CIS657 Fall 2018
Assignment Disclosure Form

Programming Assignment #3

Name: Vaibhav Kumar (SUID: 689190843), Sayali Naval (SUID: 964389618), Gauri Amberkar (SUID: 710146260)

1. Did you consult with anyone other than instructor or TA/grader on parts of this assignment?
   If Yes, please give the details.
   No

2. Did you consult an outside source such as an Internet forum or a book on parts of this assignment?
   If Yes, please give the details.
No

We assert that, to the best of our knowledge, the information on this sheet is true.

Signature:__Vaibhav Kumar, Sayali Naval, Gauri Amberkar_____

Date :__12/07/2018

Programming Assignment 3

CIS 657

Nucleus Message Passing

Final Report

Developed By:

Gauri Amberkar (*710146260)*
Sayali Naval (964389618)
Vaibhav Kumar (689190843)

Instructor:

Dr. Mina Jung

Date: December 7th, 2018

Introduction

In order to design any advance information system, it is necessary for a designer to have control over the mode of operation (batch processing, real time scheduling, priority scheduling etc.) on the operating system. Unfortunately, current operating systems don't provide that power to an application programmer or system designer which in turn is limiting the power of application programmer. This limitation also exposes weaknesses of an Operating System's support for multiprogramming.

Clearly, it is not practical to replace current operating system with a "new" one as it can have adverse effect on its support to existing applications running on it. Rather, the current operating system can be extended along with a *System Nucleus.*

System Nucleus

A system nucleus is an interrupt response program with complete control of input/output and storage protection. When extended with an existing operating system, system nucleus provides stronger support for multiprogramming, synchronization and coordination among processes by creating a hierarchy of processes for uniform parallel execution and cooperation. It does so by making use of fundamental set of primitives which allows dynamic creation of processes and control of hierarchy of processes and commands among them.

Processes

A system nucleus practices an "unambiguous" of the term process. All the processes can be categorized into two groups based upon execution and I/O viz. Internal and External processes.

Internal and External Processes

An internal process can be identified as a program executing in a specified storage location which means that, in order to communicate with it, other processes do not require an address, rather the communication can be done just by knowing the *unique process name* (Since, the storage location is known in advance).

An external process is an I/O of a given process identified by unique process name.

Our objective in this programming assignment is to implement communication between processes using the process names.

Objective

The primary objective that this assignment fulfills is the communication between two processes using nucleus message passing System. The Task in hand is to implement following system calls :

- SendMessage
- WaitMessage
- SendAnswer
- WaitAnswer

Design and Implementation

The Nucleus Message Passing system implemented in this assignment makes use of the asynchronous message passing communication technique where the sender sends a message to the receiver using common buffer. Following are the major components required for implementation of nucleus message passing.

1. **Buffer:** A buffer is an unordered map which contains a unique buffer ID (Key) and a List (Value) which contains the actual messages.
2. **Message Queue:** Message queue is also an unordered map which has the buffer ID as key and count (of message) as value.

   std::map<int,Buffer> messageQueue;*

3. **Process Pool:** Process pool contains all the processes that are currently running.
4. **Buffer Pool:** Buffer pool is a collection of buffer objects accessible to processes for communication.

   std::map<int,List<Buffer>* > bufferPool;*

Following are the implementation details: -

1. User programs prog1.c, prog2.c and prog3.c are created for testing.
2. System calls *SendMessage, WaitMessage, SendAnswer, WaitAnswer* are written along with the relevant invocation methods in exception.cc.
3. Necessary changes in Start.S are made to accommodate user programs prog1.c , prog2.c and prog3.c.
4. Method to handle Exit() has been written in the exception.cc file.
5. Class Buffer has been created with relevant data members and member functions in Buffer.h under test folder in nachos. The buffer class is as follows: -

6. In order to provide facility to configure and enforce a certain limit on the number of messages sent, following flag has been declared in the Kernel class.

   int limit;

7. The default value of the limit flag is set to 5 in the constructor of the Kernel class which can be overwritten by the limit value provided during runtime.
8. Event handler for all the system calls have been written in the ExceptionHandler method in the exception.cc files.

Testing and Output

To start testing any of the test scenarios mentioned below, please build and compile the nachos by executing the following commands in order provided.

