```
nguyensang@K-virtual-machine:~/Desktop/OS2020/3$ ./1.exe
Producer-2 (pid 3735) produces A
                                        Consumer 0 (pid 3736) consumes A
Producer-0 (pid 3733) produces B
                                        Consumer 1 (pid 3737) consumes B
Producer-1 (pid 3734) produces C
                                        Consumer 2 (pid 3738) consumes C
Producer-2 (pid 3735) produces D
                                        Consumer 0 (pid 3736) consumes D
Producer-0 (pid 3733) produces E
                                        Consumer 1 (pid 3737) consumes E
Producer-1 (pid 3734) produces F
                                        Consumer 2 (pid 3738) consumes F
Producer-2 (pid 3735) produces G
                                        Consumer 0 (pid 3736) consumes G
Producer-0 (pid 3733) produces H
                                        Consumer 1 (pid 3737) consumes H
Producer-1 (pid 3734) produces I
                                        Consumer 2 (pid 3738) consumes I
Producer-2 (pid 3735) produces J
                                        Consumer 0 (pid 3736) consumes J
Producer-0 (pid 3733) produces K
                                        Consumer 1 (pid 3737) consumes K
Producer-1 (pid 3734) produces L
                                        Consumer 2 (pid 3738) consumes L
Producer-2 (pid 3735) produces M
Producer-0 (pid 3733) produces N
Producer-1 (pid 3734) produces O
                                        Consumer 0 (pid 3736) consumes M
                                        Consumer 1 (pid 3737) consumes N
                                        Consumer 2 (pid 3738) consumes O
Producer-2 (pid 3735) produces P
```

4 массива структур

```
1.
2.  #define BIN_SEM   0
3.  #define BUFFER_EMPTY 1
4.  #define BUFFER_FULL  2
5.
6.  truct sembuf producerP[2] = { {BUFFER_EMPTY, DEC, SEM_UNDO},
7.                                {BIN_SEM, DEC, SEM_UNDO}};
8.
9.  struct sembuf producerV[2] = { {BIN_SEM, INC, SEM_UNDO},
10.                               {BUFFER_FULL, INC, SEM_UNDO}};
11.
12. struct sembuf consumerP[2] = { {BUFFER_FULL,  DEC, SEM_UNDO},
13.                               {BIN_SEM,   DEC, SEM_UNDO}};
14.
15. struct sembuf consumerV[2] = { {BIN_SEM, INC, SEM_UNDO},
16.                               {BUFFER_EMPTY, INC, SEM_UNDO}};
17.
18.
19. void producer(const int id)
```

```
{
    while(1)
    {
        sleep(randint(0, PRODUCERS_DELAY));

        if (semop(semId, producerP, 2) == -1)
        {
            perror("semop");
            exit(1);
        }

        /* положить в буфер */
        int symbol = 'A' + *shm_producer_count % ('Z' - 'A');
        *(shm + *shm_producer_count) = symbol;
        printf("Producer-%d (pid %d) produces %c\n", id, getpid(), symbol);
        (*shm_producer_count)++;

        if (semop(semId, producerV, 2) == -1)
        {
            perror("semop");
            exit(1);
        }
    }
}

void consumer(const int id)
{
    while(1)
    {
        sleep(randint(0, CONSUMERS_DELAY));

        if (semop(semId, consumerP, 2) == -1)
        {
            perror("semop");
            exit(1);
        }
        /* взять из буфера */
        printf("\t\t\t\t\tConsumer %d (pid %d) consumes %c\n", id, getpid(), *(shm + *s
hm_consumer_count));
        (*shm_consumer_count)++;

        if (semop(semId, consumerV, 2) == -1)
        {
            perror("semop");
            exit(1);
        }
    }
}
```

Полная программа:

```
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/sem.h>
#include <sys/shm.h>
```

```c
#include <time.h>
#include <unistd.h>
#include <sys/wait.h>

#define BIN_SEM   0
#define BUFFER_EMPTY 1
#define BUFFER_FULL  2

#define DEC -1
#define INC 1

#define PRODUCERS_COUNT 3
#define CONSUMERS_COUNT 3

#define PRODUCERS_DELAY 3
#define CONSUMERS_DELAY 2

struct sembuf producerP[2] = { {BUFFER_EMPTY, DEC, SEM_UNDO},
                               {BIN_SEM, DEC, SEM_UNDO}};

struct sembuf producerV[2] = { {BIN_SEM, INC, SEM_UNDO},
                               {BUFFER_FULL, INC, SEM_UNDO}};

struct sembuf consumerP[2] = { {BUFFER_FULL,  DEC, SEM_UNDO},
                               {BIN_SEM,   DEC, SEM_UNDO}};

struct sembuf consumerV[2] = { {BIN_SEM, INC, SEM_UNDO},
                               {BUFFER_EMPTY, INC, SEM_UNDO}};


#define N 26 /*'Z' - 'A' */

#define PERMS S_IRWXU | S_IRWXG | S_IRWXO

int semId = -1;
int shmId = -1;

int *shm = NULL;
int *shm_producer_count = NULL;
int *shm_consumer_count = NULL;
int *shm_symbol_now = NULL;

int randint(int a, int b)
{
    return a + rand() % (b - a + 1);
}

int semrel(int semId)
{
    return semctl(semId, 0, IPC_RMID, 0);
}

int shmrel(int semId)
{
    return shmctl(shmId, IPC_RMID, NULL);
}

void forkChildren(const int n, void (*func)(const int))
{
    for (int i = 0; i < n; ++i)
    {
```

```
68.         const pid_t pid = fork();
69.         if (pid == -1)
70.         {
71.             perror("fork");
72.             exit(1);
73.         }
74.         else if (pid == 0)
75.         {
76.             if (func)
77.                 func(i);
78.             exit(1);
79.         }
80.     }
81. }
82.
