Student: Cristiam Martin Jackson
[SOI] - Systemy Operacyjne

REPORT
LABORATORY N. 3 - SYNCHRONIZATION OF PROCESSES USING SEMAPHORES

PURPOSE

Implement a chat service in order to communicate between two group of users (with and without privileges), the chat room shall be implemented in the same machine, using an automatic message generation system and with the use of semaphores.

SOLUTION

For the solution, C language will be used under the Linux environment. For the semaphores and the critical zone, the libraries given by Linux will be used.
A queue will be implemented in order to keep track of the received messages, the queue will work in cyclic way, in that way allocate messages with higher priority will be easier, as well as pop them out, associated to the queue, there should be three basic operations: push, push front and pop.
Three executable files will be created (producer a, producer b, chat management).
Producer a: This executable will simulate the user with higher priorities, it means, its messages are going to be located at the head of the queue.
Producer b: It will simulate regular priority messages, its messages are going to be located at the end of the queue.
Chat management: It will pop the messages to the "chat room" in order, it means, popping from the head of the queue, so that messages added by producer a will be shown first.
The code implemented is shown in the following tables, it will be seen the next files:

BUFFER.h BUFFER.c	Contain the main logic of the semaphores and the buffer, it was used as additional parameter when compiling	
bufchat2.c	Contain the main test in order to check the semaphores operations, adition tests are added in the 'testing' section. Input parameters are memory key and buffer size.	Cc -o b2chat bufchat2.c BUFFER.c
Bufproa.c and bufprob.c	(Producer tasks), 'a' produces the high priority messages while 'b' generate the regular messages. Input parameters are memory ID and buffer size.	Cc -o bproa bufproa.c BUFFER.c Cc -o bprob bufprob.c BUFFER.c
bufrea.c	In charge of pop out	Cc -o brea bufrea.c

	messages from the queue (consument task). Input parameters are memory ID and buffer size.	BUFFER.c

BUFFER.h

```
#ifndef shared_QUEUE
#define shared_QUEUE

/*Memory Segment*/
int memid;
int memsize;

/* functions */
int* queue;
int* init_queue(int,int);
int* attach_queue(int);

/*MANAGE QUEUE*/
void q_clear(void);
void push_rear(int);
void push_front(int);
void pop(void);
void q_detache(void);
void show_queue(void);

/*MANAGE BUFFER*/
void buf_push_rear(int);
void buf_push_front(int);
void buf_pop(void);

/*SEMAPHORES*/
int create_semaphores(int);
void del_semaphores(void);
#endif
```

