SOI LAB 3 – Bufor komunikacyjny

Implementacja bufora

```
struct Buffer_s {
    Data _data[BUFFER_SIZE];
    int _top;
    int sem_group;
};
typedef struct Buffer_s Buffer;
```

Bufor który zaimplementowałem jest strukturą zawierającą tablicę danych typu Data, przechowującą komunikaty (składniki pierogów) w formie stosu. Pole _top jest szczytem stosu. Pole sem_group jest identyfikatorem grupy semaforów odpowiedzialnych za wykluczanie. Bufor można obsługiwać funkcjami buffer_init, buffer_insert oraz buffer_pop. Stała BUFFER_SIZE określa maksymalny rozmiar bufora. W kodzie ustawiona jest na 10.

Implementacja semaforów

Na samym początku programu tworzę grupy 3 semaforów dla każdego z buforów składników funkcją *semget:*

```
// creating semaphore groups
int dough_sem_id = semget( key: IPC_PRIVATE,  nsems: 3,  semflg: 0666);
if (dough_sem_id < 0) {
    perror( s: "Error creating semaphore group");
    exit( status: 1);
}
int meat_sem_id = semget( key: IPC_PRIVATE,  nsems: 3,  semflg: 0666);
if (meat_sem_id < 0) {
    perror( s: "Error creating semaphore group");
    exit( status: 1);
}
int cheese_sem_id = semget( key: IPC_PRIVATE,  nsems: 3,  semflg: 0666);
if (cheese_sem_id < 0) {
    perror( s: "Error creating semaphore group");
    exit( status: 1);
}
int cabbage_sem_id = semget( key: IPC_PRIVATE,  nsems: 3,  semflg: 0666);
if (cabbage_sem_id < 0) {
    perror( s: "Error creating semaphore group");
    exit( status: 1);
}
```

Deklaruje ona semafory w systemie i zwraca ID grupy, z której później będzie można korzystać w programie. Indeksy poszczególnych semaforów w grupie są zdefiniowane na początku pliku main.c:

```
#define S_FULL          0
#define S_EMPTY         1
#define S_MUTEX         2
```

Przy inicjalizacji bufora ustawiam początkowe wartości każdego z semaforów:

```
void buffer_init(Buffer *buf, int sem_group_id) {
    buf->_top = 0;
    buf->sem_group = sem_group_id;

    if(semctl( semid: sem_group_id,  semnum: S_EMPTY,  cmd: SETVAL, BUFFER_SIZE - 1) < 0 ||
    semctl( semid: sem_group_id,  semnum: S_FULL,  cmd: SETVAL, 0) < 0 ||
    semctl( semid: sem_group_id,  semnum: S_MUTEX,  cmd: SETVAL, 1) < 0)
    {
        perror( s: "Error initializing semaphores!");
        exit( status: 1);
    }
}
```

Następnie konsumenci i producenci korzystają z funkcji up i down do uzyskania blokowania. Funkcje te zostały zdefiniowane w następujący sposób:

```
void down(int sem_id, ushort which_one) {
    struct sembuf sem_op;
    sem_op.sem_num = which_one;
    sem_op.sem_op = -1;
    sem_op.sem_flg = 0;

    if (semop( semid: sem_id,  sops: &sem_op,  nsops: 1) < 0) {
        perror( s: "Error in locking semaphore");
        exit( status: 1);
    }
}

void up(int sem_id, ushort which_one) {
    struct sembuf sem_op;
    sem_op.sem_num = which_one;
    sem_op.sem_op = 1;
    sem_op.sem_flg = 0;

    if (semop( semid: sem_id,  sops: &sem_op,  nsops: 1) < 0) {
        perror( s: "Error in locking semaphore");
        exit( status: 1);
    }
}
```

Argument sem_id to id grupy semaforów do której się odnosimy, natomiast which_one określa na którym semaforze w grupie operacja powinna zostać wykonana.

Implementacja producenta i konsumenta

```
void producer(Buffer *buf, char *ingredient_type, int time_taken, int id) {
    Data data;
    int i = 0;
    while (1) {
        data = produce_item(&i, ingredient: ingredient_type, amount: 10.0f, time_taken, producer_id: id);
        printf( format: "%28s_%d [P] Wait for space in buffer...\n", ingredient_type, id);
        down( sem_id: buf->sem_group, which_one: S_EMPTY);
        down( sem_id: buf->sem_group, which_one: S_MUTEX);
        buffer_insert(buf, data);
        printf( format: "%28s_%d [P] Added to buffer!\n", ingredient_type, id);
        up( sem_id: buf->sem_group, which_one: S_MUTEX);
        up( sem_id: buf->sem_group, which_one: S_FULL);
    }
}

void consumer(Buffer *dough_buffer, Buffer *filling_buffer, char *pierogi_type, int time_taken, int id) {
    Data dough, filling;
    while (1) {
        printf( format: "%28s_%d [C] Waiting for ingredients...\n", pierogi_type, id);
        down( sem_id: dough_buffer->sem_group, which_one: S_FULL);
        down( sem_id: filling_buffer->sem_group, which_one: S_FULL);

        down( sem_id: dough_buffer->sem_group, which_one: S_MUTEX);
        down( sem_id: filling_buffer->sem_group, which_one: S_MUTEX);

        dough = buffer_pop( buf: dough_buffer);
        filling = buffer_pop( buf: filling_buffer);

        up( sem_id: filling_buffer->sem_group, which_one: S_MUTEX);
        up( sem_id: dough_buffer->sem_group, which_one: S_MUTEX);

        up( sem_id: filling_buffer->sem_group, which_one: S_EMPTY);
        up( sem_id: dough_buffer->sem_group, which_one: S_EMPTY);

        consume_item(&dough, &filling, pierog_type: pierogi_type, time_taken, consumer_id: id);
    }
```

