System Programming

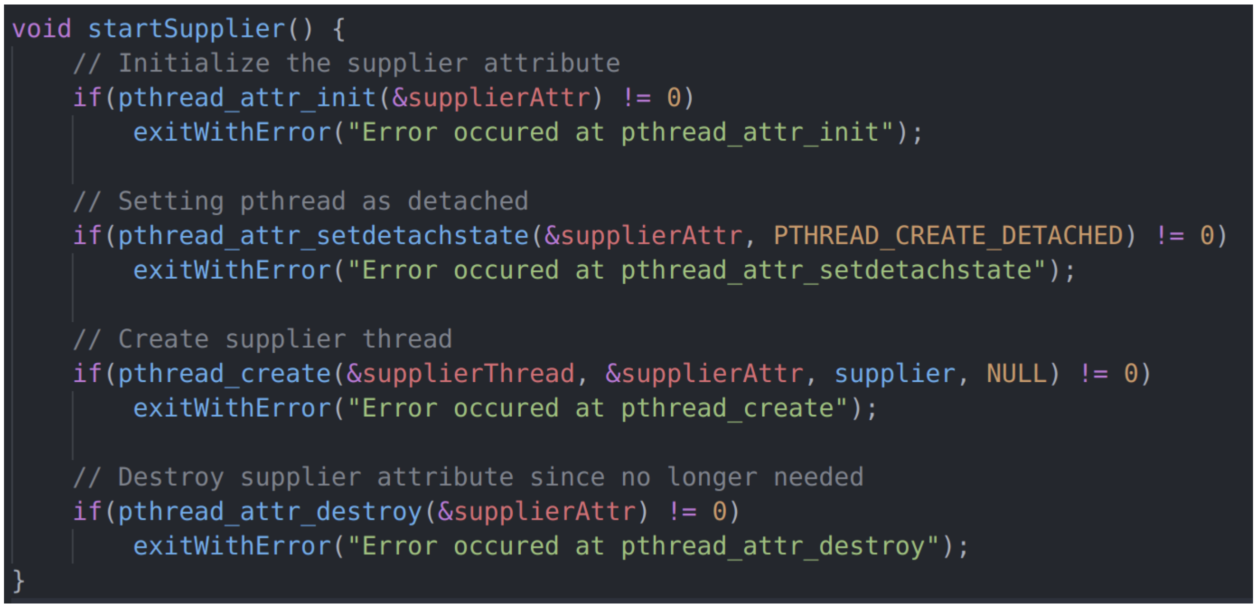
Homework 4 Report | 2021 - 2022

Ahmet Tuğkan Ayhan

1901042692

Creating Threads

Supplier Thread



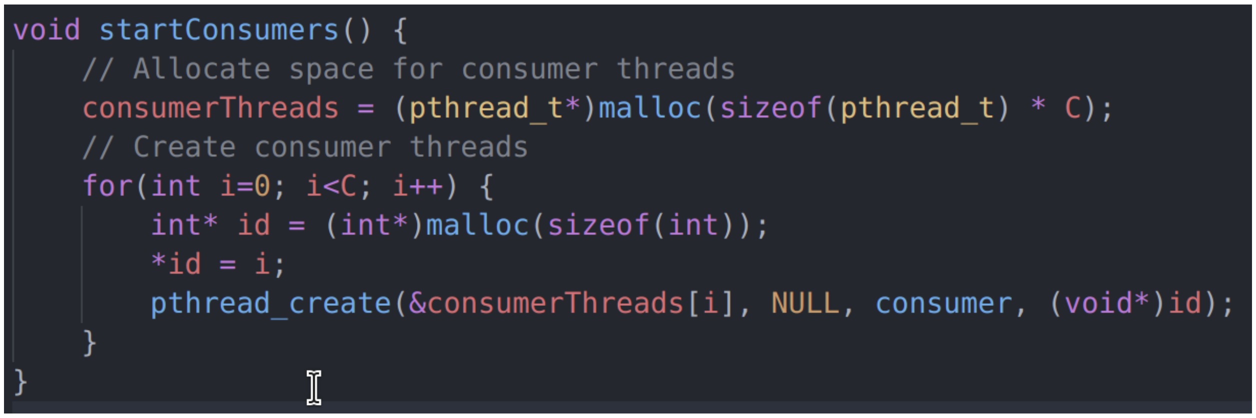
```c
void startSupplier() {
    // Initialize the supplier attribute
    if(pthread_attr_init(&supplierAttr) != 0)
        exitWithError("Error occured at pthread_attr_init");

    // Setting pthread as detached
    if(pthread_attr_setdetachstate(&supplierAttr, PTHREAD_CREATE_DETACHED) != 0)
        exitWithError("Error occured at pthread_attr_setdetachstate");

    // Create supplier thread
    if(pthread_create(&supplierThread, &supplierAttr, supplier, NULL) != 0)
        exitWithError("Error occured at pthread_create");

    // Destroy supplier attribute since no longer needed
    if(pthread_attr_destroy(&supplierAttr) != 0)
        exitWithError("Error occured at pthread_attr_destroy");
}
```

We need to pass through 4 steps to successfully create and detach the supplier thread:

1. First we initialize **pthread_attr_t** object using **pthread_attr_init.**
2. Second, we set state of the thread as **PTHREAD_CREATE_DETACHED.**
3. Third, we create the thread using **pthread_create.** In my design it doesn't need to take any parameter, so I assigned parameter *void as NULL.
4. Finally, we destroy the attribute object using **pthread_attr_destroy** because we no longer need it.

After these 4 steps, **supplier** function successfully created and detached. I'll explain what happens inside supplier function while I'm talking about system V semaphores.

Consumer Threads



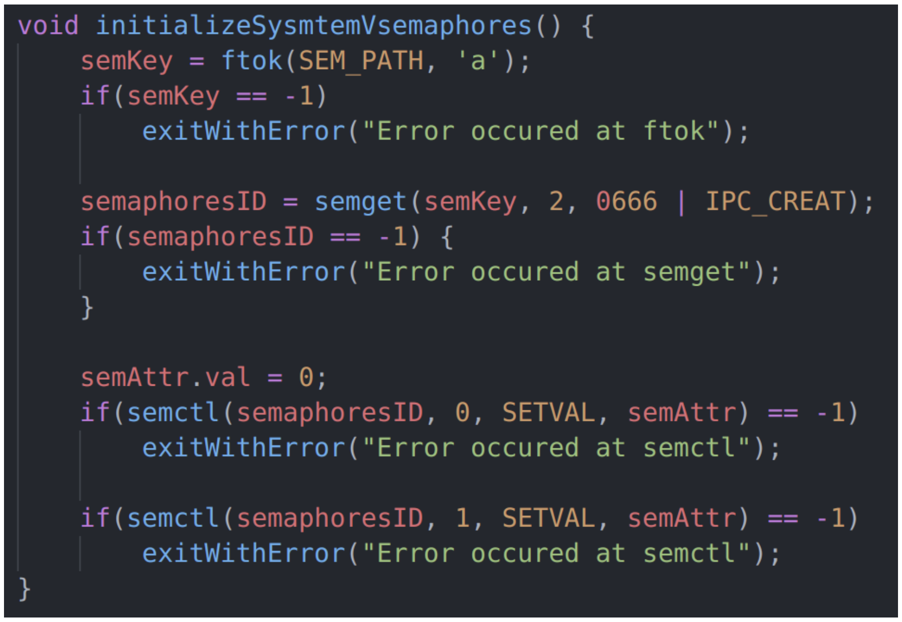
```c
void startConsumers() {
    // Allocate space for consumer threads
    consumerThreads = (pthread_t*)malloc(sizeof(pthread_t) * C);
    // Create consumer threads
    for(int i=0; i<C; i++) {
        int* id = (int*)malloc(sizeof(int));
        *id = i;
        pthread_create(&consumerThreads[i], NULL, consumer, (void*)id);
    }
}
```

