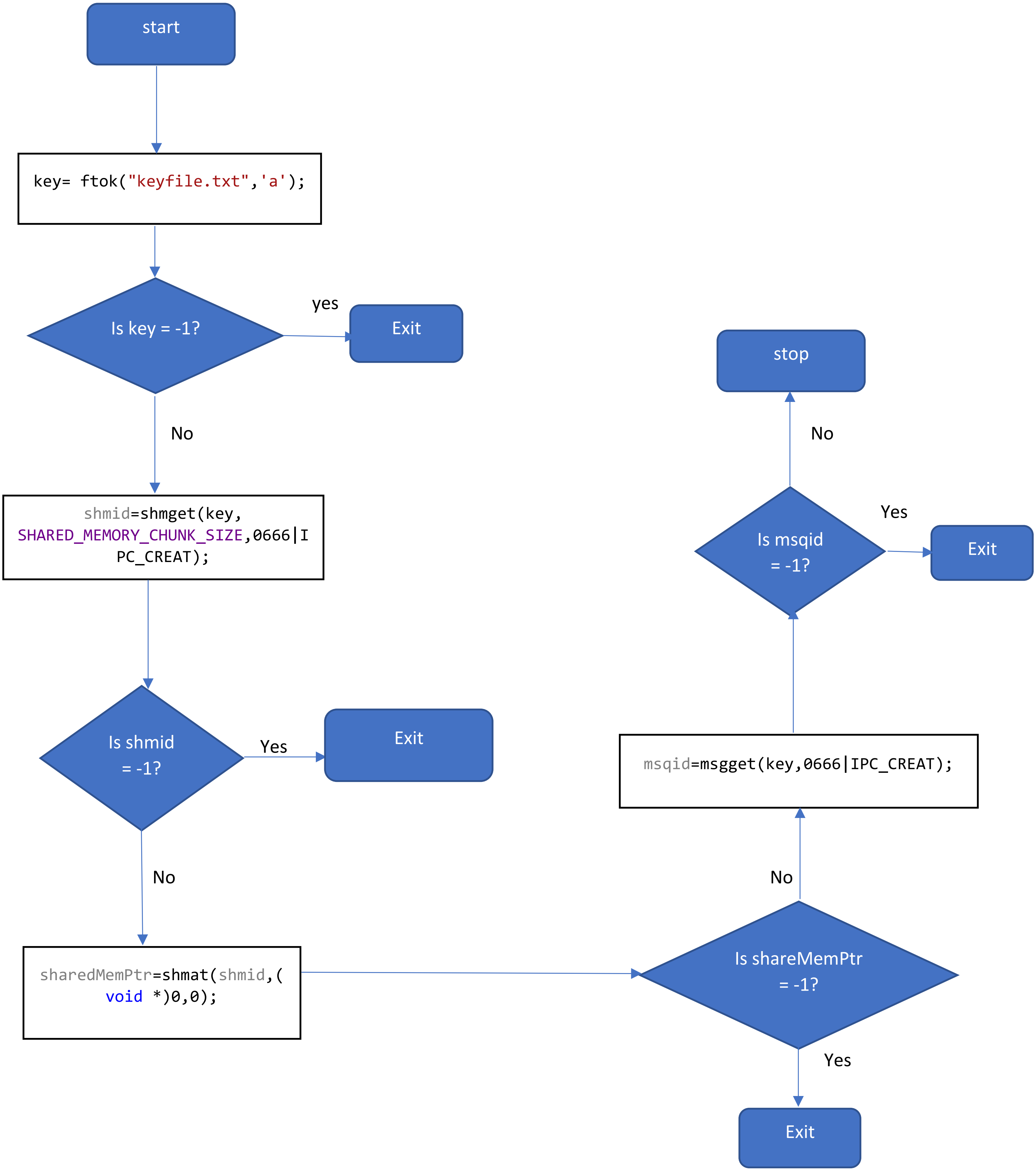
V. Flowchart Design:

I. Sender.cpp

1. Main function

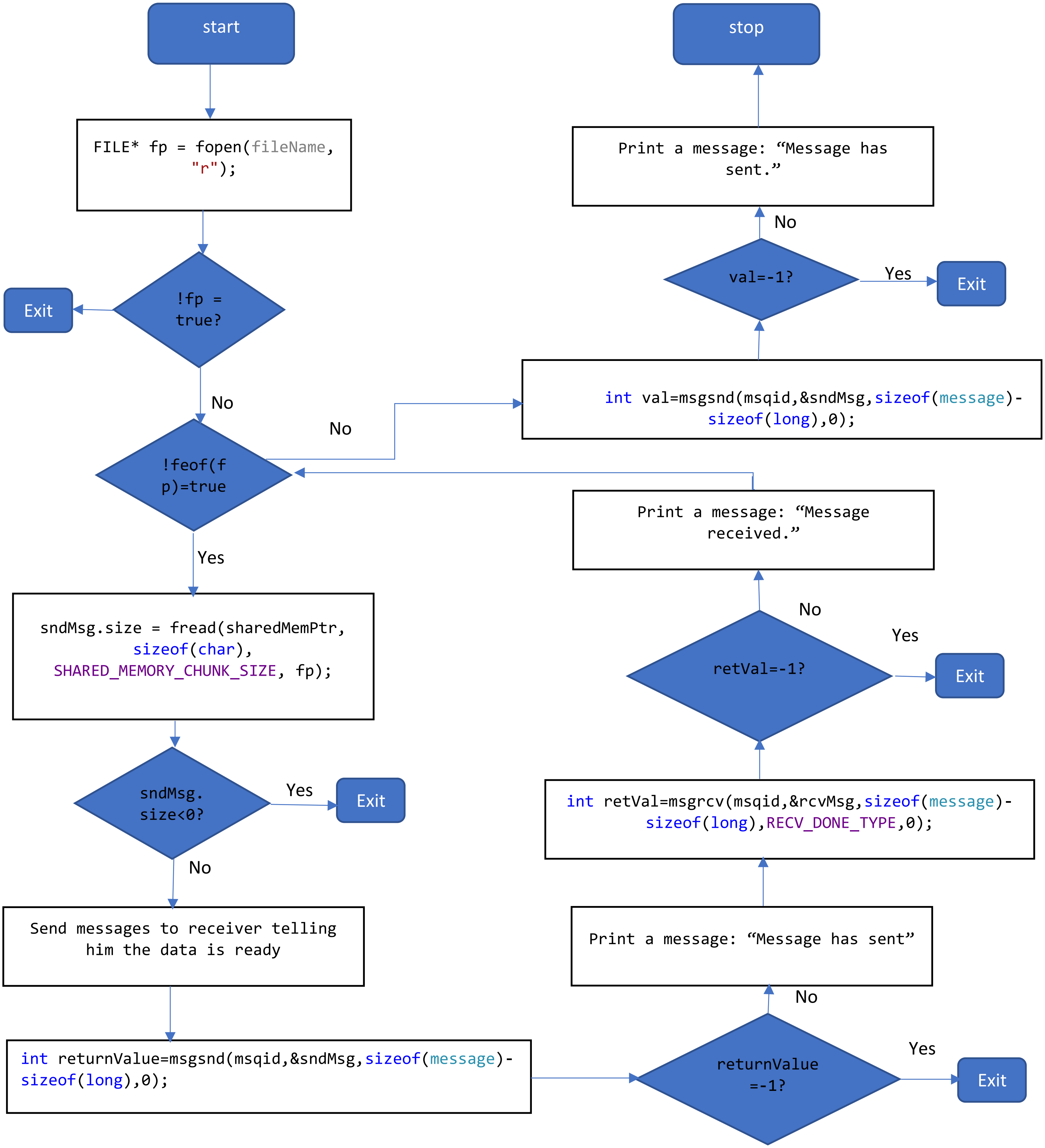

Start
Is argc < 2?
yes
Exit
No
void init(int& shmid, int& msqid, void*& sharedMemPtr)
void cleanUp(const int& shmid, const int& msqid, void* sharedMemPtr)
send(const char* fileName)
stop