BUFFER.c

```
#include "BUFFER.h" /* data structure for the messages in chat room */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ipc.h>
```

```
#include <sys/sem.h>
#include <sys/types.h>
#include <sys/shm.h>

/* Semaphores' identificators */
#define S_FULL 100
#define S_EMPTY 110
#define S_MUTEX 120

/* Semaphores' variables */
int s_full;
int s_empty;
int s_mutex;
int onoff;

/* Memory segment

int memid;
int memsize;*/


int s_alloc(key_t,int);
int s_dealloc(int);
int s_ini(int,int);


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

/************************************ Begin data structure */

int* init_queue(int shmid, int q_size){
 memsize=q_size;
 if((memid=shmget(shmid,(q_size+3)*sizeof(int),0666|IPC_CREAT))<=0){
  printf("Error creating memory segment with id: %d\n",memid);
  return queue;
 }
 printf("[BUFFER.c] memid: %d\n",memid);
 if((queue=shmat(memid,0,0))<=0)
  printf("Error attaching memory");

 queue[q_size]=0;/*front of queue*/
 queue[q_size+1]=q_size-1;/*rear of queue*/
 queue[q_size+2]=0;/*counter of elements (last position)*/

 return queue;
}

int* attach_queue(int keysh){
 printf("[BUFFER.c] memid: %d\n",keysh);
 if((queue=shmat(keysh,0,0))<=0)
  printf("Error attaching memory");
 return queue;
}


/*********************************** end structure for queue */

/* -=-=-=-=-=-=-=-=-=-=-=-=-=-= SEMAPHORES -=-=-=-=-=-=-=-=-=-=-=-=-=-= */
int create_semaphores(int M){

 s_full = s_alloc(S_FULL, 0666 | IPC_CREAT);
 s_empty = s_alloc(S_EMPTY, 0666 | IPC_CREAT);
 s_mutex = s_alloc(S_MUTEX, 0666 | IPC_CREAT);

 printf("Semaphores provided\ns_full: %d\ns_empty: %d\ns_mutex: %d\nM:
%d\n",s_full,s_empty,s_mutex,M);
 if( s_full<0 || s_empty<0 || s_mutex<0) return -1;

 /*printf("s_ini s_full: %d\n",s_ini(s_full, M-1));
 printf("s_ini s_empty: %d\n",s_ini(s_empty,0));
 printf("s_ini s_mutex: %d\n",s_ini(s_mutex,1));*/
 s_ini(s_full,0);
 s_ini(s_empty,M);
 s_ini(s_mutex,1);
 onoff=1;
 return 1;
}

/*-=-=-=-=-=-=-=-=-= BUFFER_DE ALLOCATION -=-=-=-=-=-=-=-=-=*/
void del_semaphores(){
 s_dealloc(s_empty);
 s_dealloc(s_mutex);
 s_dealloc(s_full);
}


/* Allocate semaphore */
int s_alloc(key_t _key_t,int s_flags){
 return semget(_key_t,1,s_flags);
}
/* Deallocate semaphore */
int s_dealloc(int s_id){
 union semun noarg;
 return semctl(s_id,1,IPC_RMID, noarg);
}
```

```
/* Initizalization */

int s_ini(int s_id, int val){
 /*printf("initializing semaphores / s_id: %d, val: %d\n",s_id,val);*/
 union semun arg;
 unsigned short values[1];
 values[0]=val;
 arg.array = values;
 return semctl(s_id,0,SETALL,arg);
}

/*==========================DOWN sem_wait -=-=-= block -=-=-= proberen*/
int sem_wait(int s_id){
 int a;
 struct sembuf sb[1];
 sb[0].sem_num=0;
 sb[0].sem_op = -1; /*allocate resources*/
 sb[0].sem_flg = SEM_UNDO; /* Automatically undone when process terminates*/
 /*printf("before semop block\n");*/
 a= semop(s_id,sb,1);
 /*printf("-=-=-= block\ns_id: %d, semop: %d\n",s_id,a);*/
 /*printf("down ");*/
 return a;
}

/*===================UP sem_post -=-=-= unblock -=-=-= verhogen*/
int sem_post(int s_id){
 int b;
 struct sembuf sb[1];
 sb[0].sem_num = 0;
 sb[0].sem_op = 1;
 sb[0].sem_flg = SEM_UNDO;
 /*printf("before semop unblock\n");*/