1. cd nachos/coff2noff
2. make clean
3. make
4. cd ../code/test
5. make clean
6. make
7. cd ../ build.linux
8. make clean
9. make depend
10. make

Test Cases:

1. Sender sends one message and receiver receives it and sends one response.
   The same message buffer should be used for the same two processes to communicate.
   Command: ./nachos -x ../test/prog1 -x ../test/prog2
   Test Programs:

```
/*** prog1.c ***/
#include "syscall.h"

int
main()
{
        int buff;
        int bufferId = SendMessage("../test/prog2","Hello prog2",-1);
        WaitAnswer(1,"wait ans 1",bufferId);
        Halt();
        Exit(0);
}

/*** prog2.c ***/
#include "syscall.h"
```

```
int
main()
{
        int buffId = WaitMessage("../test/prog1","wait message 1", -1);
        SendAnswer(1,"Hello prog1",buffId);
        Halt();
        Exit(0);
}
```

Output:

```
snaval@lcs-vc-cis486:~/nachos/code/build.linux$ make
make: 'nachos' is up to date.
snaval@lcs-vc-cis486:~/nachos/code/build.linux$ ./nachos -x ../test/prog1 -x ../test/prog2
Send Message
---------------
limit 10
BufferId: 0
../test/prog1 Sending Message to ../test/prog2
Message: Hello prog2
Wait Answer
---------------
Wait Message
---------------
BufferId: 0
../test/prog2 Received Message from ../test/prog1
Message: Hello prog2
Send Answer
---------------
BufferId: 0
../test/prog2 Sending Answer to ../test/prog1
Answer: Hello prog1
Wait Answer
---------------
BufferId: 0
../test/prog1 Received Answer from ../test/prog2
Answer: Hello prog1
Machine halting!

Ticks: total 150, idle 0, system 90, user 60
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
snaval@lcs-vc-cis486:~/nachos/code/build.linux$
```

2. Sender sends two messages and receiver receives it and sends two responses.
   The same message buffer should be used for the same two processes to communicate.
   Command: ./nachos -x ../test/prog3 -x ../test/prog4
   Test Programs:

```
/*** prog3.c ***/
#include "syscall.h"
```

```
int
main()
{
    int buff;
    int bufferId = SendMessage("../test/prog4","Hello prog4",-1);
    WaitAnswer(1,"wait ans 3",bufferId);
    bufferId = SendMessage("../test/prog4","Where are you",bufferId);
    WaitAnswer(1,"wait ans 4",bufferId);
    Halt();
    Exit(0);
}

/*** prog4.c ***/
#include "syscall.h"

int
main()
{
    int buffId = WaitMessage("../test/prog3","wait message 3", -1);
    SendAnswer(1,"Hello prog3",buffId);
    buffId = WaitMessage("../test/prog3","wait message 4", buffId);
    SendAnswer(1,"I am right here!",buffId);
    Halt();
    Exit(0);
}
```

Output:

```
snaval@lcs-vc-cis486:~/nachos/code/build.linux$ ./nachos -x ../test/prog3 -x ../test/prog4
Send Message
---------------
limit 10
BufferId: 0
../test/prog3 Sending Message to ../test/prog4
Message: Hello prog4
Wait Answer
---------------
Wait Message
---------------
BufferId: 0
../test/prog4 Received Message from ../test/prog3
Message: Hello prog4
Send Answer
---------------
BufferId: 0
../test/prog4 Sending Answer to ../test/prog3
Answer: Hello prog3
Wait Answer
---------------
BufferId: 0
../test/prog3 Received Answer from ../test/prog4
Answer: Hello prog3
Send Message
---------------
limit 10
../test/prog3 Sending Message to ../test/prog4
Wait Answer
---------------
Wait Message
---------------
BufferId: 0
../test/prog4 Received Message from ../test/prog3
Message: Where are you
Send Answer
---------------
```