2. Init function:

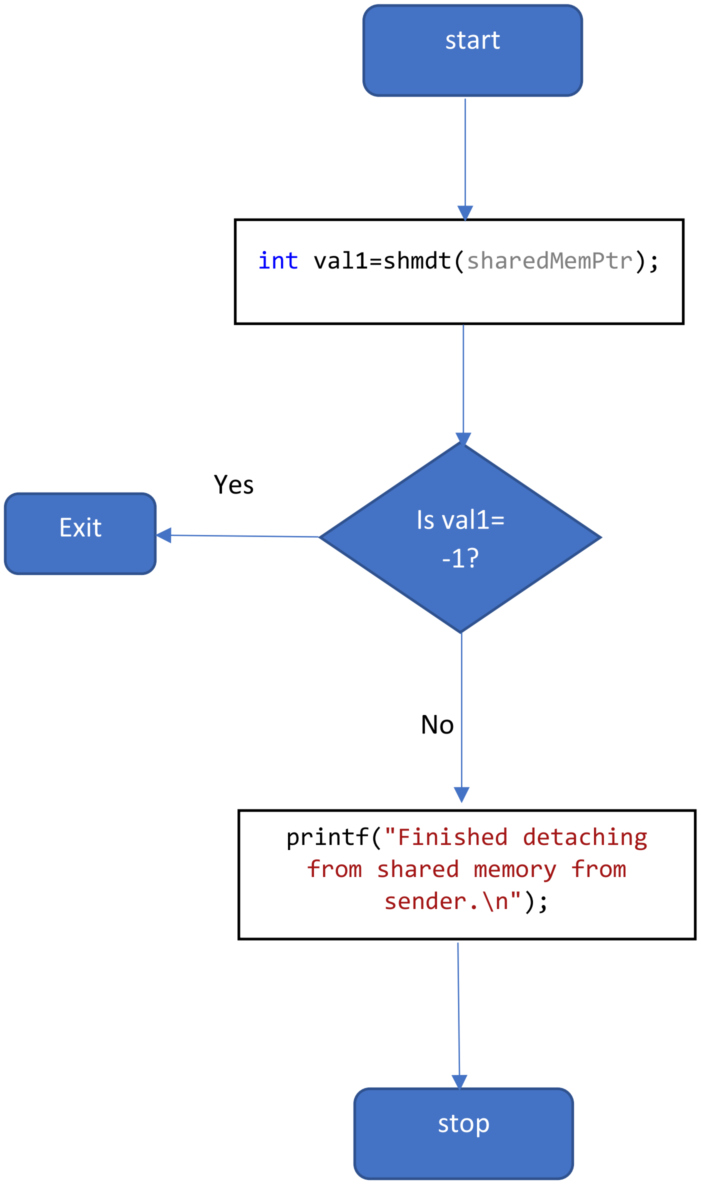


```
key= ftok("keyfile.txt",'a');
shmid=shmget(key, SHARED_MEMORY_CHUNK_SIZE,0666|IPC_CREAT);
sharedMemPtr=shmat(shmid,(void *)0,0);
msqid=msgget(key,0666|IPC_CREAT);
```

