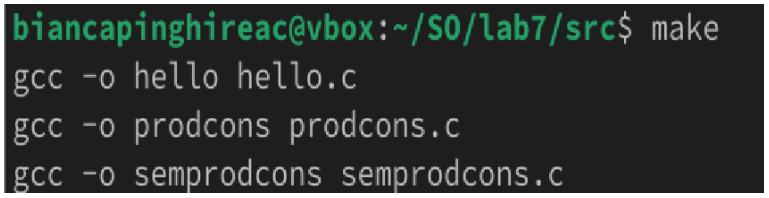
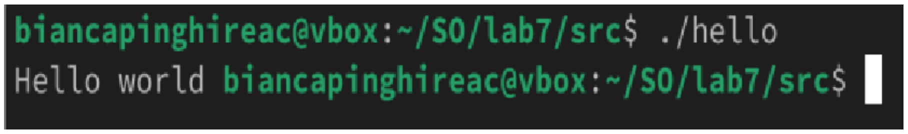
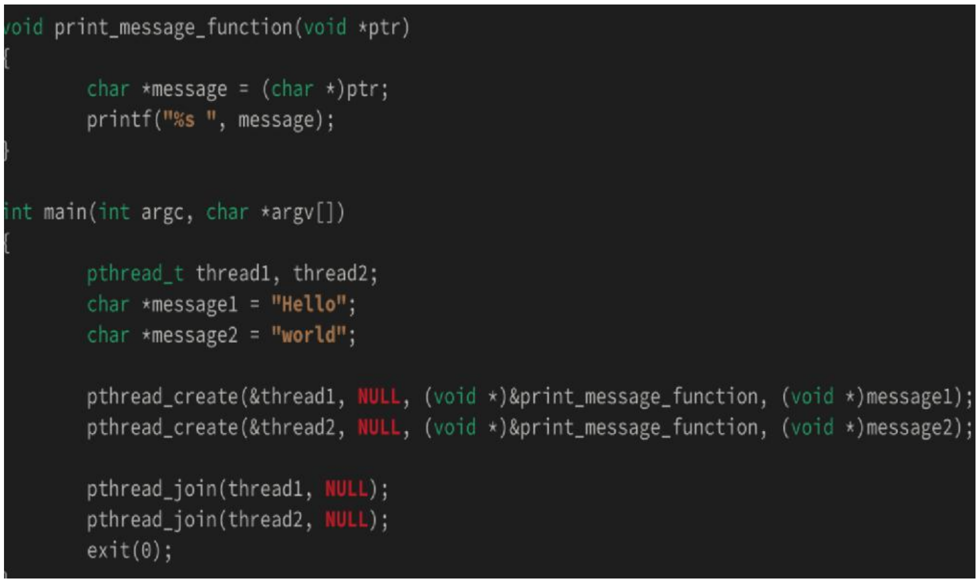
Laborator 7



```
biancapinghireac@vbox:~/SO/lab7/src$ make
gcc -o hello hello.c
gcc -o prodcons prodcons.c
gcc -o semprodcons semprodcons.c
```



```
biancapinghireac@vbox:~/SO/lab7/src$ ./hello
Hello world biancapinghireac@vbox:~/SO/lab7/src$
```

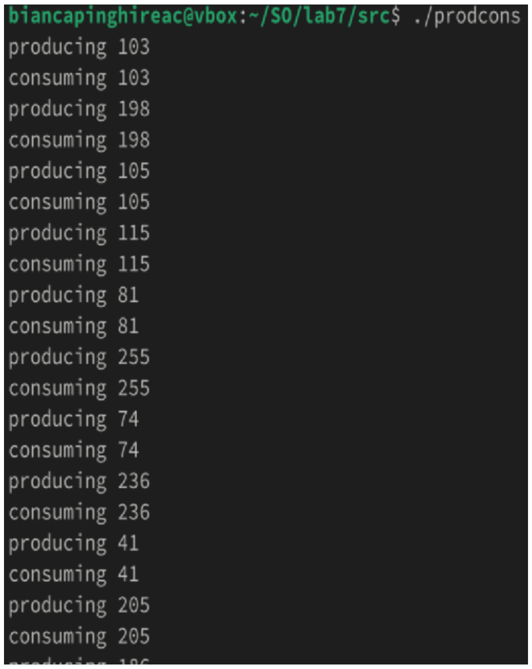


```
void print_message_function(void *ptr)
{
        char *message = (char *)ptr;
        printf("%s ", message);
}

int main(int argc, char *argv[])
{
        pthread_t thread1, thread2;
        char *message1 = "Hello";
        char *message2 = "world";

        pthread_create(&thread1, NULL, (void *)&print_message_function, (void *)message1);
        pthread_create(&thread2, NULL, (void *)&print_message_function, (void *)message2);

        pthread_join(thread1, NULL);
        pthread_join(thread2, NULL);
        exit(0);
```

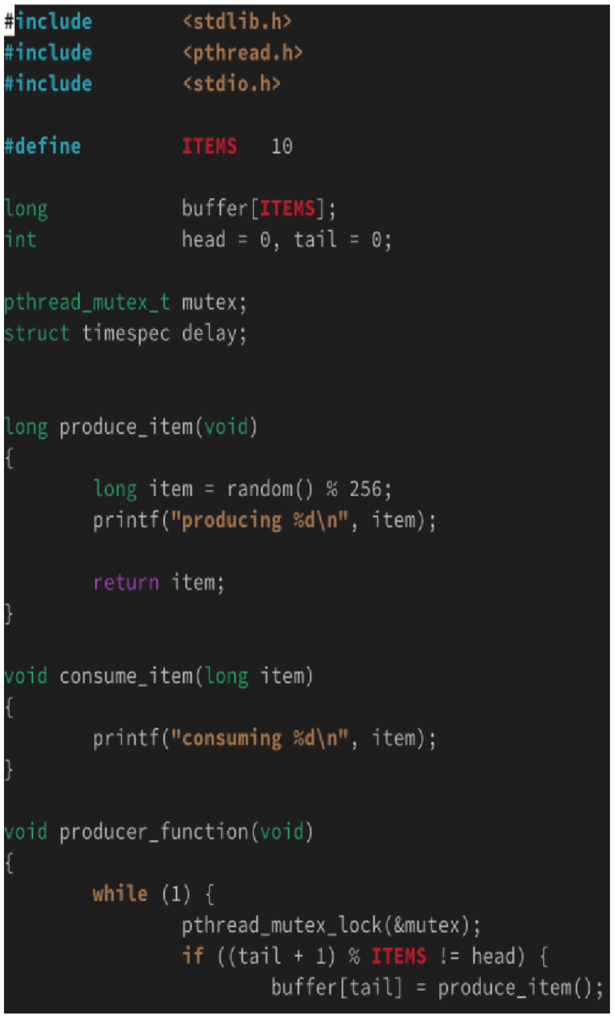
Hello.c

In codul de mai sus se initializeaza doua fire de executie(threads), fiecare are atribuit un mesaj(Hello/world), se asteapta terminarea lor apoi se incheie programul.

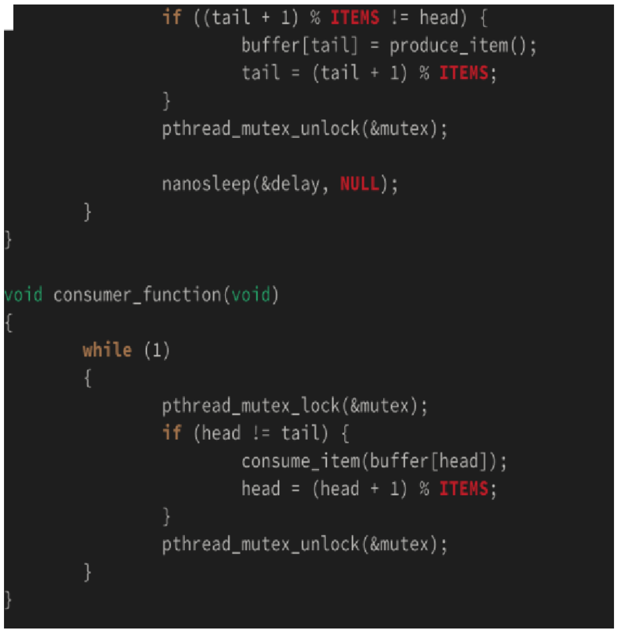
Rezultatul: printarea pe eran al celor 2 mesaje (ordinea poate fi gresita – ex word Hello, deoarece cele 2 fire nu sunt sincronizate)



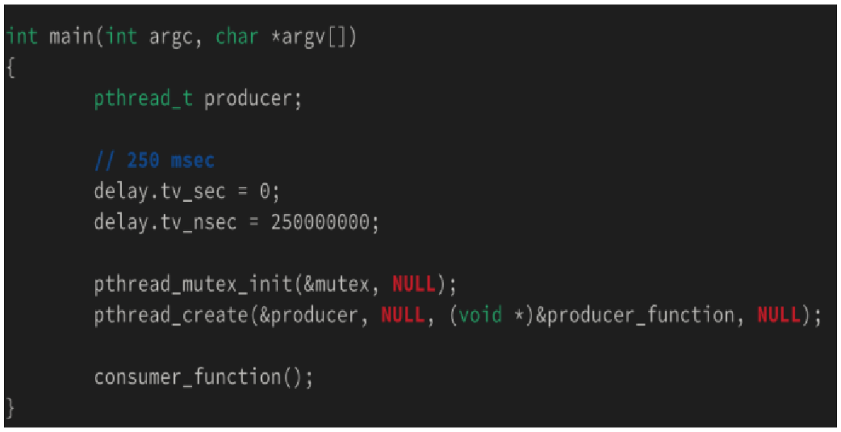
```
biancapinghireac@vbox:~/SO/lab7/src$ ./prodcons
producing 103
consuming 103
producing 198
consuming 198
producing 105
consuming 105
producing 115
consuming 115
producing 81
consuming 81
producing 255
consuming 255
producing 74
consuming 74
producing 236
consuming 236
producing 41
consuming 41
producing 205
consuming 205
```



```c
#include        <stdlib.h>
#include        <pthread.h>
#include        <stdio.h>

#define         ITEMS   10

long            buffer[ITEMS];
int             head = 0, tail = 0;

pthread_mutex_t mutex;
struct timespec delay;


long produce_item(void)
{
        long item = random() % 256;
        printf("producing %d\n", item);

        return item;
}

void consume_item(long item)
{
        printf("consuming %d\n", item);
}

void producer_function(void)
{
        while (1) {
                pthread_mutex_lock(&mutex);
                if ((tail + 1) % ITEMS != head) {
                        buffer[tail] = produce_item();
```



```
                if ((tail + 1) % ITEMS != head) {
                        buffer[tail] = produce_item();
                        tail = (tail + 1) % ITEMS;
                }
                pthread_mutex_unlock(&mutex);

                nanosleep(&delay, NULL);
        }
}

void consumer_function(void)
{
        while (1)
        {
                pthread_mutex_lock(&mutex);
                if (head != tail) {
                        consume_item(buffer[head]);
                        head = (head + 1) % ITEMS;
                }
                pthread_mutex_unlock(&mutex);
        }
}
```



```
int main(int argc, char *argv[])
{
        pthread_t producer;

        // 250 msec
        delay.tv_sec = 0;
        delay.tv_nsec = 250000000;

        pthread_mutex_init(&mutex, NULL);
        pthread_create(&producer, NULL, (void *)&producer_function, NULL);

        consumer_function();
}
```

