Λειτουργικά Συστήματα

3η Άσκηση

Κωνσταντίνος Τσάμπρας
up1083865

➔ Επίλυση του προβλήματος με νήματα:

Αρχικά ορίζουμε το μέγεθος του buffer, το πλήθος παραγωγών και καταναλωτών (N, P, C) και μια συνάρτηση που τυπώνει τον buffer για να φαίνεται η κατάσταση του κάθε στιγμή.

consumer_producer_threads.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

#define N 10
#define P 5
#define C 5

// Declare semaphores
sem_t mutex, full, empty;
int buffer[N]={0};
int buffer_index = 0;

void print_buffer() {
    printf("Buffer: ");
    for(int i=0; i<N; i++) {
        printf("%d ", buffer[i]);
    }
    printf("\n");
}
```

Στη συνέχεια ορίζουμε τις συναρτήσεις του παραγωγού και του καταναλωτή

```c
void * producer(void * arg) {
    srand((int)(arg));
    printf("Producer created\n");
    while(1) {
        sem_wait(&empty);
        sem_wait(&mutex);

        // produce an item
        buffer[buffer_index]=rand()%9 +1;
        printf("++Produced: %d\n", buffer[buffer_index]);
        buffer_index++;
        print_buffer();

        sem_post(&mutex);
        sem_post(&full);
    }
    printf("Producer exiting\n");
}

void * consumer(void * arg) {
    printf("Consumer created\n");
    while(1) {
        sem_wait(&full);
        sem_wait(&mutex);

        // remove item from buffer
        buffer_index--;
        printf("--Consumed: %d\n", buffer[buffer_index]);
        buffer[buffer_index]=0;
        print_buffer();


        sem_post(&mutex);
        sem_post(&empty);


        // consume item
    }
    printf("Consumer exiting\n");
}
```

Στην συνάρτηση παραγωγού έχουμε ορίσει να παράγεται ένας τυχαίος αριθμός από το 1 έως το 9 και να τοποθετείται στον buffer, ενώ στον καταναλωτή αυτός ο τελευταίος αριθμός αφαιρείται και αντικαθίσταται με 0.

Τέλος στην main αρχικοποιούμε τους σημαφόρους, δημιουργούμε τα threads και τα κάνουμε join. Το αποτέλεσμα της εκτέλεσης φαίνεται παρακάτω:

Αρχικά ο buffer αρχίζει να γεμίζει από producers (ίσως επειδή δημιουργήθηκαν πρώτοι) και από ένα σημείο και μετά τόσο producers όσο και consumers εκτελούνται με αποτέλεσμα να αυξομειώνεται συνεχώς το μέγεθος του buffer και να προσθαφαιρούνται νένα στοιχεία:

```
Buffer: 0 0 0 0 0 0 0 0 0 0
Producer created
Producer created
Producer created
Producer created
Producer created
Consumer created
Consumer created
Consumer created
Consumer created
Consumer created
++Produced: 3
Buffer: 3 0 0 0 0 0 0 0 0 0
++Produced: 6
Buffer: 3 6 0 0 0 0 0 0 0 0
++Produced: 1
Buffer: 3 6 1 0 0 0 0 0 0 0
++Produced: 4
Buffer: 3 6 1 4 0 0 0 0 0 0
++Produced: 7
Buffer: 3 6 1 4 7 0 0 0 0 0
++Produced: 7
Buffer: 3 6 1 4 7 7 0 0 0 0
--Consumed: 7
Buffer: 3 6 1 4 7 0 0 0 0 0
++Produced: 9
Buffer: 3 6 1 4 7 9 0 0 0 0
--Consumed: 9
Buffer: 3 6 1 4 7 0 0 0 0 0
++Produced: 3
Buffer: 3 6 1 4 7 3 0 0 0 0
--Consumed: 3
Buffer: 3 6 1 4 7 0 0 0 0 0
++Produced: 6
Buffer: 3 6 1 4 7 6 0 0 0 0
--Consumed: 6
Buffer: 3 6 1 4 7 0 0 0 0 0
++Produced: 9
Buffer: 3 6 1 4 7 9 0 0 0 0
--Consumed: 9
Buffer: 3 6 1 4 7 0 0 0 0 0
--Consumed: 7
Buffer: 3 6 1 4 0 0 0 0 0 0
++Produced: 8
Buffer: 3 6 1 4 8 0 0 0 0 0
--Consumed: 8
Buffer: 3 6 1 4 0 0 0 0 0 0
```

➔ Επίλυση με διεργασίες:

(Αρχείο consumer_producer_processes.c)

Αρχικά ορίζουμε το μέγεθος του buffer και το πλήθος των καταναλωτών και των παραγωγών, καθώς και την συνάρτηση για εκτύπωση του buffer.

