Nucleus Message Passing

Name: Yuchang Chen

Goal:

There would be a communication mechanism in the JNachos Operating System. With this principle, the message could be sent or received between the processes. There would be 4 operating in the communication:

send message (receiver, message, buffer),
wait message (sender, message, buffer),
send answer (result, answer, buffer),
wait answer (result, answer, buffer).

Required features and Stretch features:

- Here is the implement idea of the 4 functions:

  Send message copies a message into the first available buffer within the pool and delivers it in the queue of a named receiver.

  Wait message delays the requesting process until a message arrives in its queue.

  Send answer copies an answer into a buffer in which a message has been received and delivers it in the queue of the original sender.

  Wait answer delays the requesting process until an answer arrives in a given buffer.

- Message Buffer and Message Queue

  In every process, there would be many buffers and a queue for the message to move. The buffers are used to contain the message and make them remove to the queue or answer from the receiver.

  The queue in every process is used to handle the receiving operating. It follows the FIFO principle to handle the message and give out the answer to the queue of the receiver.

- The construction of the Message/Answer

  In the message, there would be Process Id of the sender, content of the message and timestamp and so on. This is the part of the stretch features. We could add more attributes into the Message Class.

  When sending the message, the message would be stored into a buffer, and then, the buffer is removed to the queue for the receiver dealing with them. The transition way of answer is similar as the way of message.

Implement:

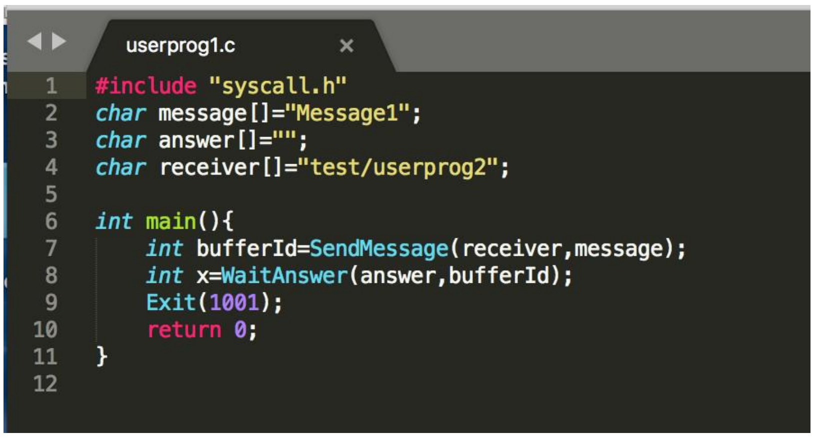
I add 4 system calls in the SystemHandler.java:

SC_SendMessage, SC_SendAnswer, SC_WaitMessage, SC_WaitAnswer, each of them is a system call in the C file.

We also add a Buffer class and a BufferPool class. The number of the buffers in the BufferPool is fixed and we can use the buffer to dequeue and enqueue and let them to be sent and received.

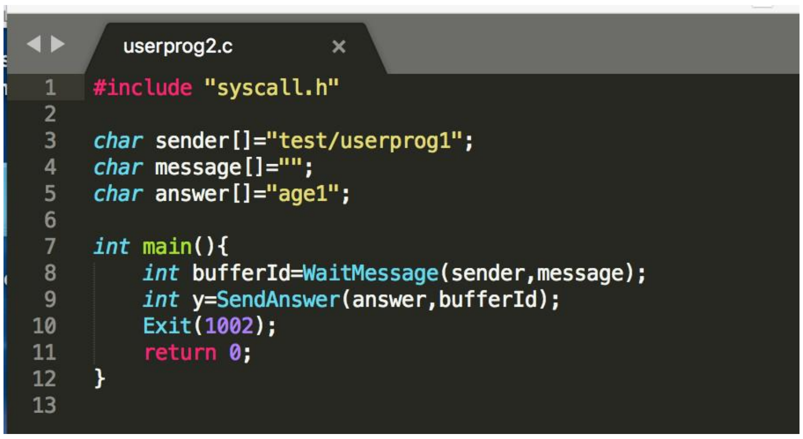
The format of the user program is like this:

Userprog1:



```c
#include "syscall.h"
char message[]="Message1";
char answer[]="";
char receiver[]="test/userprog2";

int main(){
    int bufferId=SendMessage(receiver,message);
    int x=WaitAnswer(answer,bufferId);
    Exit(1001);
    return 0;
}
```

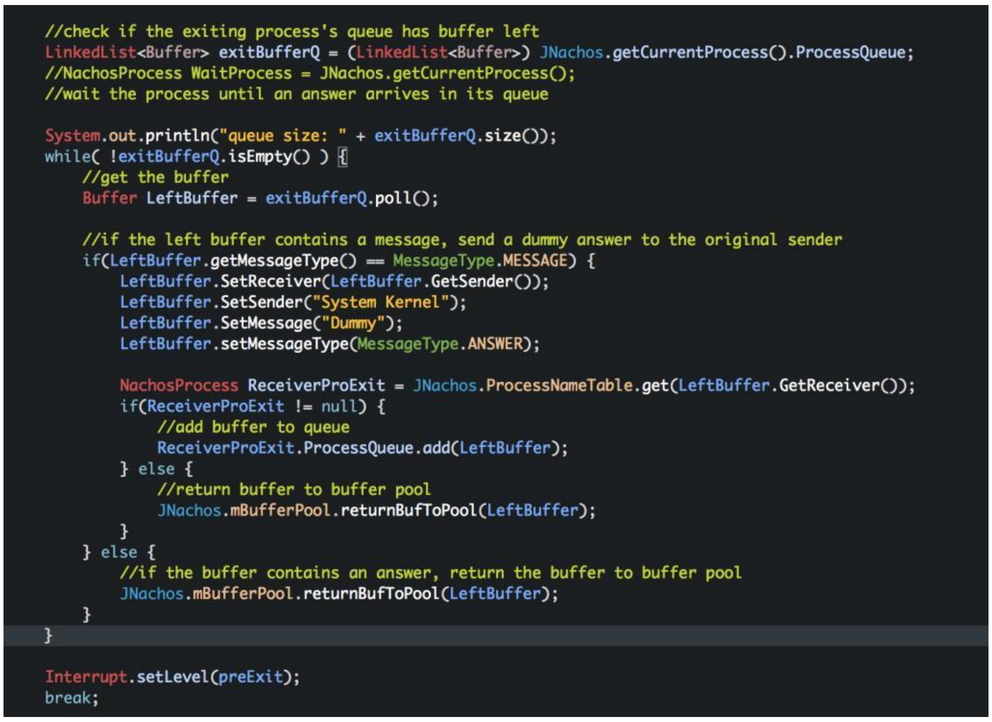
Userprog2:



```c
#include "syscall.h"

char sender[]="test/userprog1";
char message[]="";
char answer[]="age1";

int main(){
    int bufferId=WaitMessage(sender,message);
    int y=SendAnswer(answer,bufferId);
    Exit(1002);
    return 0;
}
```