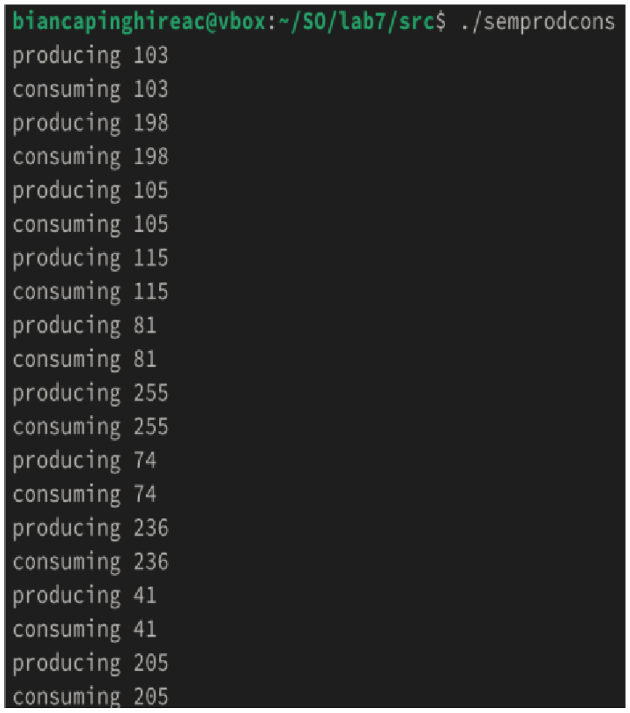
Prodcons.c

Programul ruleaza doua fire de executie: unul care produce articole si unul care consuma articolele produse.

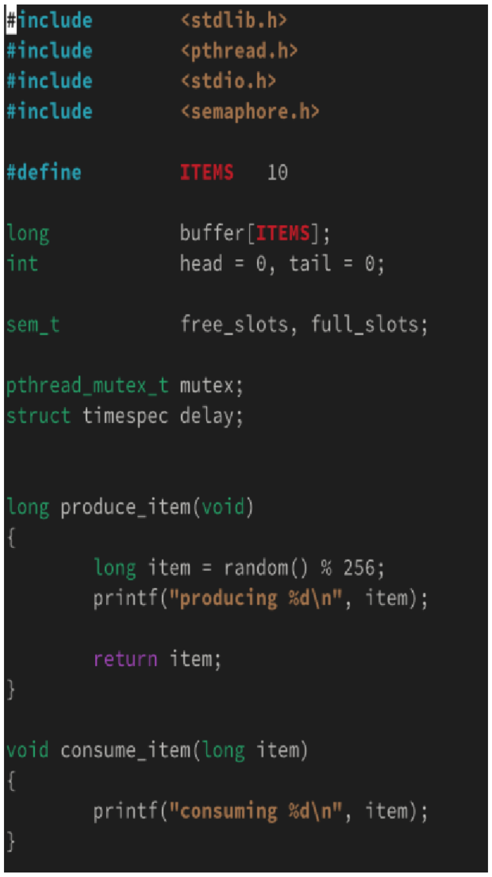
Output-ul arata cand un element este produs si cand este consumat.

Sincronizarea corecta este asigurata de mutex, prevenind accesul simultan la buffer.

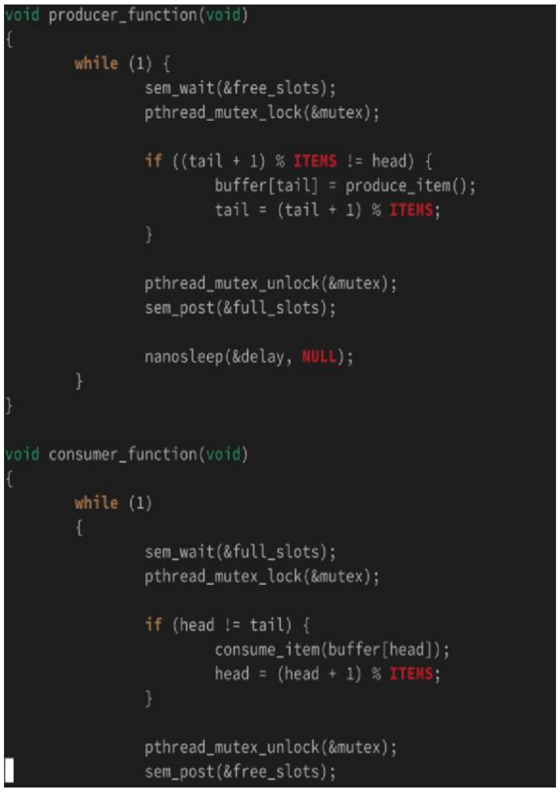
Programul simuleaza un scenariu unde producatorul și consumatorul trebuie sa partajeze resurse.



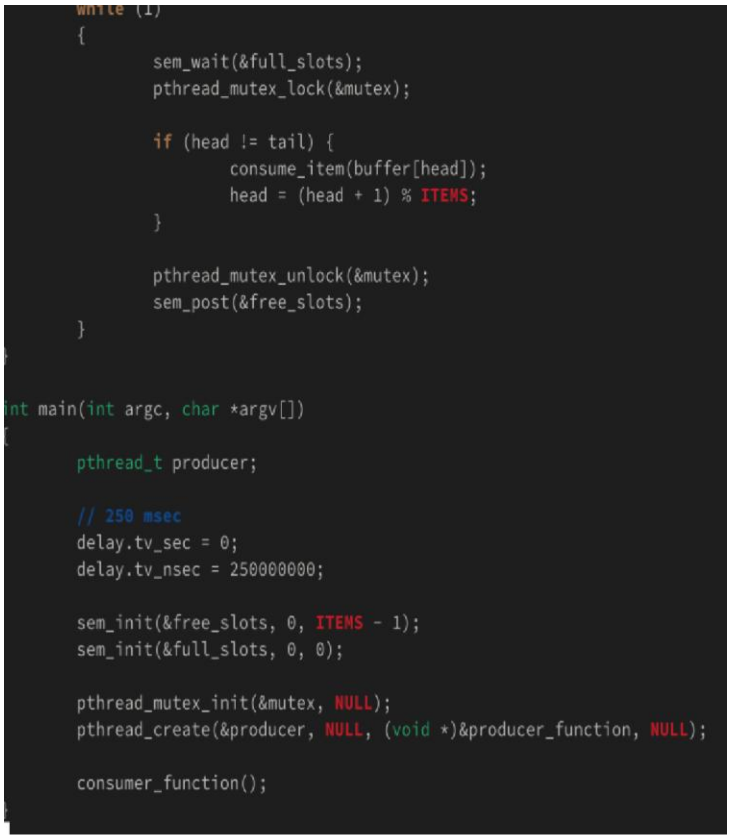
```
biancapinghireac@vbox:~/SO/lab7/src$ ./semprodcons
producing 103
consuming 103
producing 198
consuming 198
producing 105
consuming 105
producing 115
consuming 115
producing 81
consuming 81
producing 255
consuming 255
producing 74
consuming 74
producing 236
consuming 236
producing 41
consuming 41
producing 205
consuming 205
```



```
#include        <stdlib.h>
#include        <pthread.h>
#include        <stdio.h>
#include        <semaphore.h>

#define         ITEMS   10

long            buffer[ITEMS];
int             head = 0, tail = 0;

sem_t           free_slots, full_slots;

pthread_mutex_t mutex;
struct timespec delay;


long produce_item(void)
{
        long item = random() % 256;
        printf("producing %d\n", item);

        return item;
}

void consume_item(long item)
{
        printf("consuming %d\n", item);
}
```



```
void producer_function(void)
{
        while (1) {
                sem_wait(&free_slots);
                pthread_mutex_lock(&mutex);

                if ((tail + 1) % ITEMS != head) {
                        buffer[tail] = produce_item();
                        tail = (tail + 1) % ITEMS;
                }

                pthread_mutex_unlock(&mutex);
                sem_post(&full_slots);

                nanosleep(&delay, NULL);
        }
}

void consumer_function(void)
{
        while (1)
        {
                sem_wait(&full_slots);
                pthread_mutex_lock(&mutex);

                if (head != tail) {
                        consume_item(buffer[head]);
                        head = (head + 1) % ITEMS;
                }

                pthread_mutex_unlock(&mutex);
                sem_post(&free_slots);
```



```
        while (1)
        {
                sem_wait(&full_slots);
                pthread_mutex_lock(&mutex);

                if (head != tail) {
                        consume_item(buffer[head]);
                        head = (head + 1) % ITEMS;
                }

                pthread_mutex_unlock(&mutex);
                sem_post(&free_slots);
        }
}

int main(int argc, char *argv[])
{
        pthread_t producer;

        // 250 msec
        delay.tv_sec = 0;
        delay.tv_nsec = 250000000;

        sem_init(&free_slots, 0, ITEMS - 1);
        sem_init(&full_slots, 0, 0);

        pthread_mutex_init(&mutex, NULL);
        pthread_create(&producer, NULL, (void *)&producer_function, NULL);

        consumer_function();
}
```