3. **Send function:**


start
FILE* fp = fopen(fileName, "r");
!fp = true?
Exit
No
!feof(fp)=true
No
Yes
sndMsg.size = fread(sharedMemPtr, sizeof(char), SHARED_MEMORY_CHUNK_SIZE, fp);
sndMsg.size<0?
Yes
Exit
No
Send messages to receiver telling him the data is ready
int returnValue=msgsnd(msqid,&sndMsg,sizeof(message)-sizeof(long),0);
returnValue =-1?
Yes
Exit
No
Print a message: "Message has sent"
int retVal=msgrcv(msqid,&rcvMsg,sizeof(message)-sizeof(long),RECV_DONE_TYPE,0);
retVal=-1?
Yes
Exit
No
Print a message: "Message received."
int val=msgsnd(msqid,&sndMsg,sizeof(message)-sizeof(long),0);
val=-1?
Yes
Exit
No
Print a message: "Message has sent."
stop

4. Cleanup function:

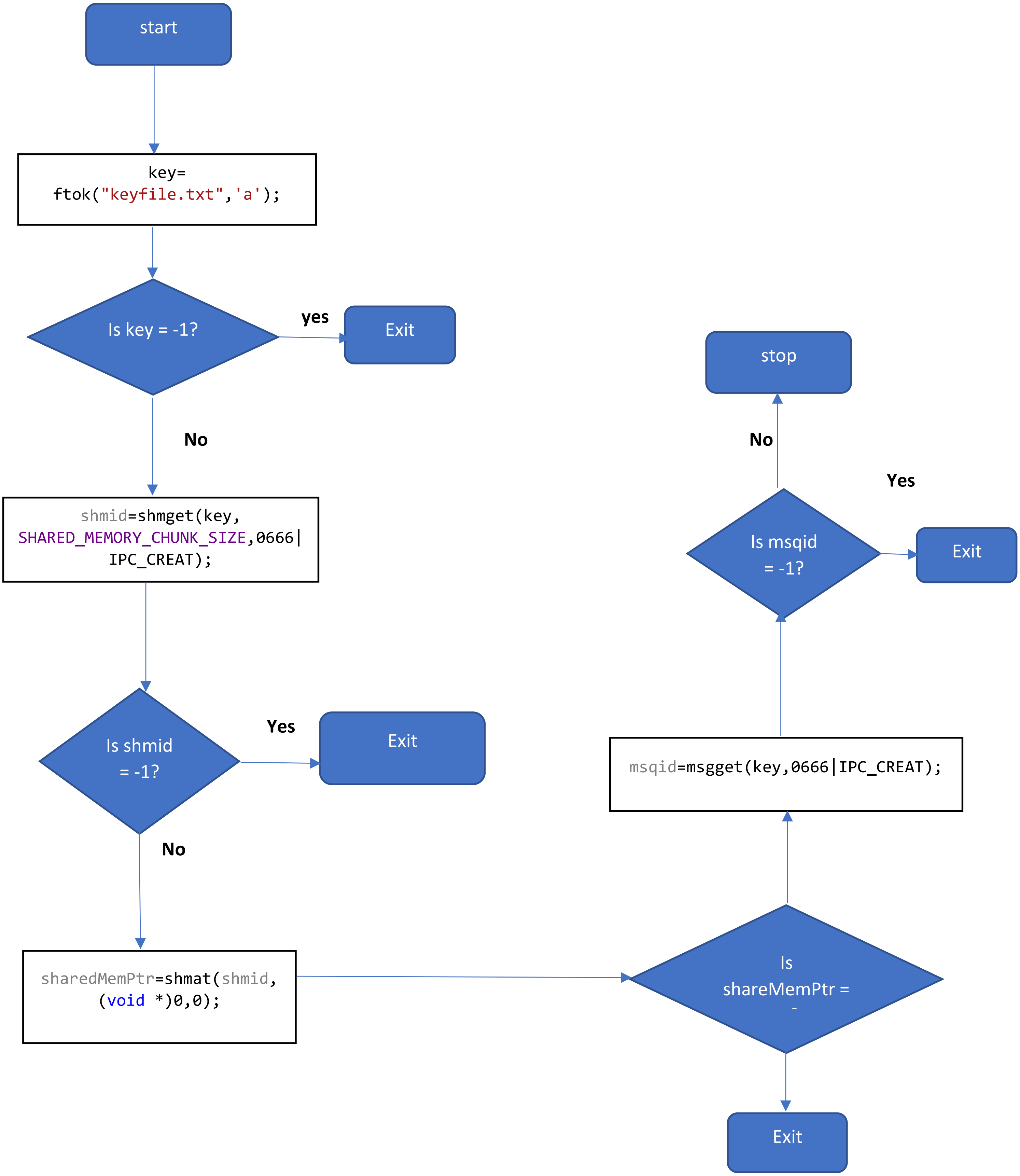


```
int val1=shmdt(sharedMemPtr);
```

```
printf("Finished detaching
from shared memory from
sender.\n");
```

