ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

ΦΡΟΝΤΙΣΤΗΡΙΟ – ΑΣΚΗΣΗ 3

ΜΠΑΣΑΓΙΑΝΝΗ ΓΕΩΡΓΙΑ

1084016

Για την εκπόνηση της άσκησης τα αρχεία με τους κώδικες στη c γράφτηκαν με χρήση VSCode και για την μεταγλώττιση και εκτέλεση τους χρησιμοποιήθηκε το WSL (μέσω terminal).

Το πρόβλημα του παραγωγού-καταναλωτή με νήματα

f3_threads.c | f3_processes.c

C: > Users > georg > OneDrive - University of Patras > Desktop > mathimata > 7o > OS > frontistiriakes > f3 > f3_threads.c

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

#define BUFFER_SIZE 5

//Buffer structure
typedef struct{
    int items[BUFFER_SIZE];
    int in; //index for insert
    int out; //index for remove
    pthread_mutex_t mutex; //mutual exclusion
    sem_t full; //counts full slots in buffer
    sem_t empty; //counts emplty slots in buffer
}Buffer;

Buffer buffer;

void *producer(void *rank);
int produce_item();
void *consumer(void *rank);
void insert_item(Buffer *buf, int item);
int remove_item(Buffer *buf);

int main(){
    //Initialize buffer and synchronization sem/mutex
    buffer.in=0;
    buffer.out=0;
    pthread_mutex_init(&buffer.mutex, NULL);
    sem_init(&buffer.full, 0, 0);
    sem_init(&buffer.empty, 0, BUFFER_SIZE);

    //create producer, consumer threads
    pthread_t t1_producer, t2_producer, t1_consumer, t2_consumer;
```

```c
    pthread_create(&t1_producer, NULL, producer, NULL);
    pthread_create(&t2_producer, NULL, producer, NULL);
    pthread_create(&t1_consumer, NULL, consumer, NULL);
    pthread_create(&t2_consumer, NULL, consumer, NULL);

    //join threads
    pthread_join(t1_producer, NULL);
    pthread_join(t2_producer, NULL);
    pthread_join(t1_consumer, NULL);
    pthread_join(t2_consumer, NULL);

    //clean up
    pthread_mutex_destroy(&buffer.mutex);
    sem_destroy(&buffer.full);
    sem_destroy(&buffer.empty);

    return 0;
}

//producer
void *producer(void* rank){
    for(int i=0; i<10; i++){

        //produce item
        int item=produce_item();

        //wait for empty slot
        sem_wait(&buffer.empty);

        //insert into buffer
        pthread_mutex_lock(&buffer.mutex);
        insert_item(&buffer, item);
        printf("produced --> item %d\n", item);
        pthread_mutex_unlock(&buffer.mutex);

        //signal for full slot
        sem_post(&buffer.full);

    }

    pthread_exit(NULL);
}

int produce_item(){
    int item;
    item = rand() % 100;
    return item;
}

//consumer
void *consumer(void* rank){
    for(int i=0; i<10; i++){

        //wait for full slot
        sem_wait(&buffer.full);

        //remove from buffer
        pthread_mutex_lock(&buffer.mutex);
        int item = remove_item(&buffer);
        printf("---------consumed --> item %d\n", item);
        pthread_mutex_unlock(&buffer.mutex);