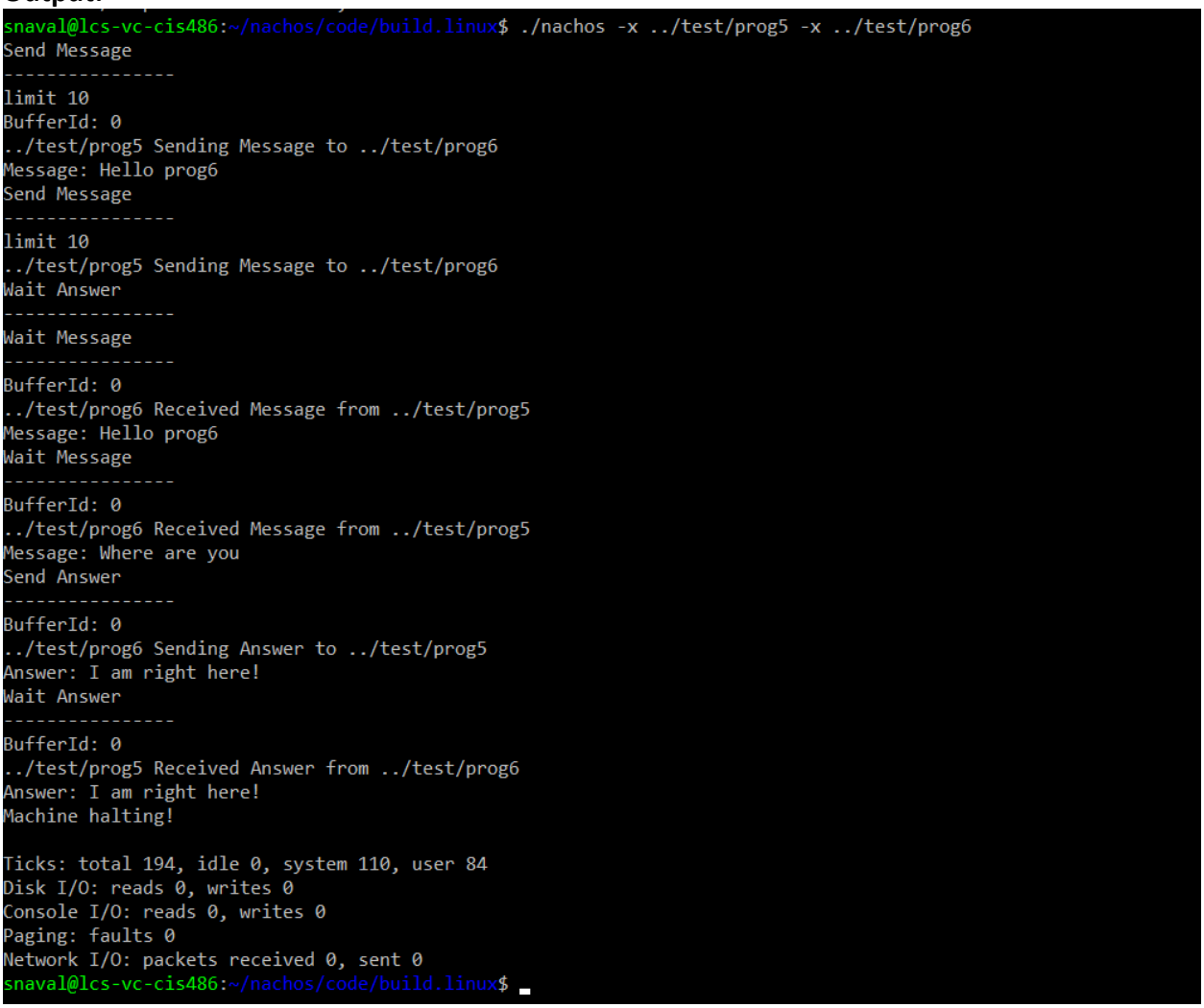
```
Message: Where are you
Send Answer
---------------
BufferId: 0
../test/prog4 Sending Answer to ../test/prog3
Answer: I am right here!
Wait Answer
---------------
BufferId: 0
../test/prog3 Received Answer from ../test/prog4
Answer: I am right here!
Machine halting!

Ticks: total 234, idle 0, system 130, user 104
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
snaval@lcs-vc-cis486:~/nachos/code/build.linux$
```

3. Sender sends two messages and receiver receives it and sends one response.
   The same message buffer should be used for the same two processes to communicate.
   Command: ./nachos -x ../test/prog5 -x ../test/prog6
   Test Programs:

```
/*** prog5.c ***/
#include "syscall.h"

int
main()
{
   int buff;
   int bufferId = SendMessage("../test/prog6","Hello prog6",-1);
   bufferId = SendMessage("../test/prog6","Where are you",bufferId);
   WaitAnswer(1,"wait ans 5",bufferId);
   Halt();
   Exit(0);
}

/*** prog6.c ***/
#include "syscall.h"

int
main()
{
   int buffId = WaitMessage("../test/prog5","wait message 5", -1);
```

```
    buffId = WaitMessage("../test/prog5","wait message 6", buffId);
    SendAnswer(1,"I am right here!",buffId);
    Halt();
    Exit(0);
}
```

Output:



4. Sender sends one message and then exits, and receiver receives it and then returns the buffer to buffer pool without sending a response.
   If a process dies while it has sent messages, then those messages should still be receivable by their co-communicators.
   Command: ./nachos -x ../test/prog7 -x ../test/prog8
   Test Programs:
   /*** prog7.c ***/
   #include "syscall.h"

```
int
main()
{
        int buff;
        int bufferId = SendMessage("../test/prog8","Hello prog8",-1);
        Exit(0);
}

/*** prog8.c ***/
#include "syscall.h"

int
main()
{
   int buffId = WaitMessage("../test/prog7","wait message 7", -1);
   SendAnswer(1,"I am right here!",buffId);
   Halt();
   Exit(0);
}
```

Output:

```
snaval@lcs-vc-cis486:~/nachos/code/build.linux$ ./nachos -x ../test/prog7 -x ../test/prog8
Send Message
---------------
limit 10
BufferId: 0
../test/prog7 Sending Message to ../test/prog8
Message: Hello prog8
Wait Message
---------------
BufferId: -1
../test/prog8 Received Message from ../test/prog7
Message: Hello prog8
Send Answer
---------------
Returning Buffer 0 to BufferPool
```

