**Cristiam David Martin Jackson** **11.19.2018**
**Laboratory Nr. 4 (Report)**

# "Synchronizacja procesów z wykorzystaniem monitorów"
# "Synchronization of processes using monitors"

## Purpose
Write in c++ under linux environment, a system for collecting short text messages (between 16 and 128 characters), basing the synchronization of access to the resources on monitors.
The task should be capable of handle more than 5 clients located in threads, every client can set a message with different priority through a number (0 for high priority, 1 for regular priority).
Those messages should be allocated in one buffer (organized by priority and time), and taking care not to repeat any message.
Another subprocess (or thread) will read the messages.

## Overview
In the chat room are group of users, topics and messages.
**Group of users:** they send the text messages with different priority.

- High access level user: Their messages are more important than the messages sent by the regular-access-level users; it allows them to push messages to the head of the data structure.
- Regular access level user: Their messages have low priority in comparison with the user with elevated permission access.

## Solution
In order to storage the messages, a linked list will be implemented, it allows to save different types of data and we can add nodes relatively easily, in that way we can choose where a node should be allocated, based in their priority. For this task three pointers will be implemented, one at the head, the second pointing the end of the priority list (the next node contains a regular message) and the last one pointing the end of the list (which is the end of the normal messages). The code is detailed up next.

```
#include "l4_monitor.h"
#include <iostream>
#include <thread>
#include <unistd.h> //sleep
#include <string>

#include <stdlib.h>//rand
#include <time.h>
using namespace std;
int M=8;

class BUFFER : private Monitor
{
   struct Node
   {
        int key;
        string mssg;
        Node *next;
    };
// private member
private:
    Condition empty, full;
    Node *head, *tprior, *tail; //pointer to the first Node, tail of priority and tail
// public member
public:
    int nr_m; //number or items in the list
    // constructor
    BUFFER()
    {
        nr_m = 0;
        head = NULL; // set head to NULL
        tprior = NULL;
        tail= NULL;
    }
    // destructor
    ~BUFFER()
    {
        tprior=NULL;
        nr_m = 0;
        Node *next = head;
        while(next)
        {
            Node *temp = next;
            next = next->next;
            delete temp;
        }
    }
    // This prepends a new value at the beginning of the list
    void addm(string _mssg, int _key)
    {
        enter();
        //cout<<"nr "<<nr_m<<" M "<<M<<endl;
        while(nr_m == M)wait(full);
        Node *n = new Node();   // create new Node
        n->key = _key;
        n->mssg = _mssg;
        nr_m++;
            cout<<"Added "<<_mssg.substr(0,16)+" nr_m "<<nr_m<<endl;
        if (head == NULL) //if no nodes
        {
            n->next = NULL;
            head = tail= n;
            if (_key == 1)
                tprior=NULL;//if first message is regular
            else
            tprior = n;
        }//0 for high priority messages
        else if(head != NULL && _key == 0)
        {
            if (tprior == NULL) //if no prior messages
            {
                n->next = head;
                head = tprior = n;
            }
            else
            {
                n->next=tprior->next;
                tprior->next = n;
                tprior = n;
            }
            if (tprior->next == NULL)
                tail = n;
        }//1 for low priority messages
        else if(head != NULL && _key == 1 && tprior == NULL)
        {
            tail->next=n;
            n->next = NULL;
            tail = n;
        }
        else if(head != NULL && _key == 1 && tprior->next == NULL)
        {
            tprior->next=n;
            n->next = NULL;
            tail = n;
        }
        else if(head != NULL && _key == 1 && tprior->next != NULL)
        {
            tail->next=n;
            n->next = NULL;
            tail = n;
        }
        /*  cout<<"PRINTING CURRENT STORED MESSAGES"<<endl;
        fflush(stdout);
            print();
        fflush(stdout);*/
        if(nr_m == 1)signal(empty);
        leave();
    }
    // returns the first element in the list and deletes the Node.
    string popm()
    {
        enter();
        if(nr_m ==0) wait(empty);
        Node *n = head;
        string m = n->mssg;
        head = head->next;
        if (n==tprior)
            tprior = NULL;
        delete n;
        nr_m--;
        cout<<"Popped\n"<<m<<endl;
        if (nr_m == M-1)signal(full);
        leave();
        return m;
    }
    void print()
    {
        Node *curr;
        curr=head;
        while(curr != NULL)
        {
            cout << curr->mssg<<" "<<curr->key;
            if (curr == tprior)
                cout<<" <- tprior\n";
            else
                cout<<"\n";
            curr = curr->next;
        }
    }
    int bufsize()
    {
        return nr_m;
    }
} buf; //BUFFER OBJECT
//-=-=-=-=-=-=-=-=-=-=-= END CLASS BUFFER -=-=-=-=-=-=-=-=-=-=-=-=
```