 b=semop(s_id,sb,1);
 /*printf("-=-=-= unblock\ns_id: %d, semop: %d\n",s_id,b);*/
/*printf("up ");*/
 return b;
}


/* ************************************************************
   ************************************************************
   ******************* MANAGE THE BUFFER ***********************
   ************************************************************
   ************************************************************ */

/*-=-=-=-=-=-=-= BUFFER_PUSH_REAR -=-=-=-=-=-=-=*/
void buf_push_rear(int data){
 if (onoff == 0){
  printf("Semaphores not initialized");
 }
 sem_wait(s_empty);
 sem_wait(s_mutex);
 printf("\nPushing %d\n",data);
 push_rear(data);
 sem_post(s_mutex);
 sem_post(s_full);
}

/*-=-=-=-=-=-=-= BUFFER_PUSH_FRONT -=-=-=-=-=-=-=*/
void buf_push_front(int data){
 if (onoff == 0){
  printf("Semaphores not initialized");
 }
 sem_wait(s_empty);
 sem_wait(s_mutex);
 printf("\nFront pushing %d\n",data);
 push_front(data);
 sem_post(s_mutex);
 sem_post(s_full);
}

/*-=-=-=-=-=-=-= BUFFER_POP -=-=-=-=-=-=-=*/
/*Returns the first element of the queue*/

void buf_pop(){
 if (onoff == 0){
  printf("Semaphores not initialized");
 }
 sem_wait(s_full);
 sem_wait(s_mutex);
 printf("\nPopping %d\n",queue[queue[memsize]]);
 pop();
 sem_post(s_mutex);
 sem_post(s_empty);
}






/* ************************************************************
   ************************************************************
   ******************* MANAGE THE QUEUE ***********************
   ************************************************************
   ************************************************************ */


/*=-=-=-=-=-=-= CLEAR THE QUEUE -=-=-=-=-=-=-=*/
```

```
void q_clear(){

 int i;
 queue[memsize]=0;/*front of queue*/
 queue[memsize+1]=memsize-1;/*rear of queue*/
 queue[memsize+2]=0;/*counter of elements (last position)*/
 for(i = 0; i<=memsize-1; i++){
  queue[i] = 0;
 }
}



/*-=-=-=-=-=-=-= PUSH_REAR -=-=-=-=-=-=-=*/
void push_rear(int data){
 queue[memsize+1]=(queue[memsize+1]+1)%memsize;/*It makes the cyclic queue*/
 queue[memsize+2]+=1;/*counter*/
 queue[queue[memsize+1]]=data;
}

/*-=-=-=-=-=-=-= PUSH_FRONT -=-=-=-=-=-=-=*/
void push_front(int data){

 if(queue[memsize+2]+1<=memsize){/*if there are free slots*/
  queue[memsize]=(queue[memsize]-1)<0?memsize-1:queue[memsize]-1;/*if below zero,
point to the end (cyclic queue)*/
  queue[memsize+2]+=1;
  queue[queue[memsize]]=data;
 }
}

/*-=-=-=-=-=-=-= POP -=-=-=-=-=-=-=*/
/*Returns the first element of the queue*/

void pop(){
 if (queue[memsize+2]>0){/*If there are still elements*/
 queue[memsize+2]-=1;/*decrease counter*/
 queue[queue[memsize]]=0;/*make zero deleted element*/
 queue[memsize]=(queue[memsize]+1)%(memsize);/*Moves the pointer in cyclic way*/
 }
}

/*-=-=-=-=-=-=-= DETACHE MEMORY -=-=-=-=-=-=-=*/
void q_detache(){
shmdt(queue);
}

/*-=-=-=-=-=-=-= SHOW_QUEUE -=-=-=-=-=-=-=*/
void show_queue(){
 /*system("clear");*/
 int i;
 printf("QueueFront: %d\nQueueRear: %d\nCounter: %d\n",
queue[memsize]+1,queue[memsize+1]+1,queue[memsize+2]);/*only for visualisation
purposes*/
 for(i = 0; i<=memsize-1; i++){
 printf("%d- ", queue[i]);
          }
printf("\n\n");
}
```