Creating consumer threads is easier since they are not detached(so instead of setting attributes, we just simply give their attributes as NULL). But to create them, first we need to go through 4 steps:

1. First, since we have ‘C’ amount of threads, before doing anything we need to allocate space for these **pthread_t** objects(which named **consumerThreads**).
2. After allocating space for pthread_t objects, we define id values for each consumer thread. We can’t just simply send an integer value as parameter, because of that we allocate these int values in heap using malloc.
3. By deferencing the id values we assign their id values.
4. Finally, we create threads using **pthread_create** with with parameter id but no attributes.

Using System V Semaphores

This is the hardest part of the homework. Because the working mechanism of system V semaphores a bit different then what we are used to with POSIX semaphores. Before creating any threads, I initialized two of the system V semaphores using **initializeSysmtemVsemaphores()** function



```
void initializeSysmtemVsemaphores() {
    semKey = ftok(SEM_PATH, 'a');
    if(semKey == -1)
        exitWithError("Error occured at ftok");

    semaphoresID = semget(semKey, 2, 0666 | IPC_CREAT);
    if(semaphoresID == -1) {
        exitWithError("Error occured at semget");
    }

    semAttr.val = 0;
    if(semctl(semaphoresID, 0, SETVAL, semAttr) == -1)
        exitWithError("Error occured at semctl");

    if(semctl(semaphoresID, 1, SETVAL, semAttr) == -1)
        exitWithError("Error occured at semctl");
}
```

To initalize system V semaphores, we need to do 4 things:

1. First, we get the semaphore key **semKey** using **ftok()** function. In this function first parameter is SEM_PATH and I defined this value inside my main.h file which is equal to "/tmp". Thus, this system V semaphores' key value is defined in /tmp folder.
2. Then using **semKey** value, we get id of the semaphores using **semget()** function. While doing our semaphore operations using **semctl** and **semop** we will use this id.
3. Third, we set **semAttr** value as 0. Which is our initial value for system V semaphores. **semAttr** is an **semun struct** object which is defined in our course book and lecture slides.
4. Finally, we set two of our semaphores initial value using **semctl**. To use the value inside **semAttr** object, we give third parameter as **SETVAL**. Second parameter shows which semaphore we are assigning these values to.

System V Semaphores inside Supplier Thread

After understanding how System V semaphores works, it is easy to use them. But understanding the concept takes some time. Inside supplier thread I first defined a **sembuf struct object** named **pushSemaphore** and everytime I read a '1' from the file, I set;

> **pushSemaphore.sem_num** to 0
>
> **pushSemaphore.sem_op** to 1
>
> **pushSemaphore.sem_flg** to 0

Then I used **semop()** function to send signal meaning **sem_num** 0 can wake up now. And eveytime I read a '2' from the file, different from than **pushSemaphore.sem_num** to 0, I assigned it to 1 which indicates to second semaphore. Then I called **semop** function again. This operations executed until there is no item left inside the input file.

Also I done the signal handling at the start of every loop for supplier function.

System V Semaphores inside Consumer Threads

Consumer part of the system V semaphores a lot easier than supplier, The things we need to do:

1. Defining a sembuf struct object (I named it doubleSem) with two elements array (it is written as "struct sembuf doubleSem[2];").
2. Then, setting attributes of the doubleSem object as:

```
doubleSem[0].sem_num = 0;
doubleSem[0].sem_op = -1;
doubleSem[0].sem_flg = 0;

doubleSem[1].sem_num = 1;
doubleSem[1].sem_op = -1;
doubleSem[1].sem_flg = 0;
```