Regarding to the monitor class, it is implemented together with the required semaphore and the condition.

```
#ifndef __l4_monitor_h
#define __l4_monitor_h

//#include <semaphore.h>



#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/types.h>
#include <iostream>

class Semaphore{
            int semid;
            public:
```

```
        Semaphore(key_t, int, int);
        ~Semaphore();
        int p();
        int v();
};

union semun{
        int val;
        struct semid_ds *buf;
        unsigned short int *array;
        struct seminf *__buf;
};

//DEFINITION

Semaphore::Semaphore(key_t key, int semFlags, int value){
        semid =  semget(key, 1, semFlags);
        union semun argument;
        unsigned short values[1];
        values[0] = value;
        argument.array = values;

        //cout<<"Semaphore created"<<endl;

        semctl(semid, 0, SETALL, argument);
}

Semaphore::~Semaphore(){
        union semun ignored_argument;

        //cout<<"Semaphore destroyed"<<endl;

        semctl(semid, 1, IPC_RMID, ignored_argument);

}

int Semaphore::p(){

        //cout<<"Semaphore p"<<endl;

        struct sembuf op[1];
        op[0].sem_num = 0;
        op[0].sem_op = -1;
        op[0].sem_flg = SEM_UNDO;

return semop(semid, op, 1);;
}

int Semaphore::v(){
        //cout<<"Semaphore v"<<endl;

        struct sembuf op[1];
        op[0].sem_num = 0;
        op[0].sem_op = 1;
        op[0].sem_flg = SEM_UNDO;

        return semop(semid, op, 1);
}
class Condition {
        friend class Monitor;
 private:
        Semaphore * s;//w
        int waitingCount; //threads waiting
 public://:s(123,0666 | IPC_CREAT,0)
        Condition(){
        s = new Semaphore(123,0666 | IPC_CREAT,0);
        waitingCount = 0;
        }
        void wait() {
        s->p();
        }
        bool signal() {
        if( waitingCount ) {
        --waitingCount;
        s->v();
        return true;
        }
        else
        return false;
        }
};

class Monitor {
 private:
        Semaphore * s;
 public://:s(123, 0666 | IPC_CREAT,1)
        Monitor(){
        s = new Semaphore(124, 0666 | IPC_CREAT,1);
        }
        void enter() {
        s->p();
        }
        void leave() {
        s->v();
        }
        void wait( Condition & cond ) {
        ++cond.waitingCount;
        leave();
        cond.wait();
```

```
            }
            bool signal( Condition & cond ) {
            if( cond.signal() ) {
            enter();
            return true;
            }
            else
            return false;
            }
};
#endif
```

```
// RANDOM TEXT GENERATOR
string textgen()
{
    string word="";
    string abc="ABCDEFGHIJKLMNOPQRSTUVWXYzabcdefghijklmnopqrstuvwxyz";
    int rnd=(rand()%112)+16;
    for(int i=0; i<rnd; i++)
    {
        word = word + abc[rand()%abc.length()];
    }
    return word;
}
// -= -= -= -= -= -= -= -= -= -= CLIENT
void client(int tid)
{
    string txt;
    int pri;//priority
    srand(time(NULL));
    for(int i=0; i<30; i++)
    //while(true)
    {
        //pri=(tid==0)?1:(tid==4)?0:(rand()%10)%2;
            pri=1;
//(rand()%10)%2;//if client = 0, it is high priority
//          pri=(tid==4)?1:(rand()%10)%2;//if client = 4, it is low priority
        txt = "Pri "+to_string(pri)+" - Cli "+to_string(tid+1)+" #"+to_string(i+1)+" "+textgen();
        buf.addm(txt,pri);
    }

}
// -= -= -= -= -= -= -= -= -= -= READER
void reader(int tid)
{
//for(int i=0;i<20;i++)
    while(true)
    {
        buf.popm();
//if (buf.bufsize()==0)
//          break;
    }

//cout<<tid<<endl;

}

void fillbuffer(){
string txt;
    int pri;//priority
    srand(time(NULL));
for (int i=0; i<7;i++)
            {
            pri=0;
            txt = "Pri "+to_string(pri)+" - Cli "+to_string(6)+" #"+to_string(i+1)+" "+textgen();
            buf.addm(txt,pri);
            }
}
int main()
{
    thread t[6];
    fillbuffer();
    for (int i=0; i<6; ++i)
    {
        if (i<5)
            t[i] = thread(client,i);
        else
        {
            cout<<"\n-=-=-=-=-=-=-=-=-=-= Reader sleeping, press enter"<<endl;
            cin.ignore();
            cout<<"\n-=-=-=-=-=-=-=-=-=-= Reader start popping messages out from
buffer"<<endl;
            t[i] = thread(reader,i);
        }
    }
    for (int i=0; i<5; ++i)
    {
        t[i].join();//pauses until they finish
    }
    sleep(1);
    cout<<"Buffer is empty"<<endl;
    //t[5].detach();
    t[5].join();
    return 0;
}
```