5. prog9 sends one message to prog11 and one message toprog10. Both the programs receive these messages and send one response each to prog9.
   Processes should be able to communicate with any arbitrary process running in the system, but you need to work out some way of identifying them.
   Command: ./nachos -x ../test/prog9 -x ../test/prog10 -x ../test/prog11
   Test Programs:

```
/*** prog9.c ***/
#include "syscall.h"

int
```

```
main()
{
    int buff;
    int bufferId = SendMessage("../test/prog10","Hello prog10",-1);
    WaitAnswer(1,"wait ans 9",bufferId);
    buff = SendMessage("../test/prog11","Hello prog11",-1);
    WaitAnswer(1,"wait ans 9",buff);
    Exit(0);
}

/*** prog10.c ***/
#include "syscall.h"

int
main()
{
        int buffId = WaitMessage("../test/prog9","wait message 9", -1);
        SendAnswer(1,"Hello prog9 from prog10",buffId);
        Exit(0);
}

/*** prog11.c ***/
#include "syscall.h"

int
main()
{
        int buffId = WaitMessage("../test/prog9","wait message 9", -1);
        SendAnswer(1,"Hello prog9 from prog11",buffId);
        Exit(0);
}
```

Output:

```
snaval@lcs-vc-cis486:~/nachos/code/build.linux$ ./nachos -x ../test/prog9 -x ../test/prog10 -x ../test/prog11
Send Message
---------------
limit 5
BufferId: 0
../test/prog9 Sending Message to ../test/prog10
Message: Hello prog10
Wait Answer
---------------
Wait Message
---------------
BufferId: 0
../test/prog10 Received Message from ../test/prog9
Message: Hello prog10
Send Answer
---------------
BufferId: 0
../test/prog10 Sending Answer to ../test/prog9
Answer: Hello prog9 from prog10
Wait Message
---------------
Wait Answer
---------------
BufferId: 0
../test/prog9 Received Answer from ../test/prog10
Answer: Hello prog9 from prog10
Send Message
---------------
limit 5
BufferId: 1
../test/prog9 Sending Message to ../test/prog11
Message: Hello prog11
Wait Answer
---------------
Wait Message
---------------
BufferId: 1
../test/prog11 Received Message from ../test/prog9
Message: Hello prog11
Send Answer
---------------
BufferId: 1
../test/prog11 Sending Answer to ../test/prog9
Answer: Hello prog9 from prog11
Wait Answer
---------------
BufferId: 1
../test/prog9 Received Answer from ../test/prog11
Answer: Hello prog9 from prog11
```

6. prog12 sends two messages to prog13 but message limit is set to 1, so it will not be able to send 2nd message to the receiver.
   A limit on the total number of messages sent by the process should be configurable and enforceable.
   Command: ./nachos -x ../test/prog12 -x ../test/prog13 -limit 1
   Test Programs:

```
/*** prog12.c ***/
#include "syscall.h"

int
main()
{
    int buff;
    int bufferId = SendMessage("../test/prog13","Hello prog13",-1);
    WaitAnswer(1,"wait ans 12",bufferId);
    bufferId = SendMessage("../test/prog13","Where are you",bufferId);
    WaitAnswer(1,"wait ans 13",bufferId);
```

```
    Halt();
    Exit(0);
}

/*** prog13.c ***/
#include "syscall.h"

int
main()
{
    int buffId = WaitMessage("../test/prog12","wait message 13", -1);
    SendAnswer(1,"Hello prog12",buffId);
    Halt();
    Exit(0);
}
```

Output:

```
snaval@lcs-vc-cis486:~/nachos/code/build.linux$ ./nachos -x ../test/prog12 -x ../test/prog13 -limit 1
Send Message
---------------
limit 1
BufferId: 0
../test/prog12 Sending Message to ../test/prog13
Message: Hello prog13
Wait Answer
---------------
Wait Message
---------------
BufferId: 0
../test/prog13 Received Message from ../test/prog12
Message: Hello prog13
Send Answer
---------------
BufferId: 0
../test/prog13 Sending Answer to ../test/prog12
Answer: Hello prog12
Wait Answer
---------------
BufferId: 0
../test/prog12 Received Answer from ../test/prog13
Answer: Hello prog12
Send Message
---------------
limit 1
Message limit exceeds
Wait Answer
---------------
Machine halting!

Ticks: total 192, idle 0, system 110, user 82
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
snaval@lcs-vc-cis486:~/nachos/code/build.linux$
```