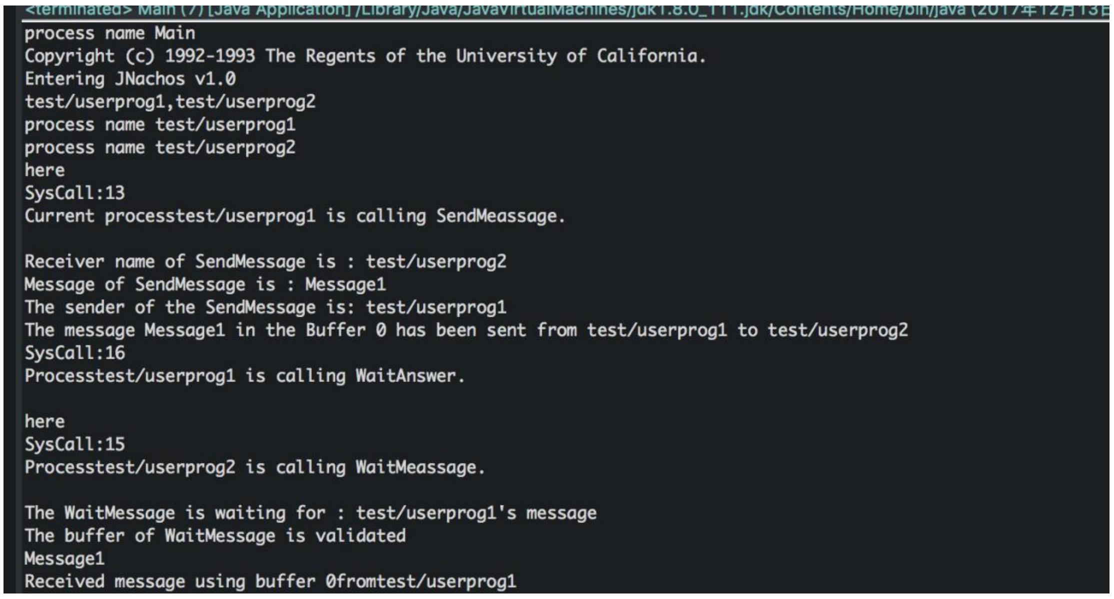
Each process has a queue to receive the buffer. When we wait for the message or the answer, the program should deal with the bufferin the queue.

Also, when the message sending terminate, the Exit system should send a dummy message to the Sender.

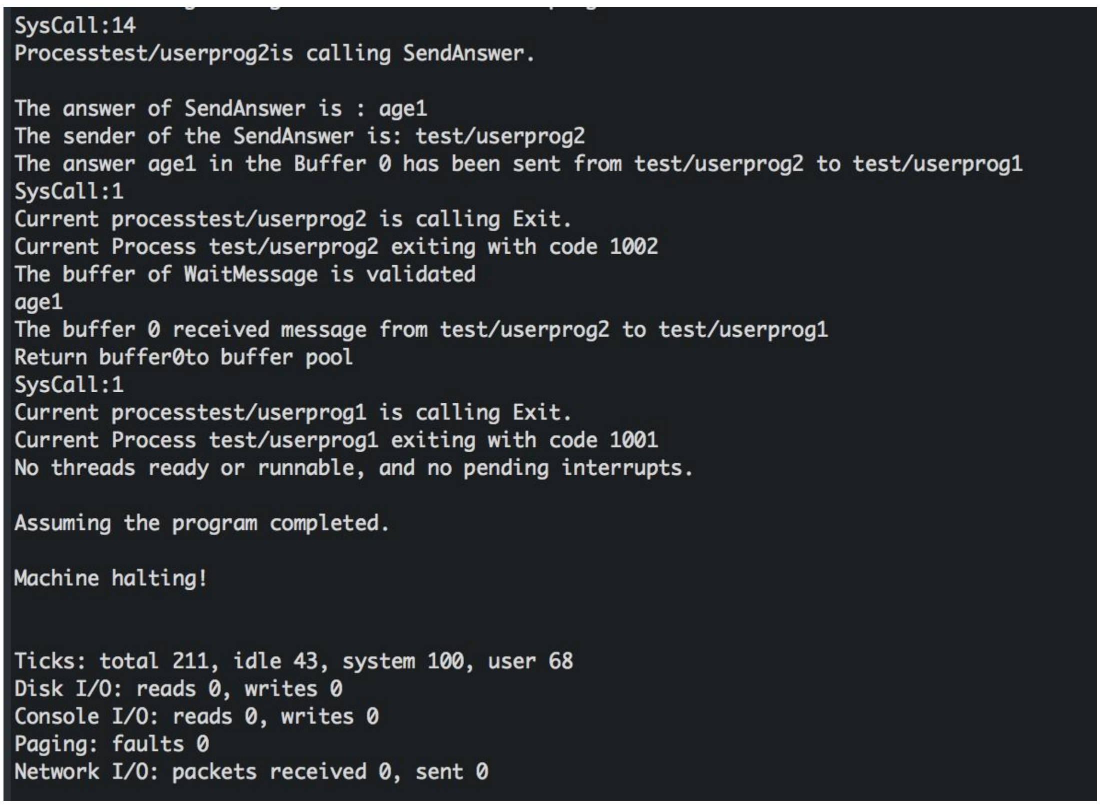


```java
//check if the exiting process's queue has buffer left
LinkedList<Buffer> exitBufferQ = (LinkedList<Buffer>) JNachos.getCurrentProcess().ProcessQueue;
//NachosProcess WaitProcess = JNachos.getCurrentProcess();
//wait the process until an answer arrives in its queue

System.out.println("queue size: " + exitBufferQ.size());
while( !exitBufferQ.isEmpty() ) {
    //get the buffer
    Buffer LeftBuffer = exitBufferQ.poll();

    //if the left buffer contains a message, send a dummy answer to the original sender
    if(LeftBuffer.getMessageType() == MessageType.MESSAGE) {
        LeftBuffer.SetReceiver(LeftBuffer.GetSender());
        LeftBuffer.SetSender("System Kernel");
        LeftBuffer.SetMessage("Dummy");
        LeftBuffer.setMessageType(MessageType.ANSWER);

        NachosProcess ReceiverProExit = JNachos.ProcessNameTable.get(LeftBuffer.GetReceiver());
        if(ReceiverProExit != null) {
            //add buffer to queue
            ReceiverProExit.ProcessQueue.add(LeftBuffer);
        } else {
            //return buffer to buffer pool
            JNachos.mBufferPool.returnBufToPool(LeftBuffer);
        }
    } else {
        //if the buffer contains an answer, return the buffer to buffer pool
        JNachos.mBufferPool.returnBufToPool(LeftBuffer);
    }
}

Interrupt.setLevel(preExit);
break;
```