3. Finally, call semop using semaphore ids and doubleSem.

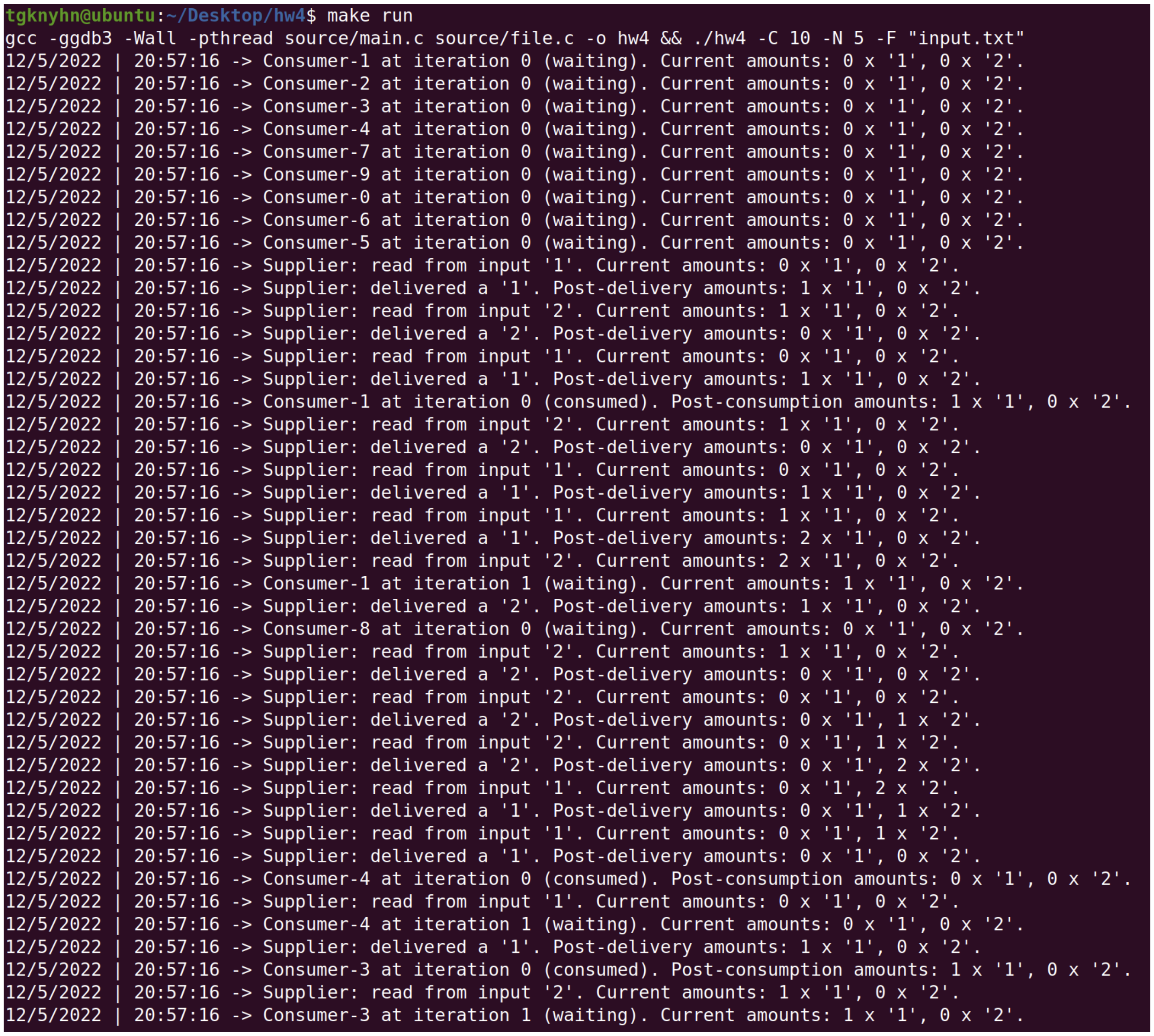
That is it. After doing these steps and making sure everything works correct(by doing error checks) then you are good to go.

