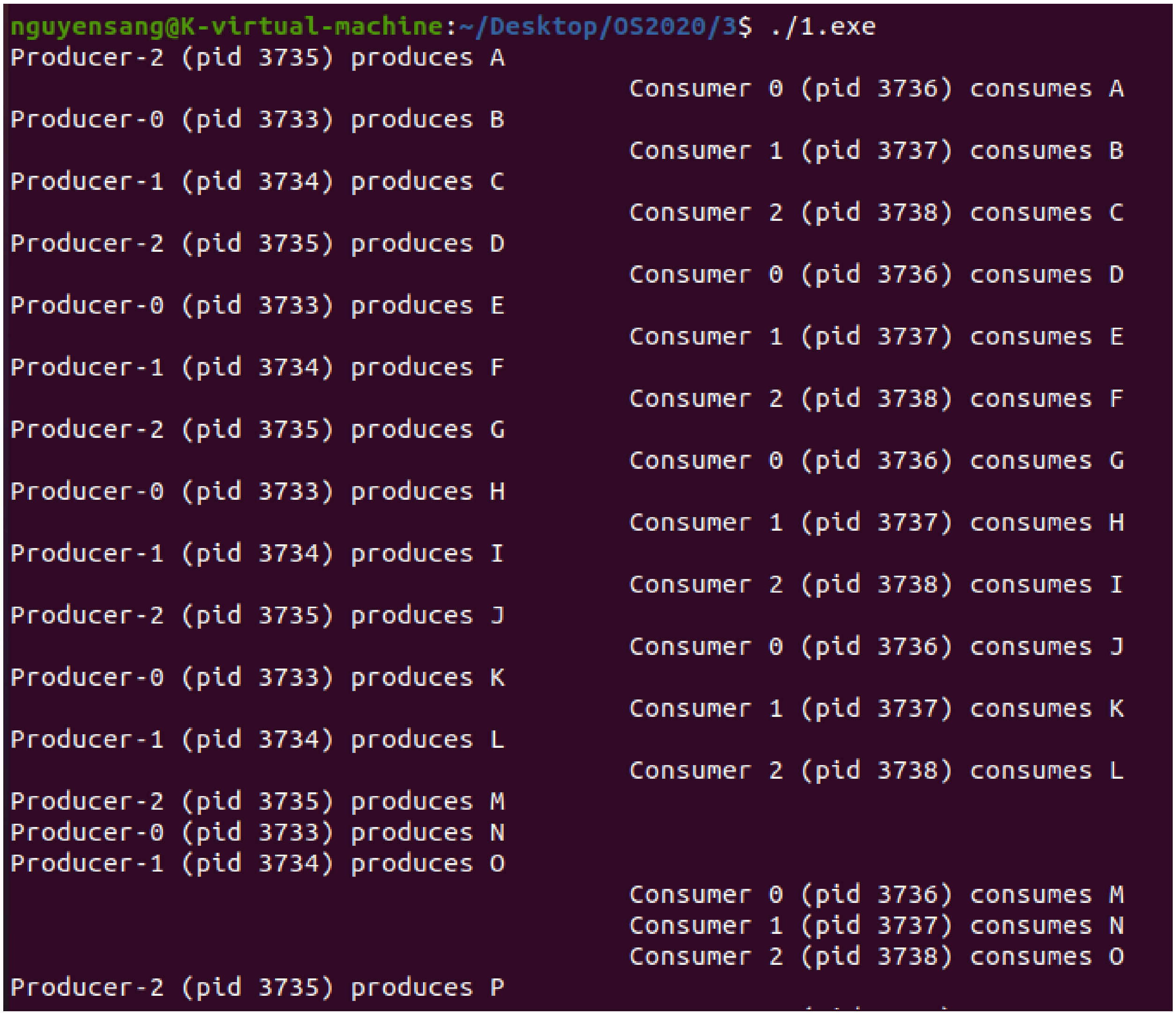

```
nguyensang@K-virtual-machine:~/Desktop/OS2020/3$ ./1.exe
Producer-2 (pid 3735) produces A
                                    Consumer 0 (pid 3736) consumes A
Producer-0 (pid 3733) produces B
                                    Consumer 1 (pid 3737) consumes B
Producer-1 (pid 3734) produces C
                                    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

```

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

truct 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}};


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 + *shm_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>
```

```
#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)
    {
        const pid_t pid = fork();
        if (pid == -1)
        {
            perror("fork");
            exit(1);
        }
        else if (pid == 0)
        {
            if (func)
                func(i);
            exit(1);
        }
    }
}

void waitChildren(const int n)
{
    for (int i = 0; i < n; ++i)
    {
        int status;
        const pid_t child_pid = wait(&status);
        if (child_pid == -1)
        {
            perror("wait");
            exit(1);
        }
        if (WIFEXITED(status))
            printf("Process %d returns %d\n", child_pid, WEXITSTATUS(status));
        else if (WIFSIGNALED(status))
            printf("Process %d terminated with signal %d\n", child_pid, WTERMSIG(status));
        else if (WIFSTOPPED(status))
            printf("Process %d stopped due signal %d\n", child_pid, WSTOPSIG(status));
    }
}

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);
        }
    }
```

```c
}

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 + *shm_consumer_count));
        (*shm_consumer_count)++;

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

void initSemaphore()
{
    /* два считающих семафора + один бинарный */
    semId = semget(IPC_PRIVATE, 3, IPC_CREAT | PERMS);

    if (semId == -1)
    {
        perror("semget");
        exit(1);
    }
    /*количество заполненных ячеек равно 0*/
    /*Все ячейки буфера изначально пусты */
    if (semctl(semId, BIN_SEM,   SETVAL, 1) == -1 ||
        semctl(semId, BUFFER_EMPTY, SETVAL, N) == -1 ||
        semctl(semId, BUFFER_FULL,  SETVAL, 0) == -1)
    {
        perror("semctl");
        exit(1);
    }
}

void createSharedMemory()
{
    // (N + 3) * sizeof(int) - kích thước
    //IPC_PRIVATE - tạo seg mới
    shmId = shmget(IPC_PRIVATE, (N + 2) * sizeof(int), IPC_CREAT | PERMS);
    if (shmId == -1)
    {
        perror("shmget");
        exit(1);
    }
    shm = shmat(shmId, 0, 0);
    if (shm == (void *) -1)
    {
        perror("shmat");
```

```
            exit(1);
        }
        shm_producer_count = shm;
        shm_consumer_count = shm + 1;
        *shm_producer_count = 0;
        *shm_consumer_count = 0;
        shm = shm + 2;
    }

    int main()
    {
        initSemaphore();
        createSharedMemory();

        forkChildren(PRODUCERS_COUNT, producer);
        forkChildren(CONSUMERS_COUNT, consumer);

        waitChildren(PRODUCERS_COUNT + CONSUMERS_COUNT);

        shmrel(semId);
        semrel(semId);
    }
```