Opis rozwiązania zadania 3

Mikołaj Olejnik

Zadanie rozwiązałem implementując program w C. Stworzyłem strukturę Buffer, z dwoma semaforami empty i full, oraz jeden semafor binarny mutex.

```
struct Buffer {
    char* item_name;
    sem_t empty;
    sem_t full;
    pthread_mutex_t mutex;
};
```

Następnie stworzyłem strukturę która reprezentuje pierogarnię. Składa się ona z 4 bufferów, każdy z nich odpowiadający innemu składnikowi.

```
struct Pierogarnia {
    Buffer ciasto;
    Buffer mieso;
    Buffer kapusta;
    Buffer ser;
};
```

W pliku main.c znajduje się główna implementacja mojego rozwiązania.

Na początku tworzę producentów, każdy z nich jest osobnym wątkiem. Argumentami funkcji produce jest index producenta oraz buffer do którego będzie on produkował składniki.

```
for (int i = 0; i < PRODUCERS; i++) {
    args.buffer = buffers[i];
    args.index = indexes[i];
    args_list_producers[i] = args;
    pthread_create(&producers[i], NULL, produce, (void *)&args_list_producers[i]);
    sleep(1);
}
```

W tablicy buffers dla każdego producenta umieściłem odpowiadający mu buffer. Na początku upewniłem się, że każdy ze składników jest produkowany przynajmniej przez jednego producenta. Reszta producentów dostała losowy buffer do uzupełniania.

```
void fill_buffers(Pierogarnia *pierogarnia, Buffer* buffers[])
{
    // we must have all the ingredients for a pierog
    buffers[0] = &pierogarnia->ciasto;
    buffers[1] = &pierogarnia->mieso;
    buffers[2] = &pierogarnia->ser;
    buffers[3] = &pierogarnia->kapusta;

    // if we have more availabe producers, then we want more "ciasto" produced
    if (PRODUCERS > 4) {
        buffers[4] = &pierogarnia->ciasto;
    }

    // any more available producers will be random
    for (int i=5; i<PRODUCERS; i++) {
        buffers[i] = get_random_buffer(pierogarnia);
    }
}
```

W funkcji produce w nieskończonej pętli producent tworzy składnik.

```
void *produce(void *arguments)
{
    arg_struct *args = arguments;
    Buffer *buffer = args->buffer;
    int index = args->index;

    int value;
    while (1) {
        sleep(1);
        sem_wait(&buffer->empty);
        pthread_mutex_lock(&buffer->mutex);

        sem_getvalue(&buffer->full, &value);
        printf("Producer %d: Created \"%s\" (%d/%d)\n",
            index, buffer->item_name, value+1, BUFFER_SIZE);

        pthread_mutex_unlock(&buffer->mutex);
        sem_post(&buffer->full);
    }
}
```

Następnie tworzę konsumentów, którzy będą robili pierogi. Argumentami funkcji consume jest index konsumenta oraz odpowiedni buffer z nadzieniem (ser, kapusta lub mięso).

```
for (int i = 0; i < CONSUMERS; i++) {
    args.buffer = buffers_nadzienie[i];
    args.index = indexes[i];
    args_list_producers[i] = args;
    pthread_create(&consumers[i], NULL, consume, (void *)&args_list_producers[i]);
    sleep(1);
}
```

W funkcji consume tworzone są pierogi. Każdy z konsumentów ma dostęp do bufora na ciasto i bufora z losowym składnikiem. Jeśli zarówno ciasto jak i nadzienie jest dostępne, to konsument tworzy pieroga.

```
void *consume(void *arguments)
{
    struct arg_struct *args = arguments;
    Buffer *buffer_nadzienie = args->buffer;
    Buffer *buffer_ciasto = args->buffer_ciasto;
    int index = args->index;

    int value_ciasto, value_nadzienie;
    while (1) {
        sleep(1);
        sem_wait(&buffer_nadzienie->full);
        sem_wait(&buffer_ciasto->full);
        pthread_mutex_lock(&buffer_nadzienie->mutex);
        pthread_mutex_lock(&buffer_ciasto->mutex);

        sem_getvalue(&buffer_nadzienie->full, &value_nadzienie);
        sem_getvalue(&buffer_ciasto->full, &value_ciasto);
        printf("Consumer %d: Made pierog with \"%s\" (%d/%d) (Ciasto %d/%d)\n",
            index, buffer_nadzienie->item_name, value_nadzienie, BUFFER_SIZE, value_ciasto, BUFFER_SIZE);

        pthread_mutex_unlock(&buffer_nadzienie->mutex);
        pthread_mutex_unlock(&buffer_ciasto->mutex);
        sem_post(&buffer_nadzienie->empty);
        sem_post(&buffer_ciasto->empty);
    }
}
```