Sisteme de operare
Tema 7

Exercițiul 1

Rulați toate programele prezentate. Asigurați-vă că le-ați înțeles funcționarea. Folosiți-vă de pagina 3 de manual (comanda `man`). În caz că nu găsiți paginile de manual, puteți să le căutați și la `https://man7.org/linux/man-pages/index.html`

```
[user@fedora destination]$ make
gcc -o hello hello.c
gcc -o prodcons prodcons.c
gcc -o semprodcons semprodcons.c
```

Am compilat exemplele cu comanda *make*.

```
[user@fedora destination]$ ./hello
world Hello [user@fedora destination]$
```

Am folosit programul hello.

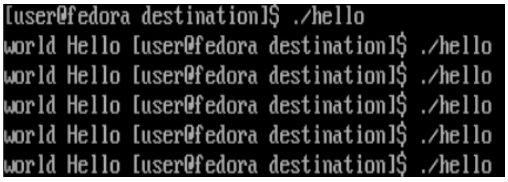
```
producing 202
consuming 202
producing 78
consuming 78
producing 232
consuming 232
producing 152
consuming 152
producing 50
consuming 50
producing 56
consuming 56
producing 224
consuming 224
producing 121
consuming 121
producing 77
consuming 77
producing 61
consuming 61
producing 52
consuming 52
producing 188
consuming 188
producing 95
consuming 95
producing 78
consuming 78
producing 119
consuming 119
producing 250
consuming 250
producing 203
consuming 203
producing 108
consuming 108
```

Am folosit programul prodcons – primul exemplu.

Am folosit programul prodcons – al doilea exemplu.

Exercițiul 2

Rulați programul `hello` de mai multe ori. Ce nedeterminare prezintă? Folosiți o variabilă mutex sau un semafor ca să ordonați pornirea celor două thread-uri, corectând problema afișării.



Am rulat programul hello de mai multe ori.

Nedeterminarea pe care o prezintă este output-ul care poate fi “Hello world” sau “world Hello”, acest lucru depinzând de ordinea în care sunt executate thread-urile.

```c
#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]=='w')
        {
                sem_wait(&sem);
        }
        printf("%s ",message);
        if(message[0]=='H')
        {
                sem_post(&sem);
        }
}

int main(int argc, char *argv[])
{
        pthread_t thread1, thread2;
        char *message1 = "Hello";
        char *message2 = "world";
        sem_init(&sem,0,0);

        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);
        printf("\n");
        exit(0);
}
```

Programul modificat.

```
[user@desktop-5p6viv2 destination]$ make hello
gcc -o hello hello.c
[user@desktop-5p6viv2 destination]$ ./hello
Hello world
[user@desktop-5p6viv2 destination]$ ./hello
Hello world
[user@desktop-5p6viv2 destination]$ ./hello
Hello world
[user@desktop-5p6viv2 destination]$ ./hello
Hello world
[user@desktop-5p6viv2 destination]$ ./hello
Hello world
```

Am rulat programul după modificare.

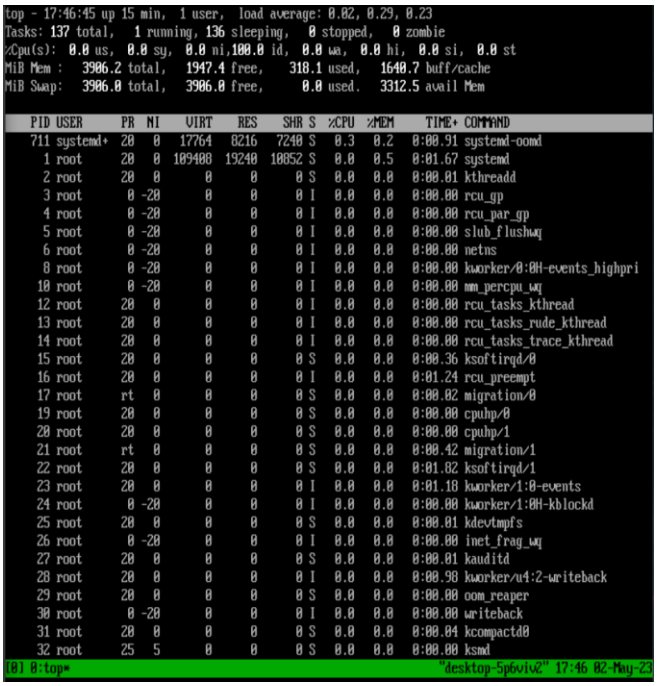
Exercițiul 3

Observați încărcarea CPU-ului provocată de execuția programului `prodcons`, folosind comanda `top`. Modificați programul adăugând o variabilă de condiție astfel ca să eliminați busy waiting-ul care provoacă această încărcare.

```
[user@desktop-5p6viv2 ~]$ tmux new-session -d 'top'
```

```
[user@desktop-5p6viv2 ~]$ tmux split-window -v './prodcons'
```

```
[user@desktop-5p6viv2 ~]$ tmux attach
```



Verificarea încărcării CPU-ului înaintea modificării.