Example Execution

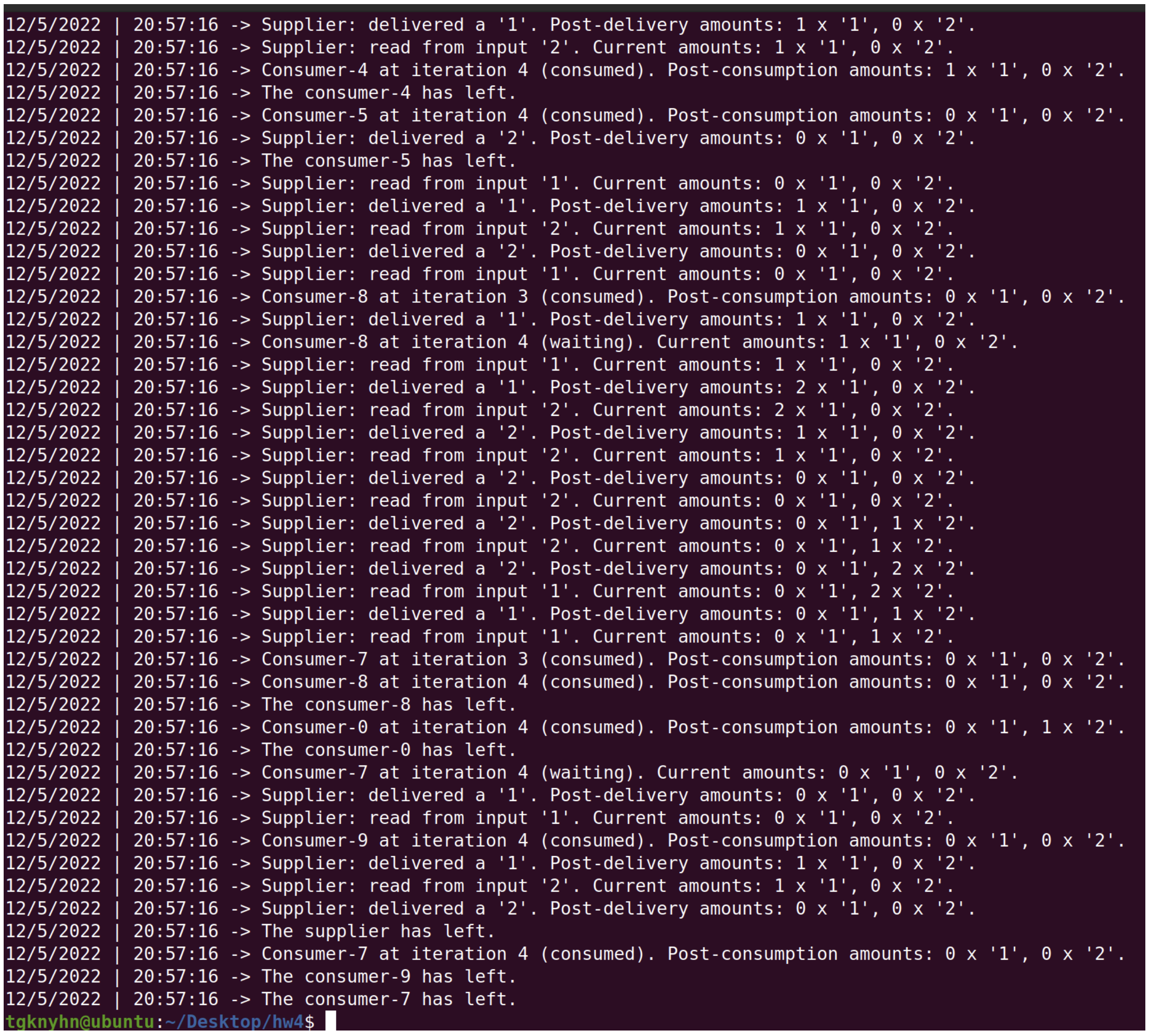
It is possible to run program by calling “make run” but calling only “make” only compiles the program. “make run” call adds the given line to Makefile and calls it:

./hw4 -C 10 -N 5 -F "input.txt"

The console result I will show here is the result of calling the line above. Since result screen is too long, I will show the beginning and end of the result (2 screenshots).



```
tgknyhn@ubuntu:~/Desktop/hw4$ make run
gcc -ggdb3 -Wall -pthread source/main.c source/file.c -o hw4 && ./hw4 -C 10 -N 5 -F "input.txt"
12/5/2022 | 20:57:16 -> Consumer-1 at iteration 0 (waiting). Current amounts: 0 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Consumer-2 at iteration 0 (waiting). Current amounts: 0 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Consumer-3 at iteration 0 (waiting). Current amounts: 0 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Consumer-4 at iteration 0 (waiting). Current amounts: 0 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Consumer-7 at iteration 0 (waiting). Current amounts: 0 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Consumer-9 at iteration 0 (waiting). Current amounts: 0 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Consumer-0 at iteration 0 (waiting). Current amounts: 0 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Consumer-6 at iteration 0 (waiting). Current amounts: 0 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Consumer-5 at iteration 0 (waiting). Current amounts: 0 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Supplier: read from input '1'. Current amounts: 0 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Supplier: delivered a '1'. Post-delivery amounts: 1 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Supplier: read from input '2'. Current amounts: 1 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Supplier: delivered a '2'. Post-delivery amounts: 0 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Supplier: read from input '1'. Current amounts: 0 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Supplier: delivered a '1'. Post-delivery amounts: 1 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Consumer-1 at iteration 0 (consumed). Post-consumption amounts: 1 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Supplier: read from input '2'. Current amounts: 1 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Supplier: delivered a '2'. Post-delivery amounts: 0 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Supplier: read from input '1'. Current amounts: 0 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Supplier: delivered a '1'. Post-delivery amounts: 1 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Supplier: read from input '1'. Current amounts: 1 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Supplier: delivered a '1'. Post-delivery amounts: 2 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Supplier: read from input '2'. Current amounts: 2 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Consumer-1 at iteration 1 (waiting). Current amounts: 1 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Supplier: delivered a '2'. Post-delivery amounts: 1 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Consumer-8 at iteration 0 (waiting). Current amounts: 0 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Supplier: read from input '2'. Current amounts: 1 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Supplier: delivered a '2'. Post-delivery amounts: 0 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Supplier: read from input '2'. Current amounts: 0 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Supplier: delivered a '2'. Post-delivery amounts: 0 x '1', 1 x '2'.
12/5/2022 | 20:57:16 -> Supplier: read from input '2'. Current amounts: 0 x '1', 1 x '2'.
12/5/2022 | 20:57:16 -> Supplier: delivered a '2'. Post-delivery amounts: 0 x '1', 2 x '2'.
12/5/2022 | 20:57:16 -> Supplier: read from input '1'. Current amounts: 0 x '1', 2 x '2'.
12/5/2022 | 20:57:16 -> Supplier: delivered a '1'. Post-delivery amounts: 0 x '1', 1 x '2'.
12/5/2022 | 20:57:16 -> Supplier: read from input '1'. Current amounts: 0 x '1', 1 x '2'.
12/5/2022 | 20:57:16 -> Supplier: delivered a '1'. Post-delivery amounts: 0 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Consumer-4 at iteration 0 (consumed). Post-consumption amounts: 0 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Supplier: read from input '1'. Current amounts: 0 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Consumer-4 at iteration 1 (waiting). Current amounts: 0 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Supplier: delivered a '1'. Post-delivery amounts: 1 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Consumer-3 at iteration 0 (consumed). Post-consumption amounts: 1 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Supplier: read from input '2'. Current amounts: 1 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Consumer-3 at iteration 1 (waiting). Current amounts: 1 x '1', 0 x '2'.