Files Modified/Added

1. **prog1.c** (Added) Path: nachos/code/test/prog1.c
2. **prog2.c** (Added) Path: nachos/code/test/prog2.c
3. **prog3.c** (Added) Path: nachos/code/test/prog3.c
4. **prog4.c** (Added) Path: nachos/code/test/prog4.c
5. **prog5.c** (Added) Path: nachos/code/test/prog5.c
6. **prog6.c** (Added) Path: nachos/code/test/prog6.c
7. **prog7.c** (Added) Path: nachos/code/test/prog7.c
8. **prog8.c** (Added) Path: nachos/code/test/prog8.c
9. **prog9.c** (Added) Path: nachos/code/test/prog9.c
10. **prog10.c** (Added) Path: nachos/code/test/prog10.c
11. **prog11.c** (Added) Path: nachos/code/test/prog11.c
12. **prog12.c** (Added) Path: nachos/code/test/prog12.c
13. **prog13.c** (Added) Path: nachos/code/test/prog13.c
14. **buffer.h** (Added) Path: nachos/code/threads/buffer.h
15. **buffer.cc** (Added) Path: nachos/code/threads/buffer.cc
16. **thread.h** (Modified) Path: nachos/code/threads/thread.h
17. **thread.cc** (Modified) Path: nachos/code/threads/thread.cc
18. **MakeFile** (Modified) Path: nachos/code/Build.Linux/MakeFile
19. **Start.S** (Modified) Path: nachos/code/test/Start.s
20. **exception.cc** (Modified) Path: nachos/code/userprog/exception.cc
21. **Addrspace.cc** (Modified) Path: nachos/code/userprog/Addrspace.cc
22. **main.cc** (Modified) Path: nachos/code/threads/main.cc
23. **kernel.h**(Modified) Path: nachos/code/threads/kernel.h
24. **kernel.cc** (Modified) Path: nachos/code/threads/kernel.cc
25. **syscall.h** (Modified) Path: nachos/code/userprog/syscall.h