**TESTING**

The next tests where performed:

1. FULL BUFFER (next message is low priority)
2. FULL BUFFER (next message is high priority)
3. Producers generating only low priority messages
4. Producers generating only high priority messages
5. Buffer with N-1 low priority messages and next messages are high priority
6. Buffer with N-1 high priority messages and next messages are low priority

The code implemented for every case was shown below, for every test some variables where modified.

- FULL BUFFER (sending a low priority message to the full buffer): The number of spaces in buffer is **8**, the messages' priorities were randomly generated, the monitor displays the priority "Pri" 0 for high, 1 for low, the producer who wrote the message "Cli" (there were five producers, denoted from 1 to 5), and the number of messages "#" send by the client (producer). Once the buffer is full, the monitor blocks all processes until there is one process that pop a message from the buffer, it is shown that the first message popped after the signal was a priority-0 one, as expected.

```
cristiam@cristiam-ubuntu:~/Desktop/cdmj$ ./fullbuffer0

-=-=-=-=-=-=-=-=-=-= Reader sleeping, press enter
Added Pri 0 - Cli 1 #1 nr_m 1
Added Pri 0 - Cli 3 #1 nr_m 2
Added Pri 1 - Cli 2 #1 nr_m 3
Added Pri 1 - Cli 5 #1 nr_m 4
Added Pri 0 - Cli 4 #1 nr_m 5
Added Pri 0 - Cli 1 #2 nr_m 6
Added Pri 0 - Cli 3 #2 nr_m 7
Added Pri 0 - Cli 2 #2 nr_m 8


-=-=-=-=-=-=-=-=-=-= Reader start popping messages out from buffer
Popped
Pri 0 - Cli 1 #1
AQqLMyPLAPvTWEHKoFnFhxoLzzpnaScunGVoDOeOvXRtqXLaKUQsjmxzdAcCqFgEG
bpdlGDfVjs
Added Pri 1 - Cli 5 #2 nr_m 8
Popped
Pri 0 - Cli 3 #1 avAScPNkMWkaOYhOzGSfWSyhFvSWJLQvhJhqPEPczBzIizMoo
Added Pri 0 - Cli 1 #3 nr_m 8
Popped
Pri 0 - Cli 4 #1 hOHKozFGnFShfWxSyohLzFzvpSWnJaLSQvch
```

- FULL BUFFER (sending a high priority message to the full buffer):

```
cristiam@cristiam-ubuntu:~/Desktop/cdmj$ ./fullbuffer1

-=-=-=-=-=-=-=-=-=-= Reader sleeping, press enter
Added Pri 0 - Cli 2 #1 nr_m 1
Added Pri 0 - Cli 4 #1 nr_m 2
Added Pri 0 - Cli 2 #2 nr_m 3
Added Pri 0 - Cli 3 #1 nr_m 4
Added Pri 1 - Cli 1 #1 nr_m 5
Added Pri 1 - Cli 4 #2 nr_m 6
Added Pri 0 - Cli 2 #3 nr_m 7
Added Pri 0 - Cli 5 #1 nr_m 8


-=-=-=-=-=-=-=-=-=-= Reader start popping messages out from buffer
Popped
Pri 0 - Cli 2 #1
PfVgdvYjjmMnmkLzcHSLnlySHJQgBfaFywrfqmbUjFzsrRaxFSvIwfOzjkaKILlhuoAtSfBfhbf
PriDGzMPqIabSxrSSKzURHvNOLKFHERPAmWYly
Added Pri 0 - Cli 3 #2 nr_m 8
Popped
Pri 0 - Cli 4 #1 GUGPWJvUIRsMsXpsYICNRLEOLqkLbigJeOAcz
Added Pri 1 - Cli 1 #2 nr_m 8
```