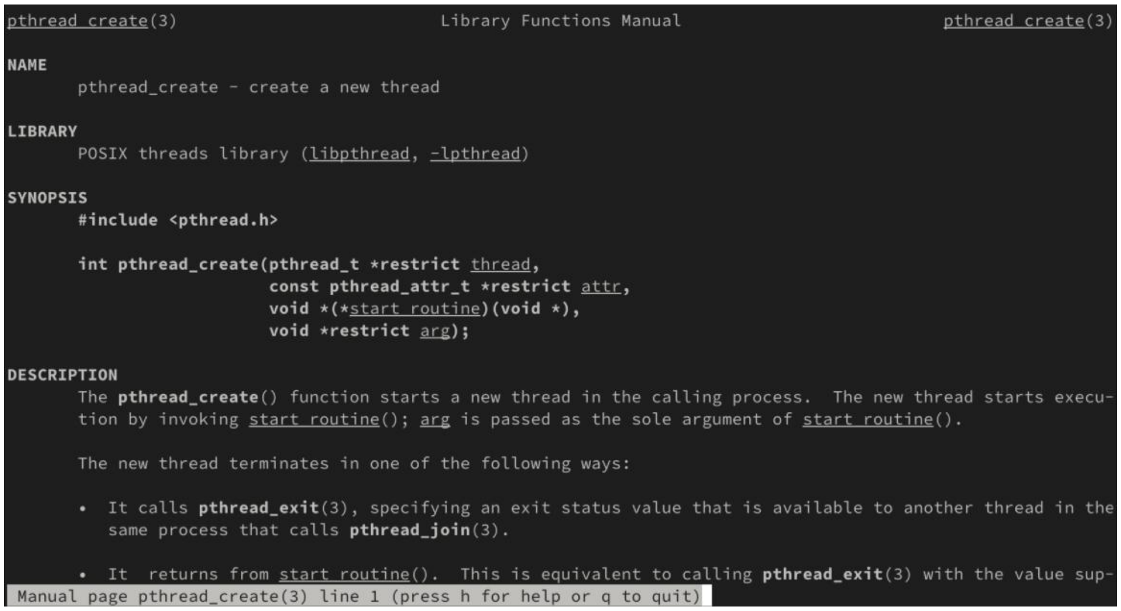
Semprodcons.c

Programul produce si consuma elemente folosind semafoare pentru a asigura sincronizarea corecta intre producator si consumator.

free_slots si **full_slots** gestioneaza cate elemente pot fi produse sau consumate

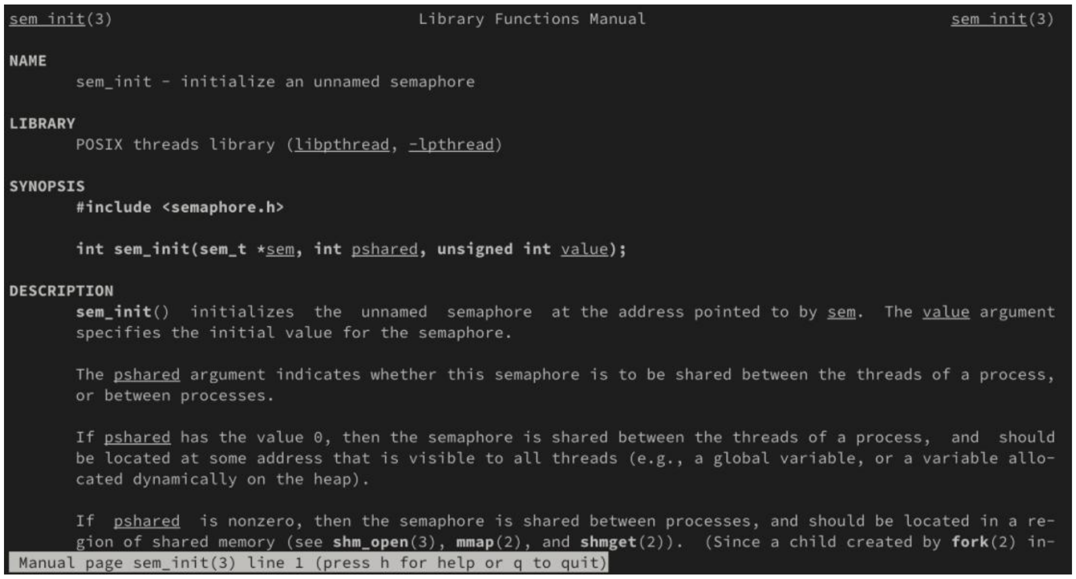
Asigura ca un producator nu va depasi capacitatea bufferului si ca un consumator va astepta pana la aparitia unui element disponibil.

Pagini de manual:



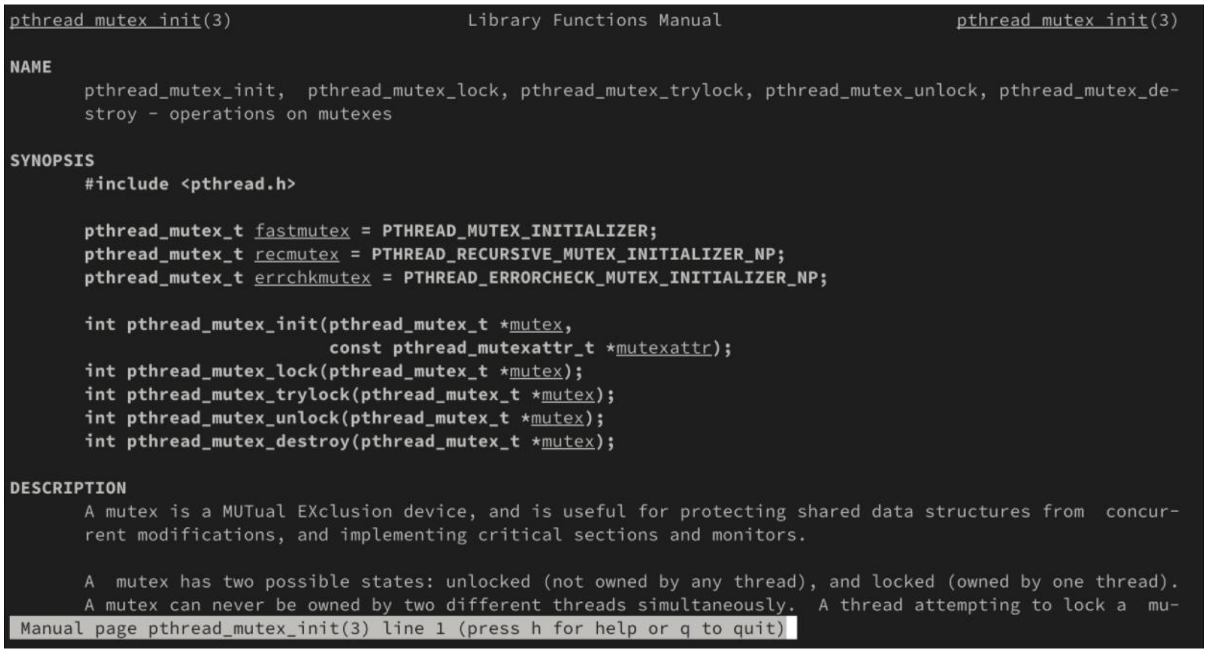
```
pthread_create(3)                Library Functions Manual                pthread_create(3)

NAME
       pthread_create - create a new thread

LIBRARY
       POSIX threads library (libpthread, -lpthread)

SYNOPSIS
       #include <pthread.h>

       int pthread_create(pthread_t *restrict thread,
                          const pthread_attr_t *restrict attr,
                          void *(*start_routine)(void *),
                          void *restrict arg);

DESCRIPTION
       The pthread_create() function starts a new thread in the calling process.  The new thread starts execu-
       tion by invoking start_routine(); arg is passed as the sole argument of start_routine().

       The new thread terminates in one of the following ways:

       •  It calls pthread_exit(3), specifying an exit status value that is available to another thread in the
          same process that calls pthread_join(3).

       •  It  returns from start_routine().  This is equivalent to calling pthread_exit(3) with the value sup-
 Manual page pthread_create(3) line 1 (press h for help or q to quit)
```

Pthread_create



```
sem_init(3)                      Library Functions Manual                      sem_init(3)

NAME
       sem_init - initialize an unnamed semaphore

LIBRARY
       POSIX threads library (libpthread, -lpthread)

SYNOPSIS
       #include <semaphore.h>

       int sem_init(sem_t *sem, int pshared, unsigned int value);

DESCRIPTION
       sem_init()  initializes  the  unnamed  semaphore  at the address pointed to by sem.  The value argument
       specifies the initial value for the semaphore.

       The pshared argument indicates whether this semaphore is to be shared between the threads of a process,
       or between processes.

       If pshared has the value 0, then the semaphore is shared between the threads of a process,  and  should
       be located at some address that is visible to all threads (e.g., a global variable, or a variable allo-
       cated dynamically on the heap).

       If  pshared  is nonzero, then the semaphore is shared between processes, and should be located in a re-
       gion of shared memory (see shm_open(3), mmap(2), and shmget(2)).  (Since a child created by fork(2) in-
 Manual page sem_init(3) line 1 (press h for help or q to quit)
```