Code Snippets

```
#include "buffer.h"
string readString(int begLoc)
{
    int val;string value = "";
    while (true){
        kernel->machine->ReadMem(begLoc, 1, &val);
        if ((char)val == '\0'){
```

```
            break;
        }else{
            value += (char)val;
            begLoc++;
        }
    }
    return value;
}

Buffer *authComm(int bID, Thread *rcv)
{
    Buffer *buf = kernel->bufferPool[bID]->Front();
    if (buf != NULL){
        if (buf->getReceiver() == rcv){
            return buf;
        }
    }
}


void write(string message, int startLoc)
{
    while (true){
        if (message == "\0"){
            break;
        }
        else{
            int msg;
            stringstream(message) >> msg;
            kernel->machine->WriteMem(startLoc, 1, msg);
            startLoc++;
        }
    }
}

Buffer *takeBuf(int bufferId,string sender){
    std::map<int, Buffer *> messagesQueue = kernel->currentThread->messageQueue;
    if (!messagesQueue.empty()){
        if (messagesQueue.find(bufferId) != messagesQueue.end()){
            return messagesQueue[bufferId];
        }
        else if(sender != ""){
            vector<Buffer *> vec;
            for (map<int, Buffer *>::iterator it = messagesQueue.begin(); it !=
messagesQueue.end(); ++it){
```

```
                vec.push_back(it->second);
            }
            for (vector<Buffer *>::iterator it=vec.begin(); it!=vec.end(); it++){
                if ((*it)->getSender()->getName() == sender){
                    return (*it);
                }
            }
        }
    }
    return NULL;
}



      case SC_Exit:
        {
            IntStatus oldLevel = kernel->interrupt->SetLevel(IntOff);
            kernel->currentThread->Finish();
            (void) kernel->interrupt->SetLevel(oldLevel);
            return;
        }

        case Send_Message:
        {
            IntStatus oldLevel = kernel->interrupt->SetLevel(IntOff);
            cout << "Send Message " << "\n" << "----------------" << endl;
            cout << "limit " << kernel->limit << endl;
            int bufferId = kernel->machine->ReadRegister(6);
            string rec = readString((int)kernel->machine->ReadRegister(4));
            string msg = readString((int)kernel->machine->ReadRegister(5));
            int buffId;
            Thread *t = Thread::getByName(rec);
            if(t->getName() != rec){
                if (bufferId >= 0)
                    kernel->clearBuffer(bufferId);
                kernel->currentThread->Finish();
            }else{
                if (kernel->currentThread->getCount() <= kernel->limit){
                    if (bufferId == -1){
                        Buffer *buff = new Buffer();
                        buff->setSender(kernel->currentThread);
                        buff->setReceiver(t);
                        buffId = buff->getBufferID();
                        if (buffId == -1){
                            cout << "No buffer free. Please try later." << endl;
```

```
                            kernel->machine->WriteRegister(2, buffId);
                            {
                                kernel->machine->WriteRegister(PrevPCReg, kernel-
>machine->ReadRegister(PCReg));
                                kernel->machine->WriteRegister(PCReg, kernel-
>machine->ReadRegister(PCReg) + 4);
                                kernel->machine->WriteRegister(NextPCReg, kernel-
>machine->ReadRegister(PCReg) + 4);
                            }
                            (void)kernel->interrupt->SetLevel(oldLevel);
                            return;
                            break;
                        }
                        cout << "BufferId: " <<buffId <<endl;
                        kernel->currentThread->bufferMap.insert({buffId, rec});
                        buff->getMessages()->Append(msg);
                        List<Buffer *> *messageList = new List<Buffer *>();
                        messageList->Append(buff);
                        kernel->bufferPool[buffId] = messageList;
                        t->messageQueue[buffId] = buff;
                        kernel->machine->WriteRegister(2, buffId);

                        cout << kernel->currentThread->getName() << " Sending
Message to " << rec << endl;
                        cout <<  "Message: " << msg << endl;
                    }
                    else{
                        Buffer *buff2 = authComm(bufferId, t);
                        if (buff2 == NULL){
                            kernel->machine->WriteRegister(2, -1);
                            cout << "Malicious Process " << kernel-
>currentThread->getName() << " trying to communicate with process " << rec <<
endl;
                        }
                        else{
                            buff2->getMessages()->Append(msg);
                            t->messageQueue[bufferId] = buff2;
                            kernel->machine->WriteRegister(2, bufferId);
                            cout << kernel->currentThread->getName() << " Sending
Message to " << rec << endl;
                            cout <<  "Message: " << msg << endl;
                        }
                    }
                }
                else{
```

```
                    Buffer *buff3 = authComm(bufferId, t);
                    if (buff3 != NULL){
                        buff3->getMessages()->Append("This is dummy message");
                        t->messageQueue[bufferId] = buff3;
                        kernel->machine->WriteRegister(2, bufferId);
                    }
                    cout<<"Message limit exceeds"<<endl;
                }
            }
            {
                /* set previous programm counter (debugging only)*/
                kernel->machine->WriteRegister(PrevPCReg, kernel->machine-
>ReadRegister(PCReg));

                /* set programm counter to next instruction (all Instructions are
4 byte wide)*/
                kernel->machine->WriteRegister(PCReg, kernel->machine-
>ReadRegister(PCReg) + 4);

                /* set next programm counter for brach execution */
                kernel->machine->WriteRegister(NextPCReg, kernel->machine-
>ReadRegister(PCReg) + 4);
            }
            (void)kernel->interrupt->SetLevel(oldLevel);
            return;
            break;
        }

        case Wait_Message:
        {
            DEBUG(dbgSys, "Wait Message " << kernel->machine->ReadRegister(4)
<<"\n");
            cout << "Wait Message " << "\n" << "----------------" << endl;
            IntStatus oldLevel = kernel->interrupt->SetLevel(IntOff);
            string sender = readString((int)kernel->machine->ReadRegister(4));
            int bufferID = kernel->machine->ReadRegister(6);
            {
                if(Thread::getByName(sender) == NULL){
                    Buffer *buff5 = takeBuf(bufferID, sender);
                    if (buff5 != NULL && !buff5->getMessages()->IsEmpty()){
                        string message = buff5->getMessages()->RemoveFront();
                        kernel->machine->WriteRegister(2, buff5->getBufferID());
                        kernel->currentThread->mcount++;
                        cout << "BufferId: " << bufferID << endl;
```

```
                        cout << kernel->currentThread->getName() << " Received
Message from " << sender << endl;
                        cout << "Message: " << message << endl;
                    }
                    else{
                        if (bufferID >= 0)
                            kernel->clearBuffer(bufferID);
                        kernel->currentThread->Finish();
                        kernel->machine->WriteRegister(2, -1);
                        cout << "Sender " << sender << " does not exist"<<endl;
                        cout << "Sending dummy message to process " << kernel-
>currentThread->getName() << endl;
                    }
                    {
                        kernel->machine->WriteRegister(PrevPCReg, kernel-
>machine->ReadRegister(PCReg));
                        kernel->machine->WriteRegister(PCReg, kernel->machine-