- FULL BUFFER WITH ONLY LOW PRIORITY MESSAGES: In this case, only low priority messages are generated, the buffer is filled with messages and waits for reader to be active. Once reader start reading the messages, it reads them in order of arrival (because there are only low priority messages), up next we can appreciate the process.

```
cristiam@cristiam-ubuntu:~/Desktop/cdmj$ ./buffwithlowprior
-=-=-=-=-=-=-=-=-=-= Reader sleeping, press enter
Added Pri 1 - Cli 1 #1 nr_m 1
Added Pri 1 - Cli 2 #1 nr_m 2
Added Pri 1 - Cli 1 #2 nr_m 3
Added Pri 1 - Cli 3 #1 nr_m 4
Added Pri 1 - Cli 2 #2 nr_m 5
Added Pri 1 - Cli 1 #3 nr_m 6
Added Pri 1 - Cli 5 #1 nr_m 7
Added Pri 1 - Cli 3 #2 nr_m 8


-=-=-=-=-=-=-=-=-=-= Reader start popping messages out from buffer
Popped
Pri 1 - Cli 1 #1 rLdNoNCBdODDAwWcKbrLdNoNCBd
Added Pri 1 - Cli 2 #3 nr_m 8
Popped
Pri 1 - Cli 2 #1 rLdNoNCBdODDAwWcKEjTPdBongB
Added Pri 1 - Cli 3 #3 nr_m 8
Popped
```

```
Pri 1 - Cli 1 #2 rLdNoNCBdDAWcKEjPdBnBWByCUQ
Added Pri 1 - Cli 5 #2 nr_m 8
Popped
Pri 1 - Cli 3 #1
WLBpyfCmUhQzXTcXsbAeHLaypDnEjoehbIWaPBojiEkhApEsQgWXswVhzkNKYODwTo
gaLpfmhzXTcXsbAeHLaypDnEjoehbIWa
```

- **BUFFER FULL WITH HIGH PRIORITY MESSAGES:** They are allocated and read in the same income order. Only high priority messages are generated.

```
cristiam@cristiam-ubuntu:~/Desktop/cdmj$ ./buffwithhighprior

-=-=-=-=-=-=-=-=-=-= Reader sleeping, press enter
Added Pri 0 - Cli 1 #1 nr_m 1
Added Pri 0 - Cli 2 #1 nr_m 2
Added Pri 0 - Cli 1 #2 nr_m 3
Added Pri 0 - Cli 3 #1 nr_m 4
Added Pri 0 - Cli 4 #1 nr_m 5
Added Pri 0 - Cli 2 #2 nr_m 6
Added Pri 0 - Cli 3 #2 nr_m 7
Added Pri 0 - Cli 1 #3 nr_m 8


-=-=-=-=-=-=-=-=-=-= Reader start popping messages out from buffer
Popped
Pri 0 - Cli 1 #1 IBRVMVdQgfAhmtBRgiOTgnqpiIBRVM
Added Pri 0 - Cli 4 #2 nr_m 8
Popped
Pri 0 - Cli 2 #1 IBRVMVdQgfAhmtBRgiOTgnqpiESapj
Added Pri 0 - Cli 2 #3 nr_m 8
Popped
Pri 0 - Cli 1 #2 IBRVMVdQgfAhmtBRgignpiSpaXXfSL
Added Pri 0 - Cli 3 #3 nr_m 8
Popped
Pri 0 - Cli 3 #1 aICXzXApfgSMuLqLrYaKFpdWAhpaXFAWzNMDyPXBUSRbutvo
Added Pri 0 - Cli 5 #1 nr_m 8
```