Here is the result:



```
<terminated> Main (7) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_111.jdk/Contents/Home/bin/java (2017年12月13日
process name Main
Copyright (c) 1992-1993 The Regents of the University of California.
Entering JNachos v1.0
test/userprog1,test/userprog2
process name test/userprog1
process name test/userprog2
here
SysCall:13
Current processtest/userprog1 is calling SendMeassage.

Receiver name of SendMessage is : test/userprog2
Message of SendMessage is : Message1
The sender of the SendMessage is: test/userprog1
The message Message1 in the Buffer 0 has been sent from test/userprog1 to test/userprog2
SysCall:16
Processtest/userprog1 is calling WaitAnswer.

here
SysCall:15
Processtest/userprog2 is calling WaitMeassage.

The WaitMessage is waiting for : test/userprog1's message
The buffer of WaitMessage is validated
Message1
Received message using buffer 0fromtest/userprog1
```



```
SysCall:14
Processtest/userprog2is calling SendAnswer.

The answer of SendAnswer is : age1
The sender of the SendAnswer is: test/userprog2
The answer age1 in the Buffer 0 has been sent from test/userprog2 to test/userprog1
SysCall:1
Current processtest/userprog2 is calling Exit.
Current Process test/userprog2 exiting with code 1002
The buffer of WaitMessage is validated
age1
The buffer 0 received message from test/userprog2 to test/userprog1
Return buffer0to buffer pool
SysCall:1
Current processtest/userprog1 is calling Exit.
Current Process test/userprog1 exiting with code 1001
No threads ready or runnable, and no pending interrupts.

Assuming the program completed.

Machine halting!


Ticks: total 211, idle 43, system 100, user 68
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0
```