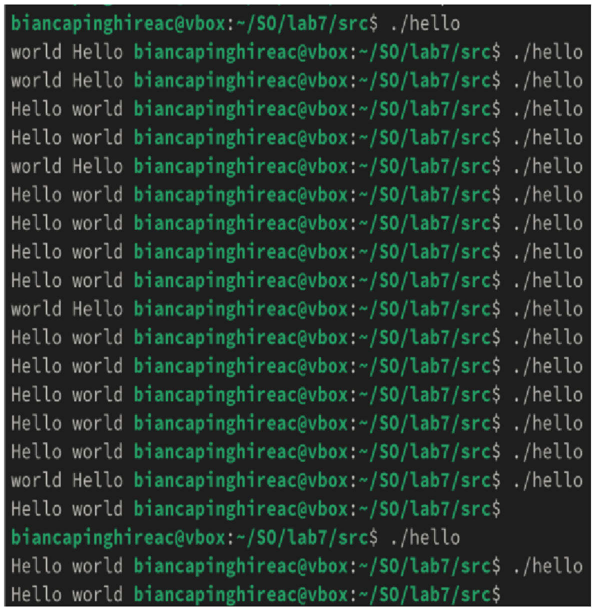
Sem_init



```
pthread_mutex_init(3)              Library Functions Manual              pthread_mutex_init(3)

NAME
       pthread_mutex_init,  pthread_mutex_lock, pthread_mutex_trylock, pthread_mutex_unlock, pthread_mutex_de-
       stroy - operations on mutexes

SYNOPSIS
       #include <pthread.h>

       pthread_mutex_t fastmutex = PTHREAD_MUTEX_INITIALIZER;
       pthread_mutex_t recmutex = PTHREAD_RECURSIVE_MUTEX_INITIALIZER_NP;
       pthread_mutex_t errchkmutex = PTHREAD_ERRORCHECK_MUTEX_INITIALIZER_NP;

       int pthread_mutex_init(pthread_mutex_t *mutex,
                              const pthread_mutexattr_t *mutexattr);
       int pthread_mutex_lock(pthread_mutex_t *mutex);
       int pthread_mutex_trylock(pthread_mutex_t *mutex);
       int pthread_mutex_unlock(pthread_mutex_t *mutex);
       int pthread_mutex_destroy(pthread_mutex_t *mutex);

DESCRIPTION
       A mutex is a MUTual EXclusion device, and is useful for protecting shared data structures from  concur-
       rent modifications, and implementing critical sections and monitors.

       A  mutex has two possible states: unlocked (not owned by any thread), and locked (owned by one thread).
       A mutex can never be owned by two different threads simultaneously.  A thread attempting to lock a  mu-
 Manual page pthread_mutex_init(3) line 1 (press h for help or q to quit)
```

pthread_mutex_init

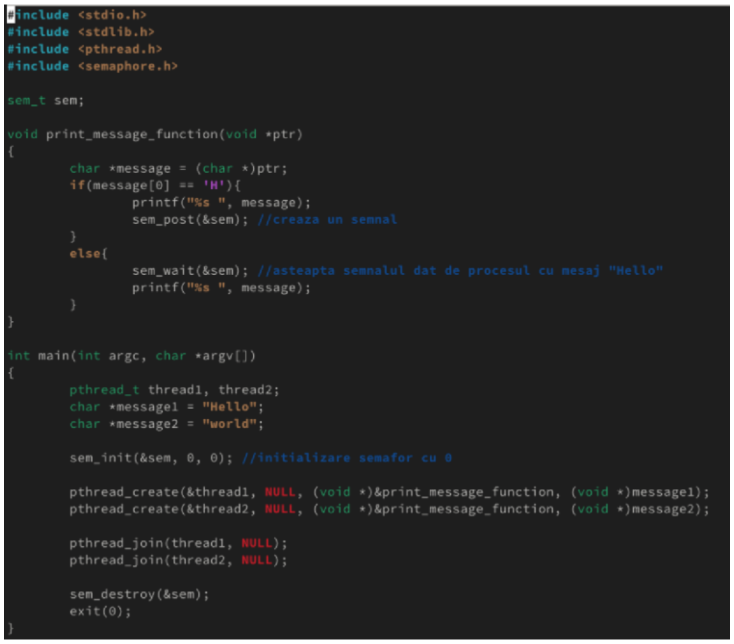
EXERCITIUL 2:



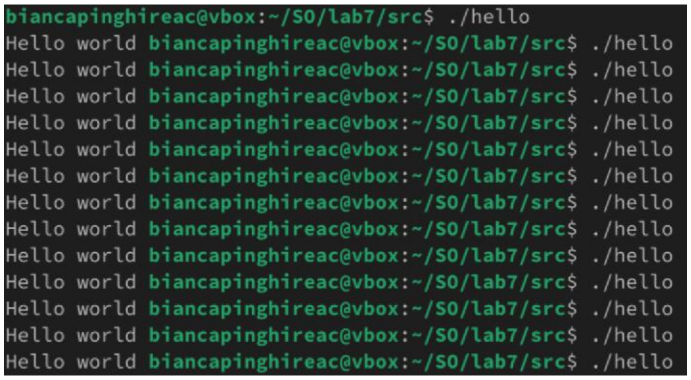
```
biancapinghireac@vbox:~/SO/lab7/src$ ./hello
world Hello biancapinghireac@vbox:~/SO/lab7/src$ ./hello
world Hello biancapinghireac@vbox:~/SO/lab7/src$ ./hello
Hello world biancapinghireac@vbox:~/SO/lab7/src$ ./hello
Hello world biancapinghireac@vbox:~/SO/lab7/src$ ./hello
world Hello biancapinghireac@vbox:~/SO/lab7/src$ ./hello
Hello world biancapinghireac@vbox:~/SO/lab7/src$ ./hello
Hello world biancapinghireac@vbox:~/SO/lab7/src$ ./hello
Hello world biancapinghireac@vbox:~/SO/lab7/src$ ./hello
Hello world biancapinghireac@vbox:~/SO/lab7/src$ ./hello
world Hello biancapinghireac@vbox:~/SO/lab7/src$ ./hello
Hello world biancapinghireac@vbox:~/SO/lab7/src$ ./hello
Hello world biancapinghireac@vbox:~/SO/lab7/src$ ./hello
Hello world biancapinghireac@vbox:~/SO/lab7/src$ ./hello
Hello world biancapinghireac@vbox:~/SO/lab7/src$ ./hello
Hello world biancapinghireac@vbox:~/SO/lab7/src$ ./hello
world Hello biancapinghireac@vbox:~/SO/lab7/src$ ./hello
Hello world biancapinghireac@vbox:~/SO/lab7/src$
biancapinghireac@vbox:~/SO/lab7/src$ ./hello
Hello world biancapinghireac@vbox:~/SO/lab7/src$ ./hello
Hello world biancapinghireac@vbox:~/SO/lab7/src$
```

Putem observa faptul ca nedeterminarea create este nesincronizarea thread-urilor, output-ul este cand “Hello world”, cand “word Hello”, acest lucru se poate corecta prin folosirea unui mutex sau semafor.

Rezolvare problema prin folosrea unui semafor:

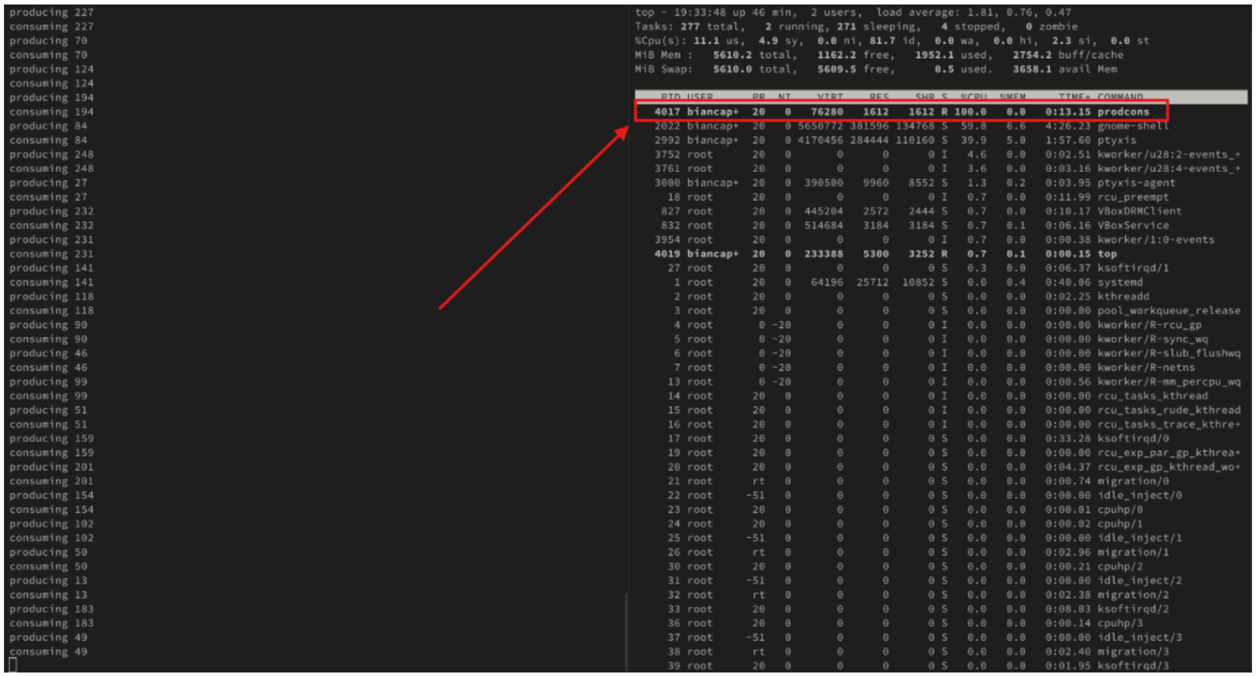


```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>

sem_t sem;

void print_message_function(void *ptr)
{
        char *message = (char *)ptr;
        if(message[0] == 'H'){
                printf("%s ", message);
                sem_post(&sem); //creaza un semnal
        }
        else{
                sem_wait(&sem); //asteapta semnalul dat de procesul cu mesaj "Hello"
                printf("%s ", message);
        }
}

int main(int argc, char *argv[])
{
        pthread_t thread1, thread2;
        char *message1 = "Hello";
        char *message2 = "world";

        sem_init(&sem, 0, 0); //initializare semafor cu 0

        pthread_create(&thread1, NULL, (void *)&print_message_function, (void *)message1);
        pthread_create(&thread2, NULL, (void *)&print_message_function, (void *)message2);

        pthread_join(thread1, NULL);
        pthread_join(thread2, NULL);

        sem_destroy(&sem);
        exit(0);
}
```



```
biancapinghireac@vbox:~/SO/lab7/src$ ./hello
Hello world biancapinghireac@vbox:~/SO/lab7/src$ ./hello
Hello world biancapinghireac@vbox:~/SO/lab7/src$ ./hello
Hello world biancapinghireac@vbox:~/SO/lab7/src$ ./hello
Hello world biancapinghireac@vbox:~/SO/lab7/src$ ./hello
Hello world biancapinghireac@vbox:~/SO/lab7/src$ ./hello
Hello world biancapinghireac@vbox:~/SO/lab7/src$ ./hello
Hello world biancapinghireac@vbox:~/SO/lab7/src$ ./hello
Hello world biancapinghireac@vbox:~/SO/lab7/src$ ./hello
Hello world biancapinghireac@vbox:~/SO/lab7/src$ ./hello
Hello world biancapinghireac@vbox:~/SO/lab7/src$ ./hello
Hello world biancapinghireac@vbox:~/SO/lab7/src$ ./hello
Hello world biancapinghireac@vbox:~/SO/lab7/src$ ./hello
Hello world biancapinghireac@vbox:~/SO/lab7/src$ ./hello
```