- It contains the main test, the program executes the compiled files as appreciated, different tests are shown at the end of this report, showing the modification of the next code.

bufchat2.c (compiled as: cc -o b2chat bufchat2.c BUFFER.c)

```
#include "BUFFER.h"
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
#include <sys/shm.h>
#include <sys/types.h>
/* This programm only initializes memory in order to execute by myself the proca, procb
and rea scripts */
int ret;
int main(int argc, char* argv[]){
char command[100];
int memkey;
memkey=atoi(argv[1]);
memsize=atoi(argv[2]);

init_queue(memkey,memsize);
q_clear();
show_queue();
ret=create_semaphores(memsize);
printf("Create semaphores: %d\n",ret);

/*-=-=-= PROA & PROB & REA -=-=-=*/

printf("\t-=-=-=-=-=bproa & bprob & brea\n");
sprintf(command,"./bproa %d %d & ./bprob %d %d & ./brea %d
%d",memid,memsize,memid,memsize,memid,memsize);
/*sprintf(command,"./bproa %d %d & ./bprob %d %d",memid,memsize,memid,memsize);*/
system(command);

printf("Finishing, detaching buffer and semaphores\n");
show_queue();
/*q_clear();
show_queue();*/

del_semaphores();
q_detache();

return 0;
}
```