Am observant că încărcarea CPU-ului provocată de execuția programului prodcons este mare.

```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;
pthread_cond_t cond;
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);
		while((tail+1)%ITEMS==head)
		{
			pthread_cond_wait(&cond,&mutex);
		}
		buffer[tail] = produce_item();
		tail = (tail + 1) % ITEMS;
		pthread_cond_signal(&cond);
		pthread_mutex_unlock(&mutex);

		nanosleep(&delay, NULL);
	}
}

void consumer_function(void)
{
	while (1)
	{
		pthread_mutex_lock(&mutex);
		while(head==tail)
		{
			pthread_cond_wait(&cond,&mutex);
		}
		consume_item(buffer[head]);
		head = (head + 1) % ITEMS;
		pthread_cond_signal(&cond);
		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_cond_init(&cond,NULL);
	pthread_create(&producer, NULL, (void *)&producer_function, NULL);

	consumer_function();
}
```

Am modificat programul în așa fel încât să reducă încărcarea CPU-ului și am eliminat busy waiting-ul. Am modificat în funcțiile producer_function și consumer_function astfel încât să folosească funcția thread_cond_wait pentru a aștepta un semnal din celălalt thread înaintea accesării buffer-ului.

```
top - 18:07:50 up 37 min,  1 user,  load average: 0.00, 0.00, 0.03
Tasks: 133 total,   1 running, 132 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   3906.2 total,   1924.2 free,    338.8 used,   1643.2 buff/cache
MiB Swap:   3906.0 total,   3906.0 free,      0.0 used.   3291.8 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
      1 root      20   0  109408  19240  10852 S   0.0   0.5   0:01.69 systemd
      2 root      20   0       0      0      0 S   0.0   0.0   0:00.01 kthreadd
      3 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_gp
      4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_par_gp
      5 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 slub_flushwq
      6 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 netns
      8 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/0:0H-events_highpri
     10 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 mm_percpu_wq
     12 root      20   0       0      0      0 I   0.0   0.0   0:00.00 rcu_tasks_kthread
     13 root      20   0       0      0      0 I   0.0   0.0   0:00.00 rcu_tasks_rude_kthread
     14 root      20   0       0      0      0 I   0.0   0.0   0:00.00 rcu_tasks_trace_kthread
     15 root      20   0       0      0      0 S   0.0   0.0   0:00.37 ksoftirqd/0
     16 root      20   0       0      0      0 I   0.0   0.0   0:01.55 rcu_preempt
     17 root      rt   0       0      0      0 S   0.0   0.0   0:00.04 migration/0
     19 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/0
     20 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/1
     21 root      rt   0       0      0      0 S   0.0   0.0   0:00.44 migration/1
     22 root      20   0       0      0      0 S   0.0   0.0   0:01.83 ksoftirqd/1
     24 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/1:0H-kblockd
     25 root      20   0       0      0      0 S   0.0   0.0   0:00.01 kdevtmpfs
     26 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 inet_frag_wq
     27 root      20   0       0      0      0 S   0.0   0.0   0:00.01 kauditd
     29 root      20   0       0      0      0 S   0.0   0.0   0:00.00 oom_reaper
     30 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 writeback
     31 root      20   0       0      0      0 S   0.0   0.0   0:00.18 kcompactd0
     32 root      25   5       0      0      0 S   0.0   0.0   0:00.00 ksmd
     33 root      39  19       0      0      0 S   0.0   0.0   0:00.00 khugepaged
     34 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 cryptd
     35 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kintegrityd
[0] 0:top*                                              "desktop-5p6vjv2" 18:07 02-May-23
```

Verificarea încărcării CPU-ului după modificare.

Exercițiul 4

Modificați programul `prodcons` astfel încât să aveți mai mulți producători și un singur consumator.

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

#define         ITEMS   10
#define         NUM_PRODUCERS 3

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

pthread_mutex_t mutex;
pthread_cond_t cond;
struct timespec delay;
```

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

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

        pthread_mutex_init(&mutex, NULL);
        pthread_cond_init(&cond,NULL);

        for(int i=0;i<NUM_PRODUCERS;i++)
        {
           pthread_create(&producers[i], NULL, (void *)&producer_function, NULL);
        }
        consumer_function();
}
```

Am modificat programul de la execițiul 3 astfel încât să aibă mai mulți producători printr-un loop și un array.

```
producing 13
producing 233
consuming 191
consuming 13
consuming 233
producing 140
producing 126
producing 78
consuming 140
consuming 126
consuming 78
producing 50
producing 189
producing 249
consuming 50
consuming 189
consuming 249
producing 124
producing 140
producing 106
consuming 124
consuming 140
consuming 106
producing 199
producing 91
producing 164
consuming 199
consuming 91
consuming 164
producing 60
producing 2
producing 244
consuming 60
consuming 2
consuming 244
```

Programul executat.

Exercițiul 5

Modificați programul `semprodcons` astfel încât să aveți mai mulți consumatori și un singur producător.

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

#define         ITEMS   10
#define         NUM_CONSUMERS 3

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);
}
```

```c
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);
        }
}
```

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

        // 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);
        for(int i=0;i<NUM_CONSUMERS;i++)
        {
           pthread_create(&consumers[i], NULL, (void *)&consumer_function, NULL);
        }
        producer_function();
}
```

Am modificat programul semprodcons astfel încât să aibă mai mulți consumatori și un singur producător. Pentru mai mulți consumatori am folosit un loop și array.

```
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
```

Programul executat.