II. **receiver.cpp**

1. **init function:**



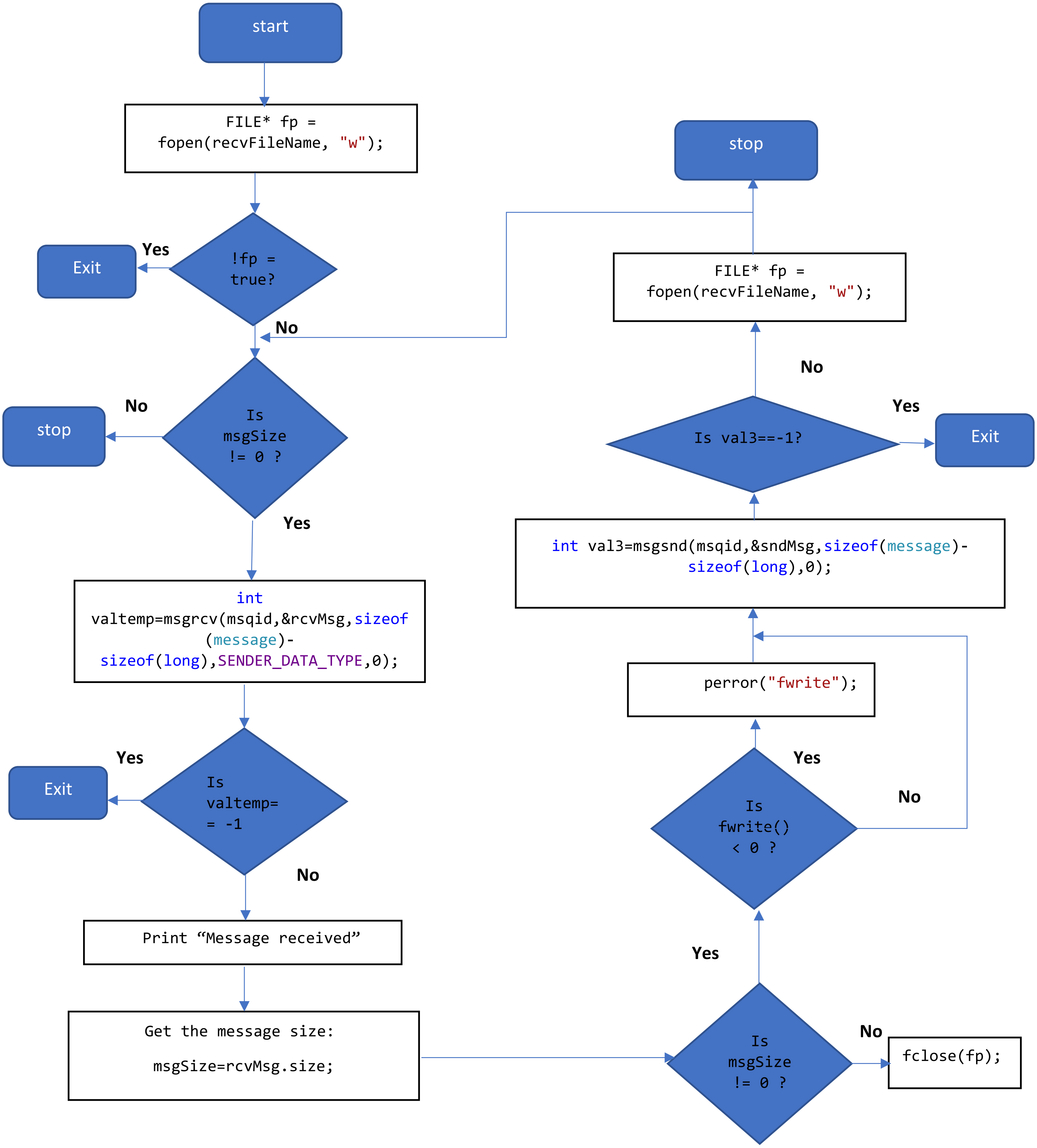
```
key=
ftok("keyfile.txt",'a');
```

```
shmid=shmget(key,
SHARED_MEMORY_CHUNK_SIZE,0666|
IPC_CREAT);
```