- Writer with higher priority

bufproa.c (compiled as: cc -o bproa bufproa.c BUFFER.c)

```
#include "BUFFER.h"
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
#include <sys/shm.h>
#include <sys/types.h>
/* Producer with higher priority */


int main(int argc, char* argv[]){
int memkey,ret;
memkey=atoi(argv[1]);
memsize=atoi(argv[2]);
int i;
printf("Printing since producera\nmemid: %d\nmemsize: %d\n",memkey,memsize);
/*init_queue(memkey,memsize);*/

ret=create_semaphores(memsize);
printf("Create semaphores bufproa: %d\n",ret);
attach_queue(memkey);
printf("attached_queue bufproa\n");

srand(time(NULL));
for(i=0;i<10;i++){
 usleep(5000);
 printf("\t\t\tWriter VIP: %d\n",i);
 buf_push_front((i+1)*10+rand()%10);
}




/*show_queue();
q_clear();
show_queue();
push_rear(4);
show_queue();
push_rear(8);
show_queue();
push_front(29);
show_queue();
push_front(30);
show_queue();
*/
return 0;
}
```

- Writer but regular priority

bufprob.c (compiled as: cc -o bprob bufprob.c BUFFER.c)

```
#include "BUFFER.h"
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
#include <sys/shm.h>
#include <sys/types.h>
/* Producer with higher priority */


int main(int argc, char* argv[]){
int memkey,ret;
memkey=atoi(argv[1]);
memsize=atoi(argv[2]);
int i;
printf("Printing since producerb\nmemid: %d\nmemsize: %d\n",memkey,memsize);




/*init_queue(memkey,memsize);*/

ret=create_semaphores(memsize);
printf("Create semaphores bufprob: %d\n",ret);
attach_queue(memkey);
printf("attached_queue bufprob\n");


srand(time(NULL));
for(i=0;i<10;i++){
 usleep(2000);
printf("\t\t\tWriter b: %d\n",i);
 buf_push_rear((i+1)*1000+rand()%10);
}
/*show_queue();
q_clear();
show_queue();
push_rear(4);
show_queue();
push_rear(8);
show_queue();
push_front(29);
show_queue();
push_front(30);
show_queue();
*/
return 0;
}
```

- Chat management, reads the messages in order of priority

bufrea.c (compiled as: cc -o brea bufea.c BUFFER.c)

```
#include "BUFFER.h"
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
#include <sys/shm.h>
#include <sys/types.h>
/* Producer with higher priority */


int main(int argc, char* argv[]){
int memkey;
memkey=atoi(argv[1]);
memsize=atoi(argv[2]);
int i,ret;
printf("Printing since readera\nmemid: %d\nmemsize: %d\n",memkey,memsize);
/*init_queue(memkey,memsize);*/

ret=create_semaphores(memsize);
printf("Create semaphores bufrea: %d\n",ret);
attach_queue(memkey);


printf("\t\t\t-=-=-= Buffer will sleep for 5 seconds -=-=-=\n");
/*srand(time(NULL));*/
sleep(5);
printf("\t\t\t-=-=-= Buffer reader entering to endless loop -=-=-=\n");
show_queue();
/*for(i=0;i<30;i++)*/
for(;;){
printf("\n\t\t\tManagement of chat space\n");
buf_pop();
show_queue();
}
/*printf("rea ");
show_queue();*/



/*show_queue();
q_clear();
show_queue();
push_rear(4);
show_queue();
push_rear(8);
show_queue();
push_front(29);
show_queue();
push_front(30);
show_queue();
*/
return 0;
}
```

TESTING

The next test was executed for the previously detailed code, in which every writer will produce 10 messages each, writer whose messages have the highest priority (A) will produce numbers less than 1000 and writer whose messages have regular priority (B) generate numbers greater greater than 1000, every number goes in a consecutive sequence so that is easier to evidence the working of the semaphores when it goes to block and unblock processes.

The size of the buffer will be 10, so that the semaphore will manage the messages in order to show them all.

In order to test the case when the buffer is full, the chat management file will start working 5 seconds after the writers start writing messages, it will assure that the buffer will be full by the time the chat management file pop them out. On the other hand, at the end of the test will be checked the case when the chat management file tries to pop messages from an empty buffer, it is done by stopping the writers and let the chat manager keep working.