        //signal for empty slot
        sem_post(&buffer.empty);
```

```
100
101         }
102
103         pthread_exit(NULL);
104     }
105
106     void insert_item(Buffer *buf, int item){
107         buf->items[buf->in] = item;
108         buf->in = (buf->in +1) % BUFFER_SIZE;
109     }
110
111     int remove_item(Buffer *buf){
112         int item = buf->items[buf->out];
113         buf->out = (buf->out +1) % BUFFER_SIZE;
114         return item;
115     }
```

Ορίζουμε την χωρητικότητα του buffer να είναι 5 (πράγμα που μπορούμε να αλλάξουμε με μία και μόνο αλλαγή στο #define. Έπειτα, δημιουργούμε την δομή του buffer με τα προϊόντα items που παράγονται και καταναλώνονται, 2 indexes που ορίζουν την θέση που εισάγεται και που αφαιρείται κάποιο προϊόν από το buffer, το mutex του αμοιβαίου αποκλεισμού και δύο σημαφόρους για τα γεμάτα και άδεια slots κάθε φορά.

Αρχικοποιούνται τα in και out στην αρχή του buffer (θέση 0), το mutex με NULL και full=0, empty=BUFFER_SIZE(=5). Δημιουργία των threads με την pthread_create 2 για παραγωγή και 2 για κατανάλωση. Επικοινωνία των threads με την pthread_join. Τέλος, καταστρέφει το mutex και τις σημαφόρους αφού πλέον δεν τις χρειάζεται.

Τα threads που κάνουν τη δουλειά του producer δημιουργούν ένα item μέσω της produce_item() που επιστρέφει έναν τυχαίο αριθμό μέσω της rand(), και κάνει wait_empty και αν υπάρχει άδειο slot μπαίνει σε mutex και εισάγει με την insert_item δεδομένο στο buffer. Επίσης μέσα στο mutex τυπώνει και το προϊόν που παράχθηκε (για τον κατάλληλο έλεγχο). Μετά, κάνει unlock το mutex και δίνει σήμα στην full.

Τα threads που κάνουν τη δουλειά του consumer κάνουν wait_full και αν υπάρχει γεμισμένο slot μπαίνει σε mutex και αφαιρεί ένα item από το buffer με την remove_item. Επίσης μέσα στο mutex τυπώνει και το προϊόν που καταναλώθηκε (για τον κατάλληλο έλεγχο). Μετά, κάνει unlock το mutex και δίνει σήμα στην empty.

Η insert_item βάζει το παραγόμενο item στην θέση in του buffer και αυξάνει την θέση in κατά 1 για να δείχνει στην επόμενη θέση. Η remove_item παίρνει το item της θέσης out του buffer και το επιστρέφει, αφού πρώτα αυξήσει το out κατά 1, στην επομένη θέση του buffer. Για να λειτουργήσει το +1 πρέπει να χρησιμοποιηθεί και το % modulo έτσι ώστε οι τιμές των in και out να είναι από 0-4 δηλαδή μέσα στα πλαίσια του BUFFER_SIZE.

Αποτελέσματα:

```
basagianni@LAPTOP-GECQDV8A:/mnt/c/Users/georg/OneDrive - University of Patras/Desktop/Mathimata/7o/OS/frontistiriakes/f3$ gcc f3_threads.c -o f3
_threads
basagianni@LAPTOP-GECQDV8A:/mnt/c/Users/georg/OneDrive - University of Patras/Desktop/Mathimata/7o/OS/frontistiriakes/f3$ ./f3_threads
produced --> item 83
produced --> item 77
produced --> item 15
---------consumed --> item 83
---------consumed --> item 77
produced --> item 86
produced --> item 93
produced --> item 35
produced --> item 86
---------consumed --> item 15
---------consumed --> item 86
produced --> item 92
produced --> item 21
---------consumed --> item 93
---------consumed --> item 35
produced --> item 49
---------consumed --> item 86
---------consumed --> item 92
produced --> item 62
produced --> item 90
---------consumed --> item 21
produced --> item 27
produced --> item 59
---------consumed --> item 49
---------consumed --> item 62
---------consumed --> item 90
produced --> item 63
produced --> item 40
produced --> item 26
---------consumed --> item 27
---------consumed --> item 59
---------consumed --> item 63

---------consumed --> item 40
---------consumed --> item 26
produced --> item 26
produced --> item 72
produced --> item 36
---------consumed --> item 26
---------consumed --> item 72
basagianni@LAPTOP-GECQDV8A:/mnt/c/Users/georg/OneDrive - University of Patras/Desktop/Mathimata/7o/OS/frontistiriakes/f3$ APTOP-GECQDV8A:/mnt/c/
```

Το πρόβλημα του παραγωγού-καταναλωτή με διεργασίες

f3_threads.c | f3_processes.c

C: > Users > georg > OneDrive - University of Patras > Desktop > mathimata > 7o > OS > frontistiriakes > f3 > f3_processes.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/mman.h>
#include <sys/wait.h>
#include <semaphore.h>

#define BUFFER_SIZE 5

//Buffer structure
typedef struct{
    int items[BUFFER_SIZE];
    int in; //index for insert
    int out; //index for remove
    sem_t mutex; //mutual exclusion
    sem_t full; //counts full slots in buffer
    sem_t empty; //counts emplty slots in buffer
}Buffer;

Buffer *buffer;