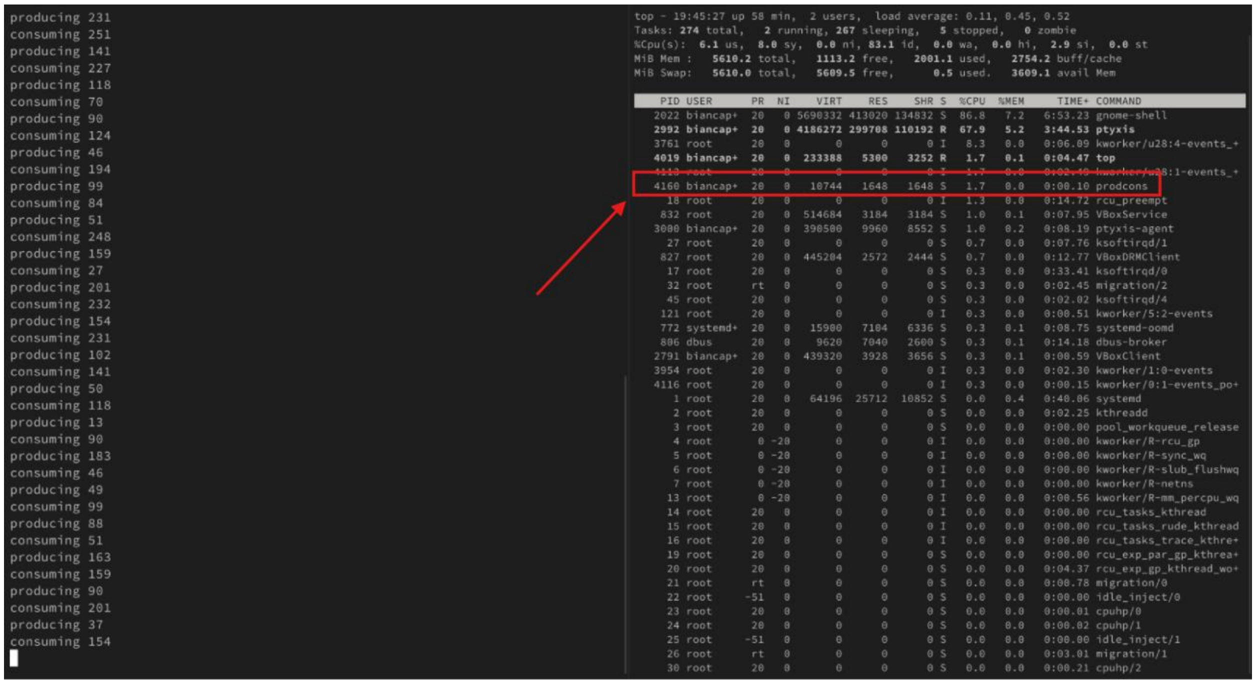
EXERCITIUL 3:



```
producing 227
consuming 227
producing 70
consuming 70
producing 124
consuming 124
producing 194
consuming 194
producing 84
consuming 84
producing 248
consuming 248
producing 27
consuming 27
producing 232
consuming 232
producing 231
consuming 231
producing 141
consuming 141
producing 118
consuming 118
producing 90
consuming 90
producing 46
consuming 46
producing 99
consuming 99
producing 51
consuming 51
producing 159
consuming 159
producing 201
consuming 201
producing 154
consuming 154
producing 102
consuming 102
producing 50
consuming 50
producing 13
consuming 13
producing 183
consuming 183
producing 49
consuming 49
```

```
top - 19:33:48 up 46 min,  2 users,  load average: 1.81, 0.76, 0.47
Tasks: 277 total,   2 running, 271 sleeping,   4 stopped,   0 zombie
%Cpu(s): 11.1 us,  4.9 sy,  0.0 ni, 81.7 id,  0.0 wa,  0.0 hi,  2.3 si,  0.0 st
MiB Mem :   5610.2 total,   1162.2 free,   1952.1 used,   2754.2 buff/cache
MiB Swap:   5610.0 total,   5609.5 free,      0.5 used.   3658.1 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   4017 biancap+  20   0   76280   1612   1612 R 100.0   0.0   0:13.15 prodcons
   2022 biancap+  20   0 5650772 381596 134768 S  59.8   6.6   4:26.23 gnome-shell
   2992 biancap+  20   0 4170456 284444 110160 S  39.9   5.0   1:57.60 ptyxis
   3752 root      20   0       0      0      0 I   4.6   0.0   0:02.51 kworker/u28:2-events_+
   3761 root      20   0       0      0      0 I   3.6   0.0   0:03.16 kworker/u28:4-events_+
   3000 biancap+  20   0  390500   9960   8552 S   1.3   0.2   0:03.95 ptyxis-agent
     18 root      20   0       0      0      0 I   0.7   0.0   0:11.99 rcu_preempt
    827 root      20   0  445204   2572   2444 S   0.7   0.0   0:10.17 VBoxDRMClient
    832 root      20   0  514684   3184   3184 S   0.7   0.1   0:06.16 VBoxService
   3954 root      20   0       0      0      0 I   0.7   0.0   0:00.38 kworker/1:0-events
   4019 biancap+  20   0  233388   5300   3252 R   0.7   0.1   0:00.15 top
     27 root      20   0       0      0      0 S   0.3   0.0   0:06.37 ksoftirqd/1
      1 root      20   0   64196  25712  10852 S   0.0   0.4   0:40.06 systemd
      2 root      20   0       0      0      0 S   0.0   0.0   0:02.25 kthreadd
      3 root      20   0       0      0      0 S   0.0   0.0   0:00.00 pool_workqueue_release
      4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/R-rcu_gp
      5 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/R-sync_wq
      6 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/R-slub_flushwq
      7 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/R-netns
     13 root       0 -20       0      0      0 I   0.0   0.0   0:00.56 kworker/R-mm_percpu_wq
     14 root      20   0       0      0      0 I   0.0   0.0   0:00.00 rcu_tasks_kthread
     15 root      20   0       0      0      0 I   0.0   0.0   0:00.00 rcu_tasks_rude_kthread
     16 root      20   0       0      0      0 I   0.0   0.0   0:00.00 rcu_tasks_trace_kthre+
     17 root      20   0       0      0      0 S   0.0   0.0   0:33.28 ksoftirqd/0
     19 root      20   0       0      0      0 S   0.0   0.0   0:00.00 rcu_exp_par_gp_kthrea+
     20 root      20   0       0      0      0 S   0.0   0.0   0:04.37 rcu_exp_gp_kthread_wo+
     21 root      rt   0       0      0      0 S   0.0   0.0   0:00.74 migration/0
     22 root     -51   0       0      0      0 S   0.0   0.0   0:00.00 idle_inject/0
     23 root      20   0       0      0      0 S   0.0   0.0   0:00.01 cpuhp/0
     24 root      20   0       0      0      0 S   0.0   0.0   0:00.02 cpuhp/1
     25 root     -51   0       0      0      0 S   0.0   0.0   0:00.00 idle_inject/1
     26 root      rt   0       0      0      0 S   0.0   0.0   0:02.96 migration/1
     30 root      20   0       0      0      0 S   0.0   0.0   0:00.21 cpuhp/2
     31 root     -51   0       0      0      0 S   0.0   0.0   0:00.00 idle_inject/2
     32 root      rt   0       0      0      0 S   0.0   0.0   0:02.38 migration/2
     33 root      20   0       0      0      0 S   0.0   0.0   0:08.03 ksoftirqd/2
     36 root      20   0       0      0      0 S   0.0   0.0   0:00.14 cpuhp/3
     37 root     -51   0       0      0      0 S   0.0   0.0   0:00.00 idle_inject/3
     38 root      rt   0       0      0      0 S   0.0   0.0   0:02.40 migration/3
     39 root      20   0       0      0      0 S   0.0   0.0   0:01.95 ksoftirqd/3
```

