实验3 进程同步

## 实验要求：

见书p236

## 代码说明：

buffer.h 文件里带一个全局的 buffer变量声明 和两个对buffer变量进行操作的接口函数声明：

int insert\_item(buffer\_item item);

int remove\_item(buffer\_item\* item);

buffer.c 是对接口和全局变量的实现和定义。

main.c 是主程序，基本跟书上内容没差, osx不能用sem\_init就只能用sem\_open代替了。

## 

## 效果演示：

## 代码：

//buffer.h

#ifndef \_\_BUFFER\_H\_\_

#define \_\_BUFFER\_H\_\_

#include <unistd.h>

#define BUFFER\_SIZE 5

typedef int buffer\_item;

typedef int buffer\_idx;

typedef struct {

buffer\_item buf[BUFFER\_SIZE];

size\_t len;

buffer\_idx head;

buffer\_idx end;

} buffer\_t;

extern buffer\_t buffer;

int insert\_item(buffer\_item item);

int remove\_item(buffer\_item\* item);

#endif

//buffer.c

#include "buffer.h"

buffer\_t buffer = {

.len = 0,

.head = 0,

.end = 0

};

int insert\_item(buffer\_item item)

{

if (buffer.len >= BUFFER\_SIZE){

return -1;

}else{

buffer.len++;

buffer.buf[buffer.end] = item;

buffer.end = (buffer.end + 1) % BUFFER\_SIZE;

return 0;

}

return -1;

}

int remove\_item(buffer\_item\* item)

{

if (buffer.len <= 0){

return -1;

}else{

buffer.len--;

(\*item) = buffer.buf[buffer.head];

buffer.head = (buffer.head + 1) % BUFFER\_SIZE;

return 0;

}

return -1;

}

//main.c

#include <stdlib.h>

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#include "buffer.h"

void \*producer(void\* pnumber);

void \*consumer(void\* pnumber);

pthread\_mutex\_t mutex;

sem\_t \*empty, \*full;

const char \*empty\_name = "Empty!";

const char \*full\_name = "Full!"; //Die!!

int main(int argc, char const \*argv[])

{

// wtf,osx sem\_init 不能用

// sem\_init(&empty, 0, 5);

// sem\_init(&full, 0, 0);

int i;

int sleep\_time = atoi(argv[1]);

int pro\_thread\_count = atoi(argv[2]);

int con\_thread\_count = atoi(argv[3]);

if ((empty = sem\_open(empty\_name, O\_CREAT, 0644, 5)) == SEM\_FAILED) {

perror("semaphore initilization");

exit(1);

}

if ((full = sem\_open(full\_name, O\_CREAT, 0644, 0)) == SEM\_FAILED) {

perror("semaphore initilization");

exit(1);

}

pthread\_t pro, con;

int \*pnumber = malloc(sizeof(int));

for (i = 0; i < pro\_thread\_count; ++i)

{

\*pnumber = i;

pthread\_create(&pro, NULL, producer, (void\*)pnumber);

}

for (i = 0; i < con\_thread\_count; ++i)

{

\*pnumber = i;

pthread\_create(&con, NULL, consumer, (void\*)pnumber);

}

// pthread\_join(pro, NULL);

// pthread\_join(con, NULL);

sleep(sleep\_time);

pthread\_mutex\_destroy(&mutex);

// sem\_destroy(&empty);

// sem\_destroy(&full);

sem\_unlink(empty\_name);

sem\_unlink(full\_name);

printf("program exit\n");

return 0;

}

void \*producer(void\* pnumber)

{

buffer\_item item;

int number = \*((int\*)pnumber);

while(1) {

sleep(rand() % 3);

item = rand();

sem\_wait(empty);

pthread\_mutex\_lock(&mutex);

printf("producer%d produced %d\n", number, item);

if (insert\_item(item)){

fprintf(stderr, "report error condition");

}

pthread\_mutex\_unlock(&mutex);

sem\_post(full);

}

}

void \*consumer(void\* pnumber)

{

buffer\_item item;

int number = \*((int\*)pnumber);

while(1) {

sleep(rand() % 3);

item = rand();

sem\_wait(full);

pthread\_mutex\_lock(&mutex);

if (remove\_item(&item)){

fprintf(stderr, "report error condition");

}else{

printf("consumer%d consumed %d\n", number, item);

}

pthread\_mutex\_unlock(&mutex);

sem\_post(empty);

}

}