>ReadRegister(PCReg) + 4);
                        kernel->machine->WriteRegister(NextPCReg, kernel-
>machine->ReadRegister(PCReg) + 4);
                    }
                    (void)kernel->interrupt->SetLevel(oldLevel);
                    return;
                    break;
                }else{
                    Thread *s = Thread::getByName(sender);
                    std::map<int, string>::iterator it;
                    for (it = s->bufferMap.begin(); it != s->bufferMap.end();
++it){
                        if (it->second == kernel->currentThread->getName()){
                            bufferID = it->first;
                            break;
                        }
                    }
                    Buffer *buff4 = takeBuf(bufferID, sender);
                    if (buff4 != NULL && !buff4->getMessages()->IsEmpty()){
                        cout << "BufferId: " << bufferID << endl;
                        string message = buff4->getMessages()->RemoveFront();
                        kernel->machine->WriteRegister(2, buff4->getBufferID());
                        kernel->currentThread->mcount++;
                        cout << kernel->currentThread->getName() << " Received
Message from " << sender << endl;
                        cout << "Message: " << message << endl;
                        {
```

```
                            kernel->machine->WriteRegister(PrevPCReg, kernel-
>machine->ReadRegister(PCReg));
                            kernel->machine->WriteRegister(PCReg, kernel-
>machine->ReadRegister(PCReg) + 4);
                            kernel->machine->WriteRegister(NextPCReg, kernel-
>machine->ReadRegister(PCReg) + 4);
                        }
                        (void)kernel->interrupt->SetLevel(oldLevel);
                        return;
                        break;
                    }
                    else{
                        kernel->currentThread->Yield();
                    }
                }
            }
            (void)kernel->interrupt->SetLevel(oldLevel);
            return;
            break;
        }
        case Send_Answer:
        {
            DEBUG(dbgSys, "Send Answer " << kernel->machine->ReadRegister(4) <<
"\n");
            cout << "Send Answer " << "\n" << "-----------------" << endl;
            IntStatus oldLevel = kernel->interrupt->SetLevel(IntOff);

            int bufferID = kernel->machine->ReadRegister(6);
            int result = kernel->machine->ReadRegister(4);
            string answer = readString((int)kernel->machine->ReadRegister(5));
            Buffer *buff6 = kernel->currentThread->messageQueue[bufferID];
            if (buff6 != NULL){
                Thread *receiver = buff6->getSender();
                if (receiver != NULL && Thread::IsTAlive(receiver-
>getThreadId())){
                    if (receiver->messageQueue[bufferID] == NULL){
                        Buffer *buff = new Buffer(bufferID);
                        buff->setSender(kernel->currentThread);
                        buff->setReceiver(receiver);
                        buff->getMessages()->Append(answer);
                        kernel->bufferPool[bufferID]->Append(buff);
                        receiver->messageQueue[bufferID] = buff;
                    }
                    else{
```

```
                        receiver->messageQueue[bufferID]->getMessages()-
>Append(answer);
                    }
                    kernel->machine->WriteRegister(2, kernel->currentThread-
>getThreadId());
                    kernel->currentThread->mcount--;
                    cout << "BufferId: " << bufferID << endl;
                    cout << kernel->currentThread->getName() << " Sending Answer
to " << receiver->getName() << endl;
                    cout << "Answer: " << answer << endl;
                }
                else{
                    if (bufferID >= 0)
                        kernel->clearBuffer(bufferID);
                    kernel->currentThread->Finish();

                    kernel->machine->WriteRegister(2, -1);
                    cout << "Receiver does not exist" << endl;
                }
            }
            kernel->currentThread->Yield();
            {
                kernel->machine->WriteRegister(PrevPCReg, kernel->machine-
>ReadRegister(PCReg));

                kernel->machine->WriteRegister(PCReg, kernel->machine-
>ReadRegister(PCReg) + 4);

                kernel->machine->WriteRegister(NextPCReg, kernel->machine-
>ReadRegister(PCReg) + 4);
            }
            (void)kernel->interrupt->SetLevel(oldLevel);
            return;
            break;
        }
        case Wait_Answer:
        {
            DEBUG(dbgSys, "Wait Answer " << kernel->machine->ReadRegister(4)  <<
"\n");
            cout << "Wait Answer " << "\n" << "----------------" << endl;
            IntStatus oldLevel = kernel->interrupt->SetLevel(IntOff);
            int bufferID = kernel->machine->ReadRegister(6);
            string sender = kernel->currentThread->getName();
            Buffer *buff7 = takeBuf(bufferID, sender);
            if(buff7 == NULL){
```

```
                kernel->currentThread->Yield();
            }
            else if(buff7->getMessages()->IsEmpty()){
                kernel->currentThread->Yield();
            }

            sender = "";
            if (kernel->currentThread->messageQueue[bufferID] != NULL)
                sender = kernel->currentThread->messageQueue[bufferID]-
>getSender()->getName();
            else{
                sender = kernel->bufferPool[bufferID]->Front()->getReceiver()-
>getName();
            }
            {
                if(Thread::getByName(sender) == NULL){
                    Buffer *buff7 = takeBuf(bufferID, sender);
                    if (buff7 != NULL && !buff7->getMessages()->IsEmpty()){
                        string message = buff7->getMessages()->RemoveFront();
                        kernel->machine->WriteRegister(2, kernel->currentThread-
>getThreadId());
                        cout << "Wait Answer " << "\n" << "----------------" <<
endl;
                        cout << "BufferId: " << bufferID << endl;
                        cout << kernel->currentThread->getName() << " Received
Answer from " << sender << endl;
                        cout << "Answer: " << message << endl;
                    }
                    else{
                        if (bufferID >= 0)
                            kernel->clearBuffer(bufferID);
                        kernel->currentThread->Finish();
                        write("This is dummy message", 0);
                        kernel->machine->WriteRegister(2, 0);
                        cout << "Dummy Message sent to " << kernel-
>currentThread->getName() <<" as Receiver " <<sender<<" does not exist" << endl;
                    }
                    break;
                }else{
                    buff7 = takeBuf(bufferID, sender);
                    if (buff7 != NULL && !buff7->getMessages()->IsEmpty()){
                        string message = buff7->getMessages()->RemoveFront();
                        kernel->machine->WriteRegister(2, kernel->currentThread-
>getThreadId());
```

```
                        cout << "Wait Answer " << "\n" << "----------------" <<
endl;
                        cout << "BufferId: " << bufferID << endl;
                        cout << kernel->currentThread->getName() << " Received
Answer from " << sender << endl;
                        cout << "Answer: " << message << endl;
                    }
                    else{
                        kernel->currentThread->Yield();
                    }
                }
            }
            {
                kernel->machine->WriteRegister(PrevPCReg, kernel->machine-
>ReadRegister(PCReg));

                kernel->machine->WriteRegister(PCReg, kernel->machine-
>ReadRegister(PCReg) + 4);

                kernel->machine->WriteRegister(NextPCReg, kernel->machine-
>ReadRegister(PCReg) + 4);
            }
            (void)kernel->interrupt->SetLevel(oldLevel);
            return;
            break;
        }
```