Inainte de modificare

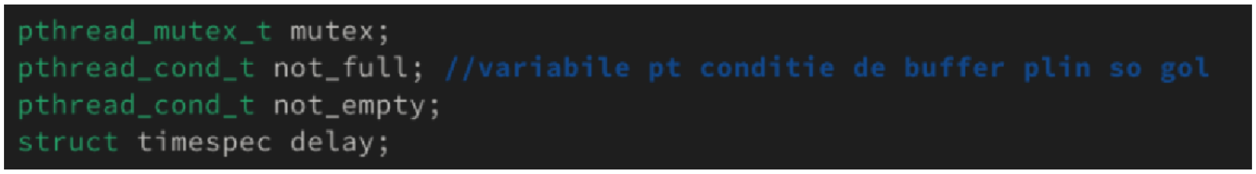


```
producing 231
consuming 251
producing 141
consuming 227
producing 118
consuming 70
producing 90
consuming 124
producing 46
consuming 194
producing 99
consuming 84
producing 51
consuming 248
producing 159
consuming 27
producing 201
consuming 232
producing 154
consuming 231
producing 102
consuming 141
producing 50
consuming 118
producing 13
consuming 90
producing 183
consuming 46
producing 49
consuming 99
producing 88
consuming 51
producing 163
consuming 159
producing 90
consuming 201
producing 37
consuming 154
```

```
top - 19:45:27 up 58 min,  2 users,  load average: 0.11, 0.45, 0.52
Tasks: 274 total,   2 running, 267 sleeping,   5 stopped,   0 zombie
%Cpu(s):  6.1 us,  8.0 sy,  0.0 ni, 83.1 id,  0.0 wa,  0.0 hi,  2.9 si,  0.0 st
MiB Mem :   5610.2 total,   1113.2 free,   2001.1 used,   2754.2 buff/cache
MiB Swap:   5610.0 total,   5609.5 free,      0.5 used.   3609.1 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   2022 biancap+  20   0 5690332 413020 134832 S  86.8   7.2   6:53.23 gnome-shell
   2992 biancap+  20   0 4186272 299708 110192 R  67.9   5.2   3:44.53 ptyxis
   3761 root      20   0       0      0      0 I   8.3   0.0   0:06.09 kworker/u28:4-events_+
   4019 biancap+  20   0  233388   5300   3252 R   1.7   0.1   0:04.47 top
   4113 root      20   0       0      0      0 I   1.7   0.0   0:02.49 kworker/u28:1-events_+
   4160 biancap+  20   0   10744   1648   1648 S   1.7   0.0   0:00.10 prodcons
     18 root      20   0       0      0      0 I   1.3   0.0   0:14.72 rcu_preempt
    832 root      20   0  514684   3184   3184 S   1.0   0.1   0:07.95 VBoxService
   3080 biancap+  20   0  390500   9960   8552 S   1.0   0.2   0:08.19 ptyxis-agent
     27 root      20   0       0      0      0 S   0.7   0.0   0:07.76 ksoftirqd/1
    827 root      20   0  445204   2572   2444 S   0.7   0.0   0:12.77 VBoxDRMClient
     17 root      20   0       0      0      0 S   0.3   0.0   0:33.41 ksoftirqd/0
     32 root      rt   0       0      0      0 S   0.3   0.0   0:02.45 migration/2
     45 root      20   0       0      0      0 S   0.3   0.0   0:02.02 ksoftirqd/4
    121 root      20   0       0      0      0 I   0.3   0.0   0:00.51 kworker/5:2-events
    772 systemd+  20   0   15900   7104   6336 S   0.3   0.1   0:08.75 systemd-oomd
    806 dbus      20   0    9620   7040   2600 S   0.3   0.1   0:14.18 dbus-broker
   2791 biancap+  20   0  439320   3928   3656 S   0.3   0.1   0:00.59 VBoxClient
   3954 root      20   0       0      0      0 I   0.3   0.0   0:02.30 kworker/1:0-events
   4116 root      20   0       0      0      0 I   0.3   0.0   0:00.15 kworker/0:1-events_po+
      1 root      20   0   64196  25712  10852 S   0.0   0.4   0:40.06 systemd
      2 root      20   0       0      0      0 S   0.0   0.0   0:02.25 kthreadd
      3 root      20   0       0      0      0 S   0.0   0.0   0:00.00 pool_workqueue_release
      4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/R-rcu_gp
      5 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/R-sync_wq
      6 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/R-slub_flushwq
      7 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/R-netns
     13 root       0 -20       0      0      0 I   0.0   0.0   0:00.56 kworker/R-mm_percpu_wq
     14 root      20   0       0      0      0 I   0.0   0.0   0:00.00 rcu_tasks_kthread
     15 root      20   0       0      0      0 I   0.0   0.0   0:00.00 rcu_tasks_rude_kthread
     16 root      20   0       0      0      0 I   0.0   0.0   0:00.00 rcu_tasks_trace_kthre+
     19 root      20   0       0      0      0 S   0.0   0.0   0:00.00 rcu_exp_par_gp_kthrea+
     20 root      20   0       0      0      0 S   0.0   0.0   0:04.37 rcu_exp_gp_kthread_wo+
     21 root      rt   0       0      0      0 S   0.0   0.0   0:00.78 migration/0
     22 root     -51   0       0      0      0 S   0.0   0.0   0:00.00 idle_inject/0
     23 root      20   0       0      0      0 S   0.0   0.0   0:00.01 cpuhp/0
     24 root      20   0       0      0      0 S   0.0   0.0   0:00.02 cpuhp/1
     25 root     -51   0       0      0      0 S   0.0   0.0   0:00.00 idle_inject/1
     26 root      rt   0       0      0      0 S   0.0   0.0   0:03.01 migration/1
     30 root      20   0       0      0      0 S   0.0   0.0   0:00.21 cpuhp/2
```

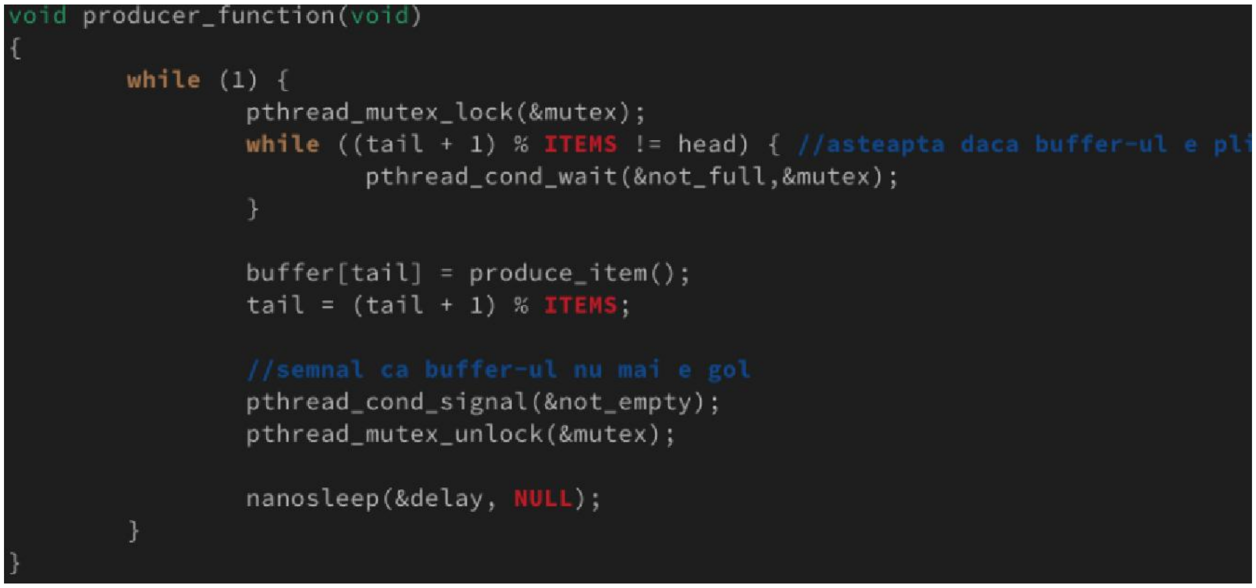
Dupa modificare

Cod modificat:



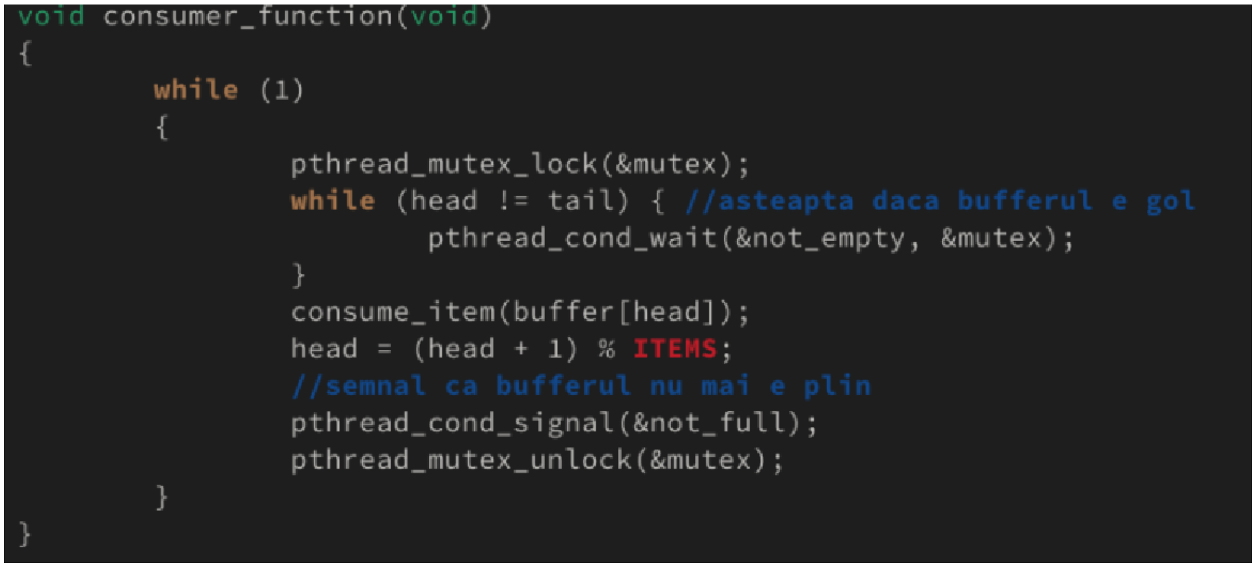
```c
pthread_mutex_t mutex;
pthread_cond_t not_full; //variabile pt conditie de buffer plin so gol
pthread_cond_t not_empty;
struct timespec delay;
```

Am adaugat variabile de conditie buffer gol/plin



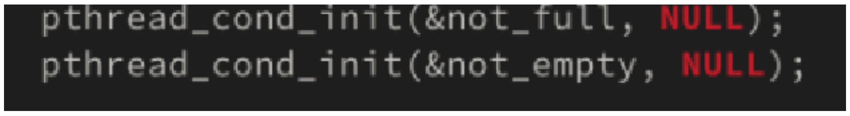
```c
void producer_function(void)
{
        while (1) {
                pthread_mutex_lock(&mutex);
                while ((tail + 1) % ITEMS != head) { //asteapta daca buffer-ul e pl
                        pthread_cond_wait(&not_full,&mutex);
                }

                buffer[tail] = produce_item();
                tail = (tail + 1) % ITEMS;

                //semnal ca buffer-ul nu mai e gol
                pthread_cond_signal(&not_empty);
                pthread_mutex_unlock(&mutex);

                nanosleep(&delay, NULL);
        }
}
```

Am modificat functia de producere, acum se asteapta pana cand buffer-ul nu mai este plin, pentru a nu ii da overload. Acest proces adauga in buffer, deci acesta il semnaleaza ca nu fiind gol.



```
void consumer_function(void)
{
        while (1)
        {
                pthread_mutex_lock(&mutex);
                while (head != tail) { //asteapta daca bufferul e gol
                        pthread_cond_wait(&not_empty, &mutex);
                }
                consume_item(buffer[head]);
                head = (head + 1) % ITEMS;
                //semnal ca bufferul nu mai e plin
                pthread_cond_signal(&not_full);
                pthread_mutex_unlock(&mutex);
        }
}
```

Similar, am schimbat functia de consummator, acum asteapta daca buffer-ul e gol (deoarce nu are ce consuma). Deoarece se consuma un produs, putem semnala ca nu mai este plin buffer-ul.