```c
#define N 10
#define P 2
#define C 2



const char *mutex = "mutex";
const char *full = "full";
const char *empty = "empty";

sem_t *mutex_id;
sem_t *full_id;
sem_t *empty_id;


void print_buffer(int * buffer) {
    printf("Buffer: ");
    for(int i=0; i<N; i++) {
        printf("%d ", buffer[i]);
    }
    printf("\n");
}
```

Στην συνέχεια έχουμε την συνάρτηση που εκτελείται από τους παραγωγούς. Αντί για while(True) έχει τοποθετηθεί ένα for 10000 επαναλήψεων για να μπορούμε να παρατηρήσουμε την συμπεριφορά του προγράμματος.

```c
void producer(int * buffer, int * buffer_index_ptr){
    int i;
    sem_t *mutex_id = sem_open(mutex, O_CREAT, 0600, 1);
    sem_t *full_id = sem_open(full, O_CREAT, 0600, 0);
    sem_t *empty_id = sem_open(empty, O_CREAT, 0600, N);
//    printf("producer\n");
    sleep(1);
    for(i=0; i<10000; i++){
        sem_wait(empty_id);
        sem_wait(mutex_id);
        buffer[*buffer_index_ptr]=rand()%9+1;
        printf("++Produced: %d\n", *buffer_index_ptr);
        (*buffer_index_ptr)++;
        print_buffer(buffer);


        sem_post(mutex_id);
        sem_post(full_id);


    }
}
```

Ομοίως για την συνάρτηση που θα εκτελείται από τους καταναλωτές.

```
void consumer(int * buffer, int * buffer_index_ptr)
{
    int i;
    sem_t *mutex_id = sem_open(mutex, O_CREAT, 0600, 1);
    sem_t *full_id = sem_open(full, O_CREAT, 0600, 0);
    sem_t *empty_id = sem_open(empty, O_CREAT, 0600, N);

    for(i=0; i<10000; i++){

        sem_wait(full_id);
        sem_wait(mutex_id);

        (*buffer_index_ptr)--;
        printf("--Consumed: %d\n", buffer[*buffer_index_ptr]);
        buffer[*buffer_index_ptr]=0;
        print_buffer(buffer);

        sem_post(mutex_id);
        sem_post(empty_id);

    }
```

Και στις δύο περιπτώσεις δίνουμε ως ορίσματα στις συναρτήσεις δυο pointers για δύο «κομμάτια μνήμης» που μοιράζονται οι δύο διεργασίες και δημιουργούνται στην main, ένα για το buffer array και ένα για το index αυτού.

Στην main

```
int main(int argc, char *argv[])
{
    pid_t pid;
    int i;
    int buffer_id;
    int buffer_index_id;


    buffer_id = shmget(IPC_PRIVATE, N*sizeof(int), SHM_R | SHM_W);
    buffer_index_id = shmget(IPC_PRIVATE, sizeof(int *), SHM_R | SHM_W);
    for(i=0; i<P; i++){

        pid = fork();
        if (pid < 0){
            perror("fork");
            exit(EXIT_FAILURE);
        }

        if (!pid){
            producer((int *)shmat(buffer_id, NULL, 0), (int *)shmat(buffer_index_id, NULL, 0));
            return 0;
        }
    }
```

Στην main δημιουργούμε τα δύο κομμάτια μνήμης και αρχικά δημιουργούμε P παραγωγούς στους οποίους περνάμε τα προαναφερθέντα ορίσματα.

Τέλος εκτελούμε το for και για τους καταναλωτές και κάνουμε unlink τους σημαφόρους:

```
    for(i=0; i<C; i++){
        pid = fork();
        if (pid < 0){
            perror("fork");
            exit(EXIT_FAILURE);
        }

        if (!pid){
            //consumer((int *)shmat(buffer_id, NULL, 0));
            consumer((int *)shmat(buffer_id, NULL, 0), (int *)shmat(buffer_index_id, NULL, 0));
            return 0;
        }
    }

    sleep(5);
    printf("unlinking\n");
    sem_unlink(mutex);
    sem_unlink(full);
    sem_unlink(empty);

    return 0;
}
```

Αποτέλεσμα του παραπάνω κώδικα είναι το παρακάτω, βλέπουμε ότι όσο εκτελείται η διεργασία του παραγωγού προστίθενται στον buffer ενώ όταν γίνεται εναλλαγή σε διεργασία καταναλωτή αρχίζουν να αφαιρούνται στοιχεία από τον buffer.