Zadanie

- Magazyn o pojemności k sztuk towaru,
- n konsumentów, każdy z nich jednorazowo odbiera partię towaru w liczbie sztuk ~ U(a, b),
- m producentów, każdy z nich jednorazowo produkuje partię towaru w liczbie sztuk ~ U(c, d).

Producenci przekazują towar konsumentom za pośrednictwem magazynu. Producenci i konsumenci obsługują tylko całe partie towaru.

Producenci produkują tylko całe partie towaru. Jeśli mejsc w magazynie jest mniej niż liczba wyprodukowanych towarów, proces producenta zostaje wstrzymany. Analogicznie dla konsumenta.

Magazyn reprezentowany jako plik. Log producentów/konsumentów zapisywany do oddzielnych plików.

Koncepcja

Do rozwiązania zadania potrzebujemy dwa semafory `fillCount` i `emptyCount`, które będą zliczać liczbę dostępnych towarów oraz liczbę wolnych miejsc w magazynie. Aby uniemożliwić innym procesom z taką samą ochroną jednoczesne wykonywanie i dostęp do tych samych lokalizacji pamięci użyjemy semafora binarnego `mutex`.

```
sem_init(0, 1);   /* mutex */
sem_init(1, 0);   /* fillCount */
sem_init(2, k);   /* emptyCount */

producer():
    while (true):
        wylosuj liczbę sztuk towaru n;
        zapisz do pliku "(time) Trying to take n items from store";
        jeśli liczba wolnych miejsc >= n
        then:
            while (n>0):
                sem_down(2);    /* emptyCount - 1 */
                sem_down(0);    /* zablokuj mutex */
                pushItem();
                sem_up(0);      /* odblokuj mutex */
                sem_up(1);      /* fillCount + 1 */
                n--;
            zapisz do pliku "(time) Took n items from store";
        else:
            zapisz do pliku "(time) Failed taking n items from store";



consumer():
    while (true):
        wylosuj liczbę sztuk towaru n;
        zapisz do pliku "(time) Trying to insert n items into store";
        jeśli liczba sztuk towaru >= n
        then:
            while (n>0):
```

```
                sem_down(1);    /* fillCount - 1 */
                sem_down(0);    /* zablokuj mutex */
                popItem();
                sem_up(0);      /* odblokuj mutex */
                sem_up(2);      /* emptyCount + 1 */
                n--;

            zapisz do pliku "(time) Inserted n items into store";
        else:
            zapisz do pliku "(time) Failed inserting n items into store";
```

Powyższy kod działa w następujący sposób: `fillCount` jest zwiększane, a `emptyCount` zmniejszane, gdy nowy element jest umieszczany w magazynie. Jeśli producent próbuje zmniejszyć wartość zmiennej `emptyCount`, gdy jej wartość wynosi zero, proces producenta zostaje wsrzymany. Następnym razem, gdy towar zostanie odebrany przez konsumenta, liczba `emptyCount` zostanie zwiększona i producent się obudzi. Konsument działa analogicznie.

Funkcje `pushItem()` i `popItem()` będą odpowiednio dodawać/usuwać jeden towar do/z magazynu. Ponieważ magazyn będzie reprezentowany jako plik z liczbą dostępnych towarów, dodawanie/usuwanie towarów będzie realizowane poprzez dodawanie/odejmowanie liczby od liczby wszystkich dostępnych w magazynie towarów. Log tych działań wraz z czasem ich stworzenia będzie zapisywany do odpowiedniego pliku.

Przykładowe działanie programu dla pustego magazynu o pojemności k = 5, zainicjowanych dwóch procesach producenta i jednego procesu konsumenta:

Plik producenta1:

```
(00:00:00) Trying to insert 2 items into store.
(00:00:10) Inserted 2 items into store.
(00:00:11) Trying to insert 4 items into store.
(00:00:40) Inserted 4 items into store.
...
```

Plik producenta2:

```
(00:00:00) Trying to insert 2 items into store.
(00:00:20) Inserted 2 items into store.
(00:00:21) Trying to insert 4 items into store.
(00:00:50) Failed inserting 4 items into store.
...
```

Plik konsumenta:

```
(00:00:00) Trying to take 3 items from store.
(00:00:30) Took 3 items from store.
(00:00:31) Trying take 6 items from store.
(00:01:00) Failed taking 6 items from store.
...
```