As input arguments, the file requires a random key for the memory and a number of spaces for the buffer, in the next test it is seen the key 126 and a buffer with 10 positions.

```
cristiam@cristiam-ubuntu:~/Desktop/cdmj$ ./b2chat 126 10
[BUFFER.c] memid: 6586381
QueueFront: 1
QueueRear: 10
Counter: 0
0- 0- 0- 0- 0- 0- 0- 0- 0- 0-

Semaphores provided
s_full: 1081344
s_empty: 1114113
s_mutex: 1146882
M: 10
Create semaphores: 1
        -=-=-=-=-=bproa & bprob & brea
Printing since producera
memid: 6586381
memsize: 10
Semaphores provided
s_full: 1081344
s_empty: 1114113
Printing since producerb
s_mutex: 1146882
Printing since readera
memid: 6586381
memid: 6586381
memsize: 10
memsize: 10
M: 10
Semaphores provided
Semaphores provided
Create semaphores bufproa: 1
s_full: 1081344
s_full: 1081344
[BUFFER.c] memid: 6586381
s_empty: 1114113
s_mutex: 1146882
attached_queue bufproa
M: 10
Create semaphores bufprob: 1
[BUFFER.c] memid: 6586381
s_empty: 1114113
attached_queue bufprob
s_mutex: 1146882
M: 10
Create semaphores bufrea: 1
[BUFFER.c] memid: 6586381
                        -=-=-= Buffer will sleep for 5 seconds -=-=-=
                        Writer b: 0

Pushing 1008
```

```
                              Writer b: 1

Pushing 2008
                              Writer VIP: 0

Front pushing 18
                              Writer b: 2

Pushing 3009
                              Writer b: 3

Pushing 4009
                              Writer VIP: 1

Front pushing 28
                              Writer b: 4

Pushing 5005
                              Writer b: 5

Pushing 6009
                              Writer b: 6

Pushing 7000
                              Writer VIP: 2

Front pushing 39
                              Writer b: 7
                              Writer VIP: 3
                              -=-=-= Buffer reader entering to endless loop -=-=-=
QueueFront: 8
QueueRear: 7
Counter: 10
1008- 2008- 3009- 4009- 5005- 6009- 7000- 39- 28- 18-


                              Management of chat space

Popping 39
QueueFront: 9
QueueRear: 7
Counter: 9
1008- 2008- 3009- 4009- 5005- 6009- 7000- 0- 28- 18-



Pushing 8002
                              Management of chat space

Popping 28
```

```
QueueFront: 10
QueueRear: 8
Counter: 9
1008- 2008- 3009- 4009- 5005- 6009- 7000- 8002- 0- 18-



Front pushing 49
                              Management of chat space

Popping 49
QueueFront: 10
QueueRear: 8
Counter: 9
1008- 2008- 3009- 4009- 5005- 6009- 7000- 8002- 0- 18-


                              Management of chat space

Popping 18
QueueFront: 1
QueueRear: 8
Counter: 8
1008- 2008- 3009- 4009- 5005- 6009- 7000- 8002- 0- 0-


                              Management of chat space

Popping 1008
QueueFront: 2
QueueRear: 8
Counter: 7
0- 2008- 3009- 4009- 5005- 6009- 7000- 8002- 0- 0-


                              Management of chat space

Popping 2008
QueueFront: 3
QueueRear: 8
Counter: 6
0- 0- 3009- 4009- 5005- 6009- 7000- 8002- 0- 0-


                              Management of chat space

Popping 3009
QueueFront: 4
QueueRear: 8
Counter: 5
```

```
0- 0- 0- 4009- 5005- 6009- 7000- 8002- 0- 0-


                    Management of chat space

Popping 4009
QueueFront: 5
QueueRear: 8
Counter: 4
0- 0- 0- 0- 5005- 6009- 7000- 8002- 0- 0-


                    Management of chat space

Popping 5005
QueueFront: 6
QueueRear: 8
Counter: 3
0- 0- 0- 0- 0- 6009- 7000- 8002- 0- 0-


                    Management of chat space

Popping 6009
QueueFront: 7
QueueRear: 8
Counter: 2
0- 0- 0- 0- 0- 0- 7000- 8002- 0- 0-


                    Management of chat space

Popping 7000
QueueFront: 8
QueueRear: 8
Counter: 1
0- 0- 0- 0- 0- 0- 0- 8002- 0- 0-


                    Management of chat space

Popping 8002
QueueFront: 9
QueueRear: 8
Counter: 0
0- 0- 0- 0- 0- 0- 0- 0- 0- 0-


                    Management of chat space
                    Writer b: 8
```

```

Pushing 9006

Popping 9006
QueueFront: 10
QueueRear: 9
Counter: 0
0- 0- 0- 0- 0- 0- 0- 0- 0- 0-


                          Management of chat space
                          Writer b: 9

Pushing 10006

Popping 10006
QueueFront: 1
QueueRear: 10
Counter: 0
0- 0- 0- 0- 0- 0- 0- 0- 0- 0-


                          Management of chat space
                          Writer VIP: 4

Front pushing 55

Popping 55
QueueFront: 1
QueueRear: 10
Counter: 0
0- 0- 0- 0- 0- 0- 0- 0- 0- 0-


                          Management of chat space
                          Writer VIP: 5

Front pushing 69

Popping 69
QueueFront: 1
QueueRear: 10
Counter: 0
0- 0- 0- 0- 0- 0- 0- 0- 0- 0-


                          Management of chat space
                          Writer VIP: 6

Front pushing 70
```

```
Popping 70
QueueFront: 1
QueueRear: 10
Counter: 0
0- 0- 0- 0- 0- 0- 0- 0- 0- 0-


                    Management of chat space
                    Writer VIP: 7

Front pushing 82

Popping 82
QueueFront: 1
QueueRear: 10
Counter: 0
0- 0- 0- 0- 0- 0- 0- 0- 0- 0-


                    Management of chat space
                    Writer VIP: 8

Front pushing 96

Popping 96
QueueFront: 1
QueueRear: 10
Counter: 0
0- 0- 0- 0- 0- 0- 0- 0- 0- 0-


                    Management of chat space
                    Writer VIP: 9

Front pushing 106

Popping 106
QueueFront: 1
QueueRear: 10
Counter: 0
0- 0- 0- 0- 0- 0- 0- 0- 0- 0-


                    Management of chat space
^CFinishing, detaching buffer and semaphores
QueueFront: 1
QueueRear: 10
Counter: 0
0- 0- 0- 0- 0- 0- 0- 0- 0- 0-
```