Producenci przyjmują jako argument jeden bufor do którego odkładają wyprodukowane przez siebie składniki danego typu (ingredient_type), swoje ID unikalne w grupie producentów zajmujących się tym samym składnikiem oraz to ile czasu zajmuje produkcja. Na produkowany jest składnik danego typu, o danej masie i danym numerze. W wypadku producentów należy zadbać o to by nie można było zapisać do pełnego bufora. Jest to zapewniane semaforem EMPTY. Semafor mutex tworzy sekcję krytyczną w której do bufora wstawiana jest przygotowana wcześniej struktura Data.

W przypadku konsumentów przyjmowanymi argumentami są 2 bufory – jeden od ciasta a drugi od składnika który jest potrzebny do ulepienia pieroga danego typu. Konsumenci także przyjmują argumenty czasu konsumpcji oraz ich ID. W tym wypadku musimy zadbać o to by konsument się zablokował gdy którykolwiek z potrzebnych buforów jest pusty, stąd wywołanie down na semaforach FULL. Następnie semafory mutex obydwu buforów zapewniają sekcję krytyczną w której ściągamy z buforów ciasto i składnik. Po wyjściu z sekcji krytycznej oraz wykonaniu funkcji UP na semaforach empty obydwu buforów konsumujemy obydwa składniki (zlepiamy z nich pieróg).

Program testowy

Program można skompilować przy użyciu polecenia cmake:

```
$cmake CmakeLists.txt
$make
```

Uruchomienie programu:

```
$./lab3 <argumenty wywołania>
```

Gdzie argumentami wywołania to po kolei:

- *rand_fraction → losowość czasu oczekiwania [0;1]*
- *dough_prod_time → czas potrzebny na produkcję ciasta [ms]*
- *meat_prod_time → czas potrzebny na produkcję mięsa [ms]*
- *cheese_prod_time → czas potrzebny na produkcję twarogu [ms]*
- *cabb_prod_time → czas potrzebny na produkcję kapusty [ms]*
- *meat_cons_time → czas potrzebny na lepienie pieroga z mięsem [ms]*
- *cheese_cons_time → czas potrzebny na lepienie pieroga z twarogiem [ms]*
- *cabb_cons_time → czas potrzebny na lepienie pieroga z kapustą [ms]*
- *n_dough_prod → ilość producentów ciasta*
- *n_meat_prod → ilość producentów mięsa*
- *n_cheese_prod → ilość producentów twarogu*
- *n_cabb_prod → ilość producentów kapusty*
- *n_meat_cons → ilość konsumentów lepiących pierogi z mięsem*
- *n_cheese_cons → ilość konsumentów lepiących pierogi z twarogiem*
- *n_cabb_cons → ilość konsumentów lepiących pierogi z kapustą*

W wypadku pominięcia któregoś z argumentów użyte będą ich podstawowe wartości zdefiniowane w kodzie źródłowym.

W funkcji main znajduje się alokacja buforów dla ciasta, mięsa, sera oraz kapusty. Następnie tworzone są grupy semaforów, po czym bufory są inicjalizowane. Po wykonaniu tych czynności wywoływane są funkcje create_n_producers oraz create_n_consumers. Jako argumenty przyjmują one bufory których dotyczy dana grupa konsumentów/producentów, to jakim produktem się zajmują oraz ile takich konsumentów/producentów ma być. Każdy z konsumentów/producentów wypisuje na standardowe wyjście informacje o tym czym się w danym momencie zajmuje wedle poniższego schematu:

```
pierog z miesem_0 [C] Combining (10.0g ciasto (B_4_5)) and (10.0g mieso (B_3_3)) to make (pierog z miesem)
```

Od lewej do prawej są to po kolei:

- produkt którym zajmuje się producent/konsument

- po znaku ‘_’ - numer producenta/konsumenta danego typu

- w kwadratowym nawiasie litera C przypomina, że jest to konsument a litera P, że jest to producent

- cała reszta wiersza jest opisem akcji która właśnie się zaczęła/została ukończona

W tym wypadku jest to komunikat konsumenta który właśnie zaczął tworzyć pieróg z mięsem (zaczął konsumpcję, ma id=0).