- **Buffer, first 7 messages where low priority and the last is high priority:** In this case, the buffer was filled with seven no-priority messages and the next messages (30 per producer) are high priority, it means, that low-priority messages are going to be read only after the last producer writes its last message, in this case the experiment is shown with an interruption (in order to no extend the table), at the bottom of the test, the low-priority messages are appreciated.

```
cristiam@cristiam-ubuntu:~/Desktop/cdmj$ ./only1last0
Added Pri 1 - Cli 6 #1 nr_m 1
Added Pri 1 - Cli 6 #2 nr_m 2
Added Pri 1 - Cli 6 #3 nr_m 3
Added Pri 1 - Cli 6 #4 nr_m 4
Added Pri 1 - Cli 6 #5 nr_m 5
Added Pri 1 - Cli 6 #6 nr_m 6
Added Pri 1 - Cli 6 #7 nr_m 7
Added Pri 0 - Cli 1 #1 nr_m 8

-=-=-=-=-=-=-=-=-=-= Reader sleeping, press enter


-=-=-=-=-=-=-=-=-=-= Reader start popping messages out from buffer
Popped
Pri 0 - Cli 1 #1 KcEtPFRYUlVOFtmDyUuXJYzSgkTEGYCQdGlsMC
Added Pri 0 - Cli 1 #2 nr_m 8
Popped
Pri 0 - Cli 1 #2
IPOWxklAiFWFquGJCqEiWKcEtPFRYUlVOFtmDyUuXJYzSgkTEGYCQdGlsMCR
IPOWxklAiFWFquGJCqEiwFloirzCfDTnuKKruurUJSkTG
Added Pri 0 - Cli 2 #1 nr_m 8
Popped
Pri 0 - Cli 2 #1 KcEtPFRYUlVOFtmDyuXYzgEYCQdGlsMCRIPOWx
Added Pri 0 - Cli 3 #1 nr_m 8
Popped
.
.
.


Pri 0 - Cli 4 #28 MyMTaSEviiYpjYzecT
Popped
Pri 1 - Cli 6 #1 KcEtPFRYUlVOFtmDyUuXJYzSgkTEGYCQdGlsMC
Added Pri 0 - Cli 4 #2 nr_m 7
Popped
Pri 0 - Cli 4 #29
LFaapxzdGNQsBqFUQzYMhiMWuMxAQtqcysenpFQwSgovyVPPVnDexPdrD
Popped
Pri 1 - Cli 6 #2
IPOWxklAiFWFquGJCqEiwFloirzCfDTnuKKruurczqhqkPbPFHxdMitWCuzhxUxTe
HLYBecdUlvhAywhFVmugfsizRryOoSsXdtBjxeEj
Popped
Pri 1 - Cli 6 #3
NLzJtgGfamLTUMMoLaEdvciQdRNjxYkKkLvdrCkuoXplkdzvgdADhKTmDIVBhhLR
U
Added Pri 0 - Cli 4 #3 nr_m 5
Popped
Pri 0 - Cli 4 #30
sUvKYtCClsHYQBgEwHaLWXazBzQeTThnPExnybRivYjLBrszyuMUROVuOlzJhi
wwmVMMwdvUeGffxzhwttsmIOJyzKHIsfFGARvwWqQAYYIVzptVKmjScssDCzM
```

```
Wet
Popped
Pri 1 - Cli 6 #4
WoliizGzmqcLNkRNnyzGlCPifYQrRMCPAnxioDJbtOOGyfvmeWUrzLzEjRXAezre
oppFsagOoWwnCsBIqVzPhAU
Popped
Pri 1 - Cli 6 #5
STVYVMeJBTOWWWkKtIzXcaHTYGihiezAaWAvjfEMavKwRWHmfIJHLQCjylQhr
RJRoKozptnPQaohwvwDfhnqypzYCRhuLTnbdcauxplNPzWowSrcAeSa
Popped
Pri 1 - Cli 6 #6
UayIlUwDIXgMzCJrnyiNUWlPqNrwltscTrNDnmGXlOjnQUegSMVPLGGdUxzEqSg
llVRYHzwVqHIGcmmWajNNqTqmQQqiKWwXrpvboTweDgNH
Popped
Pri 1 - Cli 6 #7
bdULriBKYnDQdlabJxuyVVnQttTBiavKgQxzayJaNoTSCttLTPKQlzgGucIdeDPmv
MlxmXAzLTtpoOBjdnzqoIxKMhnqMeFkqqhcpJDdcwStKTcoiceWkDhwkwOzbT
Buffer is empty
```