It can be seen in the test, that once the buffer is full, the process stops adding items into the queue, due to the operation of the semaphores, once the reader starts working and popping elements from the queue, a down operation is done and a space is released, allowing the next process to be executed and hence, store the next message in the buffer.
It can be seen also, that the order in which the messages where propped out of the buffer, was taking into account the highest priority, in that way, the numbers below 1000 are first shown on the chatroom than the messages with lowest priority. Finally when all elements are popped out from the buffer, and both message-producers run out of messages (10 each), the reader keeps waiting for new messages to be popped out, or new messages to be handled into the buffer.

- Now, executing only priority messages in a Buffer with size 8, after initializated the semaphores and the shared memory space, the next segment of code is executed

```
q_clear();
show_queue();
printf("\t-=-=-=-=-=proa & rea\n");
sprintf(command,"./bproa %d %d & ./brea %d %d",memid,memsize,memid,memsize);
system(command);
show_queue();
```

It executes the file associated to the producer of high priority messages and the element that pop the messages from the queue in order to show them in the screen. All the messages are pushed in until the Buffer is full and the semaphore indicates that cannot continue adding items, the process stops until “rea” appears and start popping messages, once one space is released, the producer receives the signal in order to complete his task and write the left messages, since the reader file actuates as soon as the message is popped in, the only action left is to pop it out the buffer as soon as the message appears. The result can be appreciate up next.

```
        -=-=-=-=-=proa & rea
Printing since producera
memid: 2588680
memsize: 8
Semaphores provided
s_full: 0
s_empty: 32769
Printing since readera
s_mutex: 65538
memid: 2588680
M: 8
memsize: 8
Semaphores provided
```

```
Create semaphores bufproa: 1
s_full: 0
[BUFFER.c] memid: 2588680
s_empty: 32769
s_mutex: 65538
M: 8
attached_queue bufproa
Create semaphores bufrea: 1
[BUFFER.c] memid: 2588680
                        -=-=-= Buffer will sleep for 5 seconds -=-=-=
                        Writer VIP: 0

Front pushing 17
                        Writer VIP: 1

Front pushing 23
                        Writer VIP: 2

Front pushing 38
                        Writer VIP: 3

Front pushing 45
                        Writer VIP: 4

Front pushing 54
                        Writer VIP: 5

Front pushing 62
                        Writer VIP: 6

Front pushing 73
                        Writer VIP: 7

Front pushing 80
                        Writer VIP: 8
                        -=-=-= Buffer reader entering to endless loop -=-=-=
QueueFront: 1
QueueRear: 8
Counter: 8
80- 73- 62- 54- 45- 38- 23- 17-
                        Management of chat space

Popping 80
QueueFront: 2
QueueRear: 8

Counter: 7
Front pushing 97
0- 73- 62- 54- 45- 38- 23- 17-
                        Management of chat space
```

```
Popping 97
QueueFront: 2
QueueRear: 8
Counter: 7
0- 73- 62- 54- 45- 38- 23- 17-


                              Management of chat space

Popping 73
QueueFront: 3
QueueRear: 8
Counter: 6
0- 0- 62- 54- 45- 38- 23- 17-


                              Management of chat space

Popping 62
QueueFront: 4
QueueRear: 8
Counter: 5
0- 0- 0- 54- 45- 38- 23- 17-
                              Management of chat space

Popping 54
QueueFront: 5
QueueRear: 8
Counter: 4
0- 0- 0- 0- 45- 38- 23- 17-
                              Management of chat space

Popping 45
QueueFront: 6
QueueRear: 8
Counter: 3
0- 0- 0- 0- 0- 38- 23- 17-


                              Management of chat space

Popping 38
QueueFront: 7
QueueRear: 8
Counter: 2
0- 0- 0- 0- 0- 0- 23- 17-
                              Management of chat space

Popping 23
```

```
QueueFront: 8
QueueRear: 8
Counter: 1
0- 0- 0- 0- 0- 0- 0- 17-
                         Management of chat space

Popping 17
QueueFront: 1
QueueRear: 8
Counter: 0
0- 0- 0- 0- 0- 0- 0- 0-
                         Management of chat space
                         Writer VIP: 9

Front pushing 101

Popping 101
QueueFront: 1
QueueRear: 8
Counter: 0
0- 0- 0- 0- 0- 0- 0- 0-
```