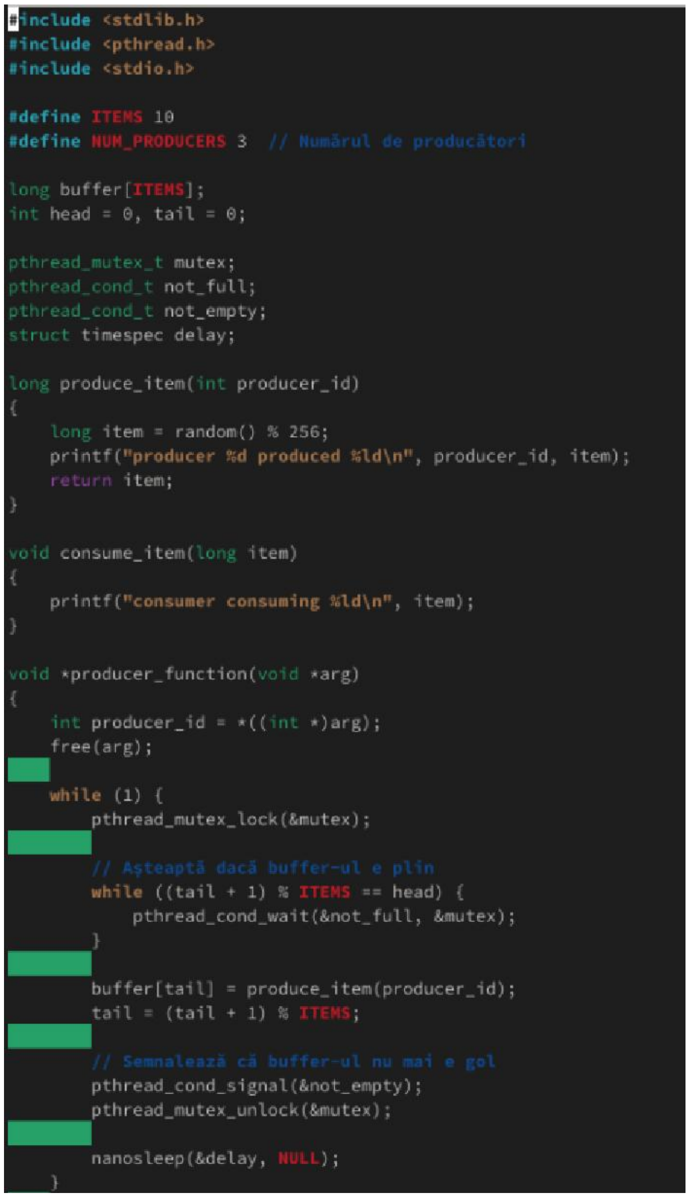


```
pthread_cond_init(&not_full, NULL);
pthread_cond_init(&not_empty, NULL);
```

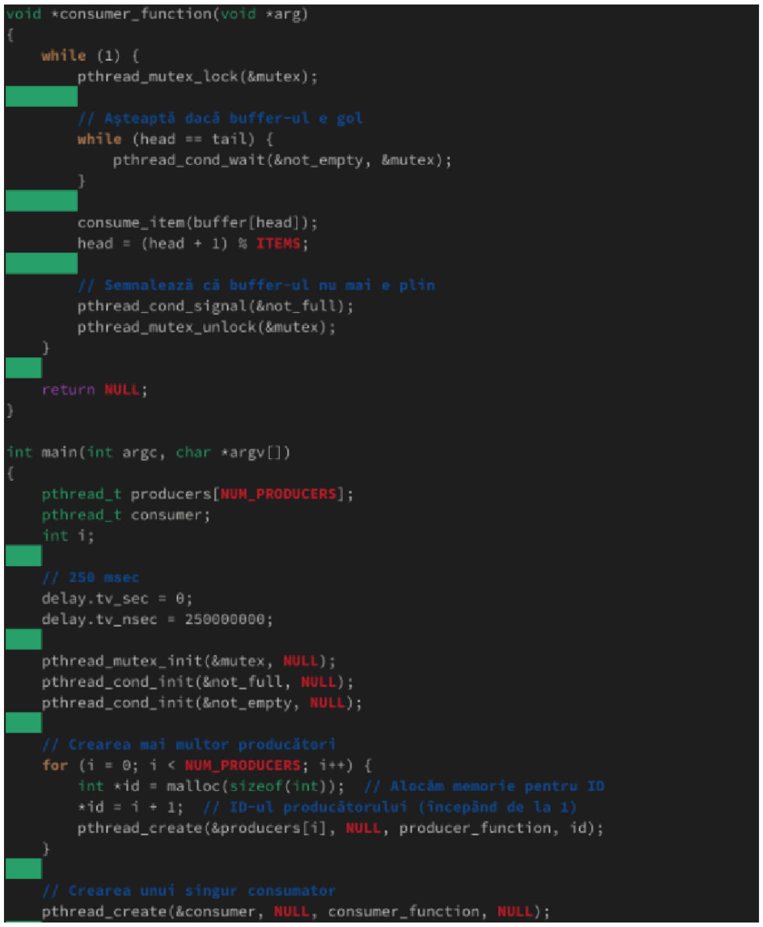
In main initializam cele 2 valori cu NULL.

In rest codul ramne la fel.

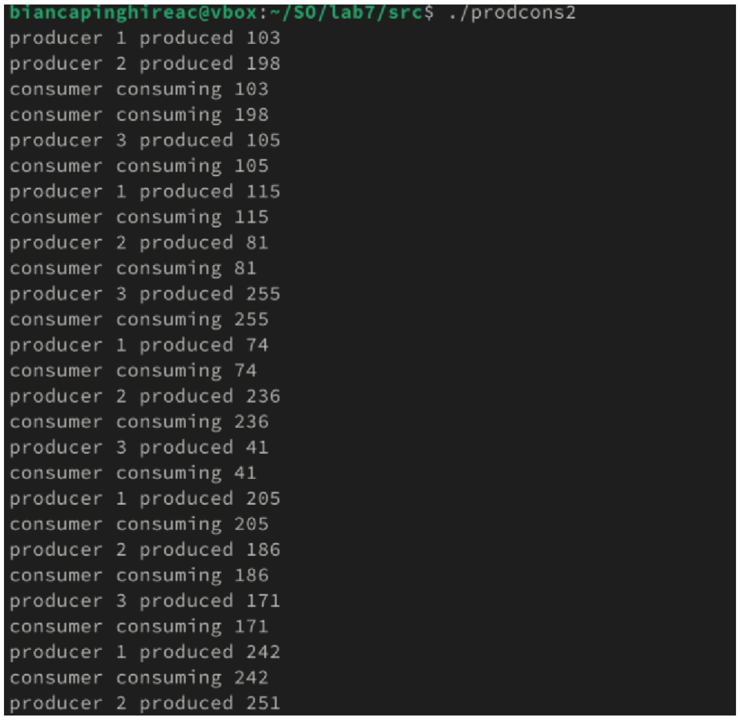
EXERCITIUL 4:



```c
#include <stdlib.h>
#include <pthread.h>
#include <stdio.h>

#define ITEMS 10
#define NUM_PRODUCERS 3  // Numărul de producători

long buffer[ITEMS];
int head = 0, tail = 0;

pthread_mutex_t mutex;
pthread_cond_t not_full;
pthread_cond_t not_empty;
struct timespec delay;

long produce_item(int producer_id)
{
    long item = random() % 256;
    printf("producer %d produced %ld\n", producer_id, item);
    return item;
}

void consume_item(long item)
{
    printf("consumer consuming %ld\n", item);
}

void *producer_function(void *arg)
{
    int producer_id = *((int *)arg);
    free(arg);
    
    while (1) {
        pthread_mutex_lock(&mutex);
        
        // Așteaptă dacă buffer-ul e plin
        while ((tail + 1) % ITEMS == head) {
            pthread_cond_wait(&not_full, &mutex);
        }
        
        buffer[tail] = produce_item(producer_id);
        tail = (tail + 1) % ITEMS;
        
        // Semnalează că buffer-ul nu mai e gol
        pthread_cond_signal(&not_empty);
        pthread_mutex_unlock(&mutex);
        
        nanosleep(&delay, NULL);
    }
```



```c
void *consumer_function(void *arg)
{
    while (1) {
        pthread_mutex_lock(&mutex);

        // Așteaptă dacă buffer-ul e gol
        while (head == tail) {
            pthread_cond_wait(&not_empty, &mutex);
        }

        consume_item(buffer[head]);
        head = (head + 1) % ITEMS;

        // Semnalează că buffer-ul nu mai e plin
        pthread_cond_signal(&not_full);
        pthread_mutex_unlock(&mutex);
    }

    return NULL;
}

int main(int argc, char *argv[])
{
    pthread_t producers[NUM_PRODUCERS];
    pthread_t consumer;
    int i;

    // 250 msec
    delay.tv_sec = 0;
    delay.tv_nsec = 250000000;

    pthread_mutex_init(&mutex, NULL);
    pthread_cond_init(&not_full, NULL);
    pthread_cond_init(&not_empty, NULL);

    // Crearea mai multor producători
    for (i = 0; i < NUM_PRODUCERS; i++) {
        int *id = malloc(sizeof(int));  // Alocăm memorie pentru ID
        *id = i + 1;  // ID-ul producătorului (începând de la 1)
        pthread_create(&producers[i], NULL, producer_function, id);
    }

    // Crearea unui singur consumator
    pthread_create(&consumer, NULL, consumer_function, NULL);
```



```
biancapinghireac@vbox:~/SO/lab7/src$ ./prodcons2
producer 1 produced 103
producer 2 produced 198
consumer consuming 103
consumer consuming 198
producer 3 produced 105
consumer consuming 105
producer 1 produced 115
consumer consuming 115
producer 2 produced 81
consumer consuming 81
producer 3 produced 255
consumer consuming 255
producer 1 produced 74
consumer consuming 74
producer 2 produced 236
consumer consuming 236
producer 3 produced 41
consumer consuming 41
producer 1 produced 205
consumer consuming 205
producer 2 produced 186
consumer consuming 186
producer 3 produced 171
consumer consuming 171
producer 1 produced 242
consumer consuming 242
producer 2 produced 251
```

Am definit o constanta **NUM_PRODUCERS** pentru a specifica numarul de producatori.