```



```
12/5/2022 | 20:57:16 -> Supplier: delivered a '1'. Post-delivery amounts: 1 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Supplier: read from input '2'. Current amounts: 1 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Consumer-4 at iteration 4 (consumed). Post-consumption amounts: 1 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> The consumer-4 has left.
12/5/2022 | 20:57:16 -> Consumer-5 at iteration 4 (consumed). Post-consumption amounts: 0 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Supplier: delivered a '2'. Post-delivery amounts: 0 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> The consumer-5 has left.
12/5/2022 | 20:57:16 -> Supplier: read from input '1'. Current amounts: 0 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Supplier: delivered a '1'. Post-delivery amounts: 1 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Supplier: read from input '2'. Current amounts: 1 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Supplier: delivered a '2'. Post-delivery amounts: 0 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Supplier: read from input '1'. Current amounts: 0 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Consumer-8 at iteration 3 (consumed). Post-consumption amounts: 0 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Supplier: delivered a '1'. Post-delivery amounts: 1 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Consumer-8 at iteration 4 (waiting). Current amounts: 1 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Supplier: read from input '1'. Current amounts: 1 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Supplier: delivered a '1'. Post-delivery amounts: 2 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Supplier: read from input '2'. Current amounts: 2 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Supplier: delivered a '2'. Post-delivery amounts: 1 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Supplier: read from input '2'. Current amounts: 1 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Supplier: delivered a '2'. Post-delivery amounts: 0 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Supplier: read from input '2'. Current amounts: 0 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Supplier: delivered a '2'. Post-delivery amounts: 0 x '1', 1 x '2'.
12/5/2022 | 20:57:16 -> Supplier: read from input '2'. Current amounts: 0 x '1', 1 x '2'.
12/5/2022 | 20:57:16 -> Supplier: delivered a '2'. Post-delivery amounts: 0 x '1', 2 x '2'.
12/5/2022 | 20:57:16 -> Supplier: read from input '1'. Current amounts: 0 x '1', 2 x '2'.
12/5/2022 | 20:57:16 -> Supplier: delivered a '1'. Post-delivery amounts: 0 x '1', 1 x '2'.
12/5/2022 | 20:57:16 -> Supplier: read from input '1'. Current amounts: 0 x '1', 1 x '2'.
12/5/2022 | 20:57:16 -> Consumer-7 at iteration 3 (consumed). Post-consumption amounts: 0 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Consumer-8 at iteration 4 (consumed). Post-consumption amounts: 0 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> The consumer-8 has left.
12/5/2022 | 20:57:16 -> Consumer-0 at iteration 4 (consumed). Post-consumption amounts: 0 x '1', 1 x '2'.
12/5/2022 | 20:57:16 -> The consumer-0 has left.
12/5/2022 | 20:57:16 -> Consumer-7 at iteration 4 (waiting). Current amounts: 0 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Supplier: delivered a '1'. Post-delivery amounts: 0 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Supplier: read from input '1'. Current amounts: 0 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Consumer-9 at iteration 4 (consumed). Post-consumption amounts: 0 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Supplier: delivered a '1'. Post-delivery amounts: 1 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Supplier: read from input '2'. Current amounts: 1 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> Supplier: delivered a '2'. Post-delivery amounts: 0 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> The supplier has left.
12/5/2022 | 20:57:16 -> Consumer-7 at iteration 4 (consumed). Post-consumption amounts: 0 x '1', 0 x '2'.
12/5/2022 | 20:57:16 -> The consumer-9 has left.
12/5/2022 | 20:57:16 -> The consumer-7 has left.
tgknyhn@ubuntu:~/Desktop/hw4$
```

- END -