- Now testing reading the Buffer when empty, and after some time (6 seconds) starting the writer of messages:

```
q_clear();
show_queue();

sprintf(command,"./brea %d %d &",memid,memsize);/*execute in background*/
system(command);
sleep(6);
printf("\t-=-=-=-=-=proa & prob\n");
sprintf(command,"./bproa %d %d & ./bprob %d %d",memid,memsize,memid,memsize);
system(command);
show_queue();
```

```
Printing since readera
memid: 2588680
memsize: 8
Semaphores provided
s_full: 0
s_empty: 32769
s_mutex: 65538
M: 8
```

```
Create semaphores bufrea: 1
[BUFFER.c] memid: 2588680
                        -=-=-= Buffer will sleep for 5 seconds -=-=-=
                        -=-=-= Buffer reader entering to endless loop -=-=-=
QueueFront: 1
QueueRear: 8
Counter: 0
0- 0- 0- 0- 0- 0- 0- 0-


                        Management of chat space
        -=-=-=-=-=proa & prob
Printing since producera
Printing since producerb
memid: 2588680
memid: 2588680
memsize: 8
memsize: 8
Semaphores provided
Semaphores provided
s_full: 0
s_full: 0
s_empty: 32769
s_empty: 32769
s_mutex: 65538
s_mutex: 65538
M: 8
M: 8
Create semaphores bufprob: 1
Create semaphores bufproa: 1
[BUFFER.c] memid: 2588680
[BUFFER.c] memid: 2588680
attached_queue bufproa
attached_queue bufprob
                        Writer b: 0

Pushing 1009

Popping 1009
QueueFront: 2
QueueRear: 1
Counter: 0
0- 0- 0- 0- 0- 0- 0- 0-


                        Management of chat space
                        Writer b: 1

Pushing 2008
```

```
Popping 2008
QueueFront: 3
QueueRear: 2
Counter: 0
0- 0- 0- 0- 0- 0- 0- 0-


                    Management of chat space
                    Writer VIP: 0

Front pushing 19

Popping 19
QueueFront: 3
QueueRear: 2
Counter: 0
0- 0- 0- 0- 0- 0- 0- 0-


                    Management of chat space
                    Writer b: 2

Pushing 3001

Popping 3001
QueueFront: 4
QueueRear: 3
Counter: 0
0- 0- 0- 0- 0- 0- 0- 0-


                    Management of chat space
                    Writer b: 3

Pushing 4007

Popping 4007
QueueFront: 5
QueueRear: 4
Counter: 0
0- 0- 0- 0- 0- 0- 0- 0-


                    Management of chat space
                    Writer VIP: 1

Front pushing 28

Popping 28
QueueFront: 5
```

```
QueueRear: 4
Counter: 0
0- 0- 0- 0- 0- 0- 0- 0-


                    Management of chat space
                    Writer b: 4

Pushing 5006

Popping 5006
QueueFront: 6
QueueRear: 5
Counter: 0
0- 0- 0- 0- 0- 0- 0- 0-


                    Management of chat space
                    Writer b: 5

Pushing 6003

Popping 6003
QueueFront: 7
QueueRear: 6
Counter: 0
0- 0- 0- 0- 0- 0- 0- 0-


                    Management of chat space
                    Writer b: 6

Pushing 7008

Popping 7008
QueueFront: 8
QueueRear: 7
Counter: 0
0- 0- 0- 0- 0- 0- 0- 0-


                    Management of chat space
                    Writer VIP: 2

Front pushing 31

Popping 31
QueueFront: 8
QueueRear: 7
Counter: 0
```

```
0- 0- 0- 0- 0- 0- 0- 0-


                          Management of chat space
                          Writer b: 7

Pushing 8001

Popping 8001
QueueFront: 1
QueueRear: 8
Counter: 0
0- 0- 0- 0- 0- 0- 0- 0-


                          Management of chat space
                          Writer b: 8

Pushing 9004

Popping 9004
QueueFront: 2
QueueRear: 1
Counter: 0
0- 0- 0- 0- 0- 0- 0- 0-


                          Management of chat space
                          Writer VIP: 3

Front pushing 47

Popping 47
QueueFront: 2
QueueRear: 1
Counter: 0
0- 0- 0- 0- 0- 0- 0- 0-


                          Management of chat space
                          Writer b: 9

Pushing 10003

Popping 10003
QueueFront: 3
QueueRear: 2
Counter: 0
0- 0- 0- 0- 0- 0- 0- 0-
```

```
                         Management of chat space
QueueFront: 3
QueueRear: 2
Counter: 0
0- 0- 0- 0- 0- 0- 0- 0-

QueueFront: 1
QueueRear: 8
Counter: 0
0- 0- 0- 0- 0- 0- 0- 0-
```

It shows that the empty semaphore blocks the processes when the buffer is empty, in that case no process can continue unless it causes a change in the up semaphore, that is the case after 6 seconds, the empty semaphore is unblocked when the writer processes appears and start adding messages into the queue.

RESULTS

It was tested the cases when one writer adds messages until the buffer is full, then executing the reading function until the buffer is empty.

It was tested the trying to read an empty buffer as well as trying to add more messages into a full buffer, and finally it was tested all the processes working at once, in every case the conditions where met, showing that the operation of the semaphores worked as expected, blocking and allowing processes according to the case.

For every case and for visualization purposes, the test was executed for finite messages and showing for every case the resulting queue, indicating the head and the tail in every process, as well as the update of the queue every time its structure was updated (by pushing or popping elements).

It is important note that the memory is deallocated once the process is finished, otherwise, the showed messages can correspond to another execution different to the current one.

REFERENCES

- Modern Operating Systems (4th Edition) by Andrew S. Tanenbaum (Author), Herbert Bos (Author)
- Data Structures Using C++ (2nd Edition) by D.S. Malik
- http://pubs.opengroup.org/onlinepubs/7908799/xsh/syssem.h.html
- http://pubs.opengroup.org/onlinepubs/7908799/xsh/semget.html
- http://pubs.opengroup.org/onlinepubs/7908799/xsh/semctl.html
- http://pubs.opengroup.org/onlinepubs/7908799/xsh/semop.html