```

```c
void producer(int rank);
int produce_item();
void consumer(int rank);
void insert_item(Buffer *buf, int item);
int remove_item(Buffer *buf);

int main(){
    //create shared memory
    buffer = mmap(NULL, sizeof(Buffer), PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);

    //Initialize buffer and synchronization sem
    buffer->in=0;
    buffer->out=0;
    sem_init(&buffer->mutex, 1, 1);
    sem_init(&buffer->full, 1, 0);
    sem_init(&buffer->empty, 1, BUFFER_SIZE);

    //create producer, consumer processes
    pid_t ppid1, ppid2, cpid1, cpid2;

    if((ppid1=fork()) == 0){
        producer(1);
        exit(0);
    }
    else if((ppid2=fork()) == 0){
        producer(2);
        exit(0);
    }
    else if((cpid1=fork()) == 0){
        consumer(1);
        exit(0);
    }
    else if((cpid2=fork()) == 0){
        consumer(2);
        exit(0);
    }

    //wait all processes
    waitpid(ppid1, NULL, 0);
    waitpid(ppid2, NULL, 0);
    waitpid(cpid1, NULL, 0);
    waitpid(cpid2, NULL, 0);

    //clean up
    munmap(buffer, sizeof(Buffer));
    sem_destroy(&buffer->mutex);
    sem_destroy(&buffer->full);
    sem_destroy(&buffer->empty);

    return 0;
}

//producer
void producer(int rank){
    for(int i=0; i<10; i++){

        //produce item
        int item=produce_item();

        //wait for empty slot
        sem_wait(&buffer->empty);

        //insert into buffer
```

```
85          sem_wait(&buffer->mutex);
86          insert_item(buffer, item);
87          printf("produced --> item %d\n", item);
88          sem_post(&buffer->mutex);
89
90          //signal for full slot
91          sem_post(&buffer->full);
92
93      }
94  }
95
96  int produce_item(){
97      int item;
98      item = rand() % 100;
99      return item;
100 }
101
102 //consumer
103 void consumer(int rank){
104     for(int i=0; i<10; i++){
105
106         //wait for full slot
107         sem_wait(&buffer->full);
108
109         //remove from buffer
110         sem_wait(&buffer->mutex);
111         int item = remove_item(buffer);
112         printf("---------consumed --> item %d\n", item);
113         sem_post(&buffer->mutex);
114
115         //signal for empty slot
116         sem_post(&buffer->empty);
117
118     }
119 }
120
121 void insert_item(Buffer *buf, int item){
122     buf->items[buf->in] = item;
123     buf->in = (buf->in +1) % BUFFER_SIZE;
124 }
125
126 int remove_item(Buffer *buf){
127     int item = buf->items[buf->out];
128     buf->out = (buf->out +1) % BUFFER_SIZE;
129     return item;
130 }
```

Η λογική είναι η ίδια με αυτή που επιγράφθηκε στη λύση με τα νήματα.

Αξίζει να σημειωθεί ότι τώρα που πρόκειται για διεργασίες η κοινή μνήμη του buffer ορίζεται στην main με την mmap με δικαιώματα γραφής και ανάγνωσης από όλες τις διεργασίες. Στο τέλος του προγράμματος με την munmap αποδεσμεύεται η κοινή μνήμη που είχε δεσμευτεί.

Αποτελέσματα:

```
basagianni@LAPTOP-GECQDV8A:/mnt/c/Users/georg/OneDrive - University of Patras/Desktop/Mathimata/7o/OS/frontistiriakes/f3$ gcc f3_processes.c -o
f3_processes
basagianni@LAPTOP-GECQDV8A:/mnt/c/Users/georg/OneDrive - University of Patras/Desktop/Mathimata/7o/OS/frontistiriakes/f3$ ./f3_processes
produced --> item 83
produced --> item 83
produced --> item 86
produced --> item 77
produced --> item 86
---------consumed --> item 83
---------consumed --> item 83
---------consumed --> item 86
---------consumed --> item 77
---------consumed --> item 86
produced --> item 77
---------consumed --> item 77
produced --> item 15
produced --> item 93
produced --> item 35
produced --> item 86
---------consumed --> item 15
---------consumed --> item 93
---------consumed --> item 35
---------consumed --> item 86
produced --> item 92
produced --> item 49
produced --> item 15
produced --> item 93
---------consumed --> item 92
---------consumed --> item 49
---------consumed --> item 15
produced --> item 35
---------consumed --> item 93
produced --> item 21
produced --> item 86
produced --> item 92
---------consumed --> item 35
---------consumed --> item 21
---------consumed --> item 86
produced --> item 49
---------consumed --> item 92
---------consumed --> item 49
produced --> item 21
---------consumed --> item 21
basagianni@LAPTOP-GECQDV8A:/mnt/c/Users/georg/OneDrive - University of Patras/Desktop/Mathimata/7o/OS/frontistiriakes/f3$
```