83. void waitChildren(const int n)
84. {
85.     for (int i = 0; i < n; ++i)
86.     {
87.         int status;
88.         const pid_t child_pid = wait(&status);
89.         if (child_pid == -1)
90.         {
91.             perror("wait");
92.             exit(1);
93.         }
94.         if (WIFEXITED(status))
95.             printf("Process %d returns %d\n", child_pid, WEXITSTATUS(status));
96.         else if (WIFSIGNALED(status))
97.             printf("Process %d terminated with signal %d\n", child_pid, WTERMSIG(status
    ));
98.         else if (WIFSTOPPED(status))
99.             printf("Process %d stopped due signal %d\n", child_pid, WSTOPSIG(status));

100.            }
101.        }
102.
103.        void producer(const int id)
104.        {
105.            while(1)
106.            {
107.                sleep(randint(0, PRODUCERS_DELAY));
108.
109.                if (semop(semId, producerP, 2) == -1)
110.                {
111.                    perror("semop");
112.                    exit(1);
113.                }
114.
115.                /* положить в буфер */
116.                int symbol = 'A' + *shm_producer_count % ('Z' - 'A');
117.                *(shm + *shm_producer_count) = symbol;
118.                printf("Producer-%d (pid %d) produces %c\n", id, getpid(), symbol);
119.                (*shm_producer_count)++;
120.
121.                if (semop(semId, producerV, 2) == -1)
122.                {
123.                    perror("semop");
124.                    exit(1);
125.                }
126.            }
```

```
127.        }
128.
129.        void consumer(const int id)
130.        {
131.            while(1)
132.            {
133.                sleep(randint(0, CONSUMERS_DELAY));
134.
135.                if (semop(semId, consumerP, 2) == -1)
136.                {
137.                    perror("semop");
138.                    exit(1);
139.                }
140.                /* взять из буфера */
141.                printf("\t\t\t\t\tConsumer %d (pid %d) consumes %c\n", id, getpid(), *(s
    hm + *shm_consumer_count));
142.                (*shm_consumer_count)++;
143.
144.                if (semop(semId, consumerV, 2) == -1)
145.                {
146.                    perror("semop");
147.                    exit(1);
148.                }
149.            }
150.        }
151.
152.        void initSemaphore()
153.        {
154.            /* два считающих семафора + один бинарный */
155.            semId = semget(IPC_PRIVATE, 3, IPC_CREAT | PERMS);
156.
157.            if (semId == -1)
158.            {
159.                perror("semget");
160.                exit(1);
161.            }
162.            /*количество заполненных ячеек равно 0*/
163.            /*Все ячейки буфера изначально пусты */
164.            if (semctl(semId, BIN_SEM,   SETVAL, 1) == -1 ||
165.                semctl(semId, BUFFER_EMPTY, SETVAL, N) == -1 ||
166.                semctl(semId, BUFFER_FULL,  SETVAL, 0) == -1)
167.            {
168.                perror("semctl");
169.                exit(1);
170.            }
171.        }
172.
173.        void createSharedMemory()
174.        {
175.            // (N + 3) * sizeof(int) - kích thước
176.            //IPC_PRIVATE - tạo seg mới
177.            shmId = shmget(IPC_PRIVATE, (N + 2) * sizeof(int), IPC_CREAT | PERMS);
178.            if (shmId == -1)
179.            {
180.                perror("shmget");
181.                exit(1);
182.            }
183.            shm = shmat(shmId, 0, 0);
184.            if (shm == (void *) -1)
185.            {
186.                perror("shmat");
```

```
187.                exit(1);
188.            }
189.            shm_producer_count = shm;
190.            shm_consumer_count = shm + 1;
191.            *shm_producer_count = 0;
192.            *shm_consumer_count = 0;
193.            shm = shm + 2;
194.        }
195.
196.        int main()
197.        {
198.            initSemaphore();
199.            createSharedMemory();
200.
201.            forkChildren(PRODUCERS_COUNT, producer);
202.            forkChildren(CONSUMERS_COUNT, consumer);
203.
204.            waitChildren(PRODUCERS_COUNT + CONSUMERS_COUNT);
205.
206.            shmrel(semId);
207.            semrel(semId);
208.        }
```