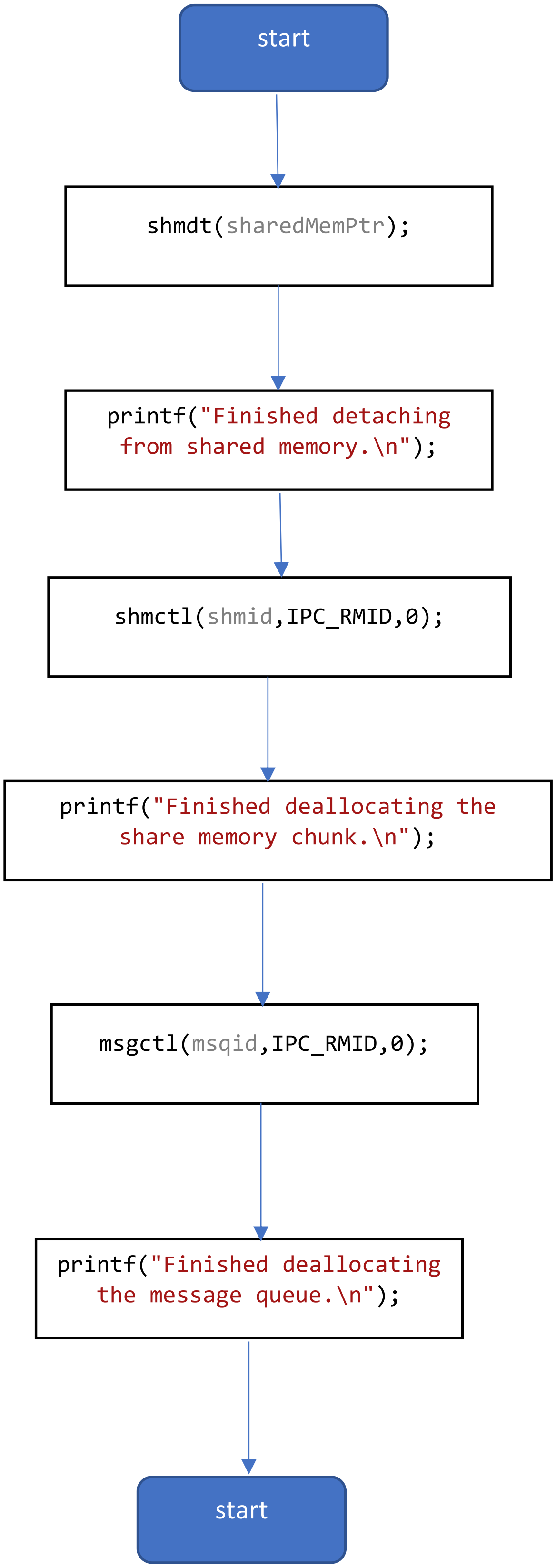
```
sharedMemPtr=shmat(shmid,
(void *)0,0);
```

```
msqid=msgget(key,0666|IPC_CREAT);
```

2. mainloop function:

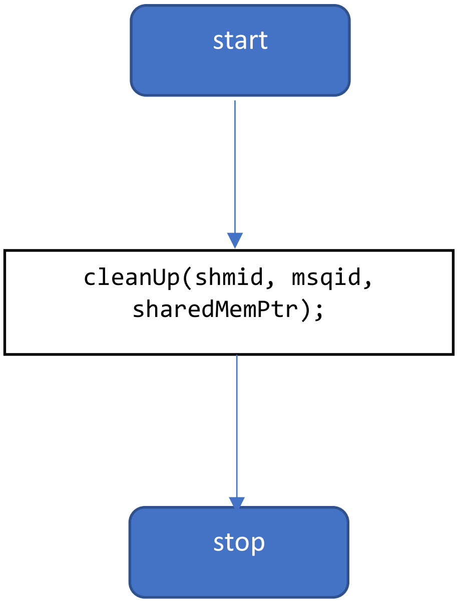


3. cleanup function:

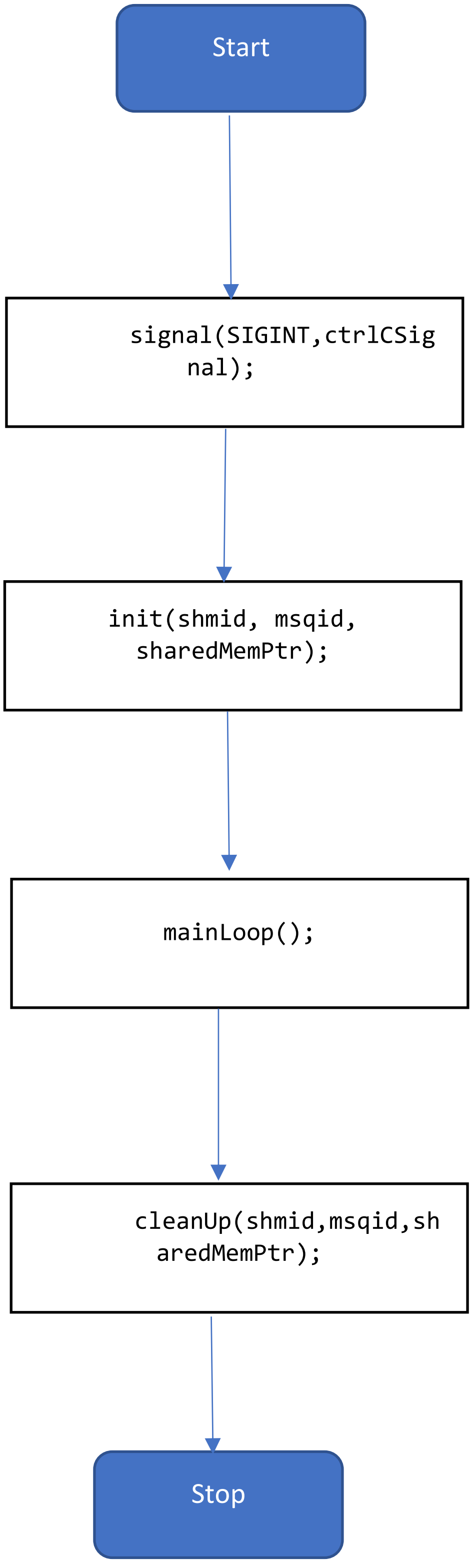


```
shmdt(sharedMemPtr);
printf("Finished detaching from shared memory.\n");
shmctl(shmid,IPC_RMID,0);
printf("Finished deallocating the share memory chunk.\n");
msgctl(msqid,IPC_RMID,0);
printf("Finished deallocating the message queue.\n");
```

4. void CtrlCsignal function:



5. main function:


Start
signal(SIGINT,ctrlCSignal);
init(shmid, msqid, sharedMemPtr);
mainLoop();
cleanUp(shmid,msqid,sharedMemPtr);
Stop