- **Buffer have 7 priority messages and the next messages are low priority:** In this case there will be tested the opposite, the buffer contains 7 high priority messages and all the next messages will be low-low priority, it is expected to read first all the 7 priority messages in order.

```
cristiam@cristiam-ubuntu:~/Desktop/cdmj$ ./only0last1
Added Pri 0 - Cli 6 #1 nr_m 1
Added Pri 0 - Cli 6 #2 nr_m 2
Added Pri 0 - Cli 6 #3 nr_m 3
Added Pri 0 - Cli 6 #4 nr_m 4
Added Pri 0 - Cli 6 #5 nr_m 5
Added Pri 0 - Cli 6 #6 nr_m 6
Added Pri 0 - Cli 6 #7 nr_m 7
Added Pri 1 - Cli 2 #1 nr_m 8

-=-=-=-=-=-=-=-=-= Reader sleeping, press enter


-=-=-=-=-=-=-=-=-= Reader start popping messages out from buffer
Popped
Pri 0 - Cli 6 #1
vOrbrFsRxsHbNEHDCtwtJBxtXUSvtxaQLRrezMvYEeAtijzMEVGpzfjyzdtVavNNpFrQROo
xsQTCbsOgpwVocgPDJIALgOYVvPlMdBlVSEXtwOBmmzC
Added Pri 1 - Cli 1 #1 nr_m 8
Popped
Pri 0 - Cli 6 #2
HRuRCuEiIdfDsQryRFvLJvhiJKUvJymRsgKuCOELTLqobKmUPJgYEpGpzCNJdzCVHorJ
eXwainOlxcFMlNMqCvHCxUnCTRXChqMnNkNvzb
Added Pri 1 - Cli 2 #2 nr_m 8
Popped
```

```
Pri 0 - Cli 6 #3
wDOkRDzjiUqMtLbwGsTJBKxQzJeXJhfhNVuGzTpHnHTgSucBOYmrKLHJUNIdvonkJhq
KAHtnOovhKAKYYYrKjbvDoEJLUwvFdNQF
Added Pri 1 - Cli 3 #1 nr_m 8
Popped
Pri 0 - Cli 6 #4 lUjaPskPCJPCczIDVQsBBfVxbaCQSHIEDweToOKSzCUc
Added Pri 1 - Cli 5 #1 nr_m 8
Popped
Pri 0 - Cli 6 #5
hHYxzzzfuYiVDaPmLTqHxJXMvpnxlrBSzzQyzrFVPPSuqihdBXNzIknDCCdnueGtDyTeRB
cgQuDickMGjzhrlwWnybbsFJNljhrAiTIaOLISvxYeyFxjBwXAXaUdjiq
Added Pri 1 - Cli 4 #1 nr_m 8
Popped
Pri 0 - Cli 6 #6
PhuzdCbrOMJIlKsjPpvtNutIUpqfXgzOpTPSxrlLfXzRhRcYiXRwtmhNczsbftqwMhrKAcxgz
WzIPBhyBaWuNfkpeEslyihmsY
Added Pri 1 - Cli 1 #2 nr_m 8
Popped
Pri 0 - Cli 6 #7 sdWYEUxNMbWKcwgylLiCRnvC
Added Pri 1 - Cli 2 #3 nr_m 8
Popped
Pri 1 - Cli 2 #1
vOrbrFsRxsHbNEHDCtwtJBxtXUSvtxaQLRrezMvYEeAtijzMEVGpzfjyzdtVavNNpFrQROo
xsQTCbsOgpwVocgPDJICtBXSvxaQReMEtjMVpfzaprRo
Added Pri 1 - Cli 3 #2 nr_m 8
Popped
Pri 1 - Cli 1 #1
vOrbrFsRxsHbNEHDtwJxtUtLrzvYeAizEGzjydtVvNNFQOxsQTCbsOgpwVocgPDJLgVIBI
wCHRRuiIQrRFhKJRguOETLqbUJEGpJHJXaFNvCxUnCTRX
Added Pri 1 - Cli 5 #2 nr_m 8
.
.
.
```

## RESULTS

The monitor is an effective mechanism to control critical zones, it was shown the proposed program in different scenarios, aiming to test what happen if the buffer is empty, full, and how will behave with high and low priority messages in different cases.

## REFERENCES

- Modern Operating Systems (4th Edition) by Andrew S. Tanenbaum (Author), Herbert Bos (Author)
- Data Structures Using C++ (2nd Edition) by D.S. Malik