Am modificat functia **produce_item()** pentru a accepta un ID de producator, permitand identificarea producatorului in mesajele afisate.

Am modificat functia producatorului pentru a primi ID-ul producatorului ca parametru.

Am creat un array de thread-uri pentru producatori in loc de un singur thread.

Am folosit un loop pentru a crea mai multe thread-uri de producator, fiecare cu propriul ID unic.

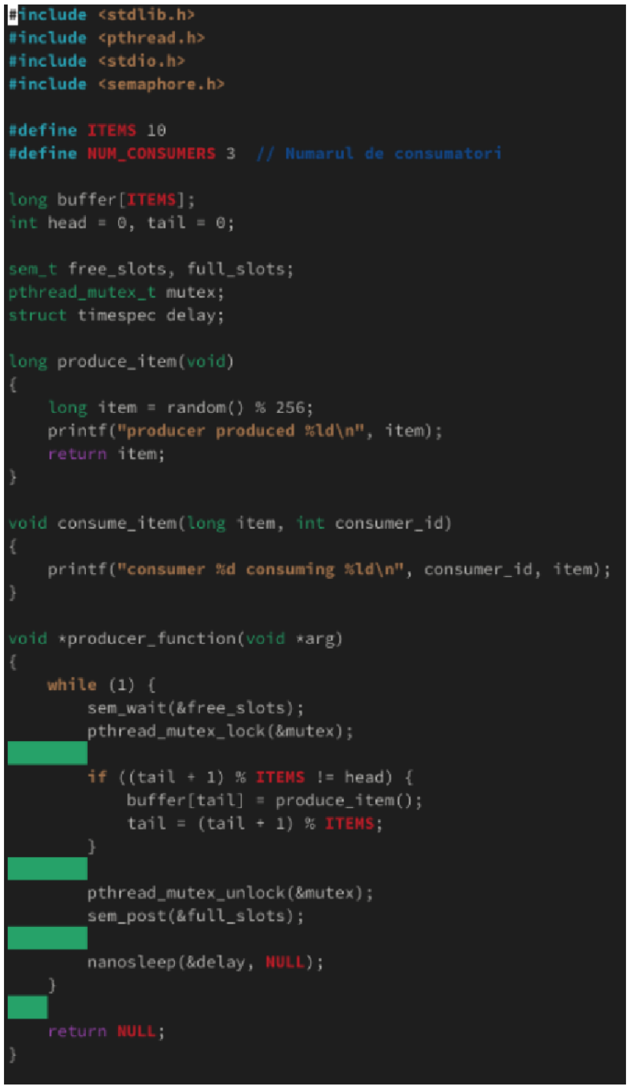
Am alocat dinamic memoria pentru ID-urile producatorilor pentru a preveni problemele de partajare a valorilor.

Am mentinut un singur thread pentru consumator, care proceseaza toate elementele produse.

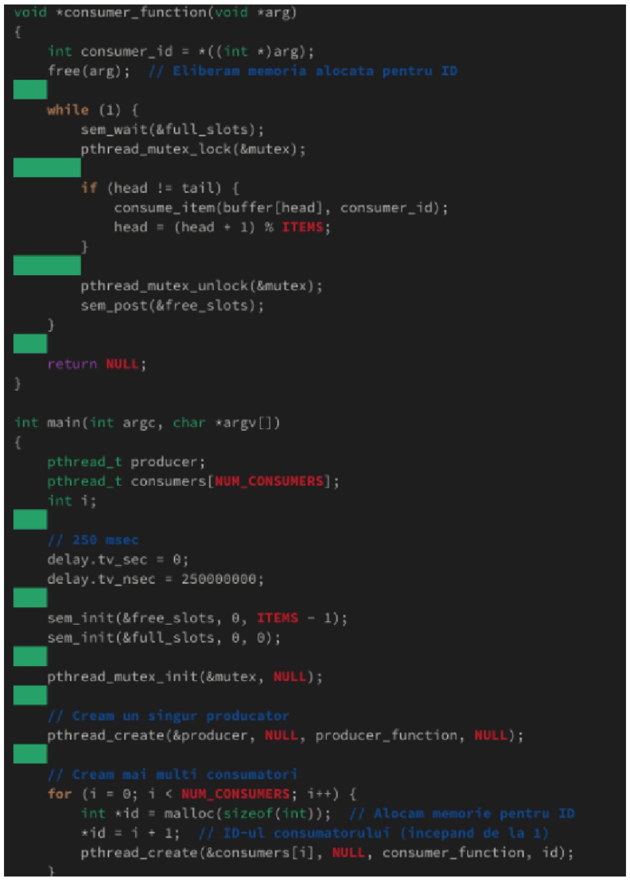
Am asigurat sincronizarea corecta prin mutex si variabile conditionale, astfel incat toti producatorii sa poata adauga elemente in buffer.

EXERCITIUL 5:

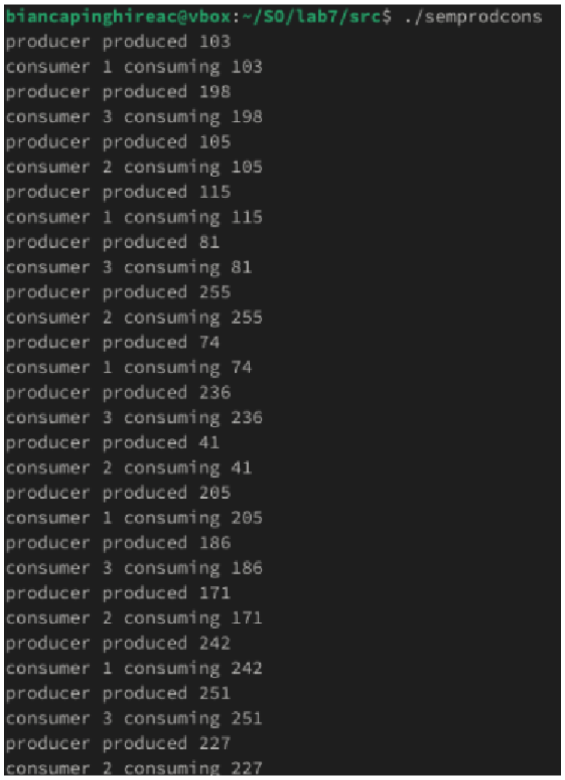
Similar cu exercitiul precedent, dar in loc de mai multi producatori avem mai multi consumatori si semafor.



```c
#include <stdlib.h>
#include <pthread.h>
#include <stdio.h>
#include <semaphore.h>

#define ITEMS 10
#define NUM_CONSUMERS 3  // Numarul de consumatori

long buffer[ITEMS];
int head = 0, tail = 0;

sem_t free_slots, full_slots;
pthread_mutex_t mutex;
struct timespec delay;

long produce_item(void)
{
    long item = random() % 256;
    printf("producer produced %ld\n", item);
    return item;
}

void consume_item(long item, int consumer_id)
{
    printf("consumer %d consuming %ld\n", consumer_id, item);
}

void *producer_function(void *arg)
{
    while (1) {
        sem_wait(&free_slots);
        pthread_mutex_lock(&mutex);

        if ((tail + 1) % ITEMS != head) {
            buffer[tail] = produce_item();
            tail = (tail + 1) % ITEMS;
        }

        pthread_mutex_unlock(&mutex);
        sem_post(&full_slots);

        nanosleep(&delay, NULL);
    }

    return NULL;
}
```



```c
void *consumer_function(void *arg)
{
    int consumer_id = *((int *)arg);
    free(arg);  // Eliberam memoria alocata pentru ID

    while (1) {
        sem_wait(&full_slots);
        pthread_mutex_lock(&mutex);

        if (head != tail) {
            consume_item(buffer[head], consumer_id);
            head = (head + 1) % ITEMS;
        }

        pthread_mutex_unlock(&mutex);
        sem_post(&free_slots);
    }

    return NULL;
}

int main(int argc, char *argv[])
{
    pthread_t producer;
    pthread_t consumers[NUM_CONSUMERS];
    int i;

    // 250 msec
    delay.tv_sec = 0;
    delay.tv_nsec = 250000000;

    sem_init(&free_slots, 0, ITEMS - 1);
    sem_init(&full_slots, 0, 0);

    pthread_mutex_init(&mutex, NULL);

    // Cream un singur producator
    pthread_create(&producer, NULL, producer_function, NULL);

    // Cream mai multi consumatori
    for (i = 0; i < NUM_CONSUMERS; i++) {
        int *id = malloc(sizeof(int));  // Alocam memorie pentru ID
        *id = i + 1;  // ID-ul consumatorului (incepand de la 1)
        pthread_create(&consumers[i], NULL, consumer_function, id);
    }
```



```
biancapinghireac@vbox:~/SO/lab7/src$ ./semprodcons
producer produced 103
consumer 1 consuming 103
producer produced 198
consumer 3 consuming 198
producer produced 105
consumer 2 consuming 105
producer produced 115
consumer 1 consuming 115
producer produced 81
consumer 3 consuming 81
producer produced 255
consumer 2 consuming 255
producer produced 74
consumer 1 consuming 74
producer produced 236
consumer 3 consuming 236
producer produced 41
consumer 2 consuming 41
producer produced 205
consumer 1 consuming 205
producer produced 186
consumer 3 consuming 186
producer produced 171
consumer 2 consuming 171
producer produced 242
consumer 1 consuming 242
producer produced 251
consumer 3 consuming 251
producer produced 227
consumer 2 consuming 227
```