Code Snippet: exception.cc

```
#define Send_Message    16
#define Wait_Message    17
#define Send_Answer    18
#define Wait_Answer    19

int SendMessage(char* receiverName, char* message, int bufferId);
int WaitMessage(char* senderName, char* message, int bufferId);
int SendAnswer(int result, char* answer, int bufferId);
int WaitAnswer(int result, char* answer, int buffer);
```

Code Snippet: syscall.h

```
class Buffer
{
  public:
    Buffer();
```

```
    Buffer(int bufferId);
    List<string>* getMessages();
    bool isActive();
    void setAliveFlag(bool f);
    Thread* getSender();
    Thread* getReceiver();
    int getBufferID();
    void setSender(Thread *s);
    void setReceiver(Thread *r);

  private:
    int bufferId;
    bool isAlive;
    Thread *sender;
    Thread *receiver;
    List<string> *messages;

};
```

Code Snippet: buffer.h

```
Buffer::Buffer()
{
    isAlive = true;
    this->bufferId = kernel->messageBuffer->FindAndSet();
    this->messages = new List<string>();
}

Buffer::Buffer(int bufferId)
{
    isAlive = true;
    this->bufferId = bufferId;
    this->messages = new List<string>();
}

void Buffer::setSender(Thread *sender){
     this->sender = sender;
}

void Buffer::setReceiver(Thread *receiver){
     this->receiver = receiver;
}
```

```
List<string>* Buffer::getMessages()
{
    return messages;
}

bool Buffer::isActive()
{
    return this->isAlive;
}

void Buffer::setAliveFlag(bool f)
{
    this->isAlive = f;
}

Thread* Buffer::getSender()
{
    return this->sender;
}

Thread* Buffer::getReceiver()
{
    return this->receiver;
}

int Buffer::getBufferID()
{
    return this->bufferId;
}
```

Code Snippet: buffer.cc

```
std::map<int, string> bufferMap ;
    AddrSpace *space;     // User code this thread is running.
    std::map<int,Buffer*> messageQueue;
```

Code Snippet : thread.h

```
std::map<int,List<Buffer*>* > bufferPool;
    static Bitmap *messageBuffer;
    void clearBuffer(int bufferId);

int limit;
```

Code Snippet: kernel.h

```
limit = 5;

else if (strcmp(argv[i], "-limit") == 0) {
        ASSERT(i + 1 < argc);
        limit = atoi(argv[i + 1]);
        i++;
    }


void
Kernel::clearBuffer(int bufferId){
    messageBuffer->Clear(bufferId);
    if(bufferPool.find(bufferId) != bufferPool.end()) {
        cout << "Returning Buffer "<<  bufferId << " to BufferPool" << endl;
        if(!bufferPool.empty())
            bufferPool.erase(bufferId);
        }
    }
```

Code Snippet: kernel.cc

----------END OF REPORT----------