Project 4: 生产者消费者问题

516030910259 刘欣鹏

1. 实验目的

通过实现对有限缓冲的生产者消费者问题的简单解决方案，加深对线程、线程间资源共享的理解。

1. 实验原理
   1. 缓冲区

从内部来说，缓冲区是一个元数据类型为buffer\_item的固定大小的数组。而从实现上来说，这个数组可以按环形队列处理。缓冲区通过insert\_item()和remove\_item()为生产者线程和消费者线程分别使用。

主函数main()接受三个命令行参数：睡眠多长之后才终止；生产者线程数量；消费者线程数量。它将缓冲初始化，创建生产者与消费者线程，睡眠一段时间，并在被唤醒时终止应用程序。

* 1. 生产者与消费者线程

生产者线程不断交替执行如下两个阶段：睡眠一段随机时间，向缓冲插入一个随机数。消费者也睡眠随机时间，在醒后，从缓冲内取出一项。

* 1. Pthread互斥量与信号锁

互斥锁采用pthread\_mutex\_t数据类型。pthread\_mutex\_init(&mutex,NULL)创建互斥锁。pthread\_mutex\_lock()和pthread\_mutex\_unlock()用来获取与释放锁。

本项目使用无名称信号量sem\_t数据类型。函数sem\_init()用于创建并初始化一个信号量，有三个参数：信号量指针，表示共享级别的标记，信号量的初始值。经典信号量操作分别为sem\_wait()和sem\_post()。

1. 实验步骤
   1. 源代码procon.c

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

#include <stdio.h>

#include <string.h>

#define BUFFER\_SIZE 5

typedef int buffer\_item;

buffer\_item buffer[BUFFER\_SIZE];

pthread\_mutex\_t mutex;

sem\_t empty,full;

pthread\_t pro\_id[10],con\_id[10];

pthread\_attr\_t pro\_attr[10],con\_attr[10];

int l,r;

int insert\_item(buffer\_item item){

sem\_wait(&empty);

pthread\_mutex\_lock(&mutex);

r++;

if (r==5) r=0;

buffer[r]=item;

pthread\_mutex\_unlock(&mutex);

sem\_post(&full);

return 0;

}

int remove\_item(buffer\_item item){

int x=0;

sem\_wait(&full);

pthread\_mutex\_lock(&mutex);

x=buffer[l];

buffer[l]=item;

l++;

if (l==5) l=0;

pthread\_mutex\_unlock(&mutex);

sem\_post(&empty);

return x;

}

void \*producer(int \*param){

buffer\_item r;

while (1){

r=rand()%10;

sleep(r);

r=rand()%10000;

if (!insert\_item(r))

printf("Producer %d produced %d successfully.\n",\*param,r);

}

}

void \*consumer(int \*param){

buffer\_item r,x=0;

while (1){

r=rand()%10;

sleep(r);

r=rand()%10000;

if (x=remove\_item(r))

printf("Consumer %d consumed %d with %d.\n",\*param,x,r);

}

}

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

if (argc!=4) {

printf("Invalid arguments!\n");

return 0;

}

int x,y,z,i;

x=atoi(argv[1]);

y=atoi(argv[2]);

z=atoi(argv[3]);

if (x<=0 || y>10 || z>10 || y<=0 || z<=0) {

printf("Too many threads!\n");

return 0;

}

void \*data=NULL;

pthread\_mutex\_init(&mutex,NULL);

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

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

srand((unsigned)time(NULL));

l=0;r=-1;

for (i=0;i<y;i++) pthread\_attr\_init(&pro\_attr[i]);

for (i=0;i<z;i++) pthread\_attr\_init(&con\_attr[i]);

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

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

\*data=i;

pthread\_create(&pro\_id[i],&pro\_attr[i],producer,data);

}

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

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

\*data=i;

pthread\_create(&con\_id[i],&con\_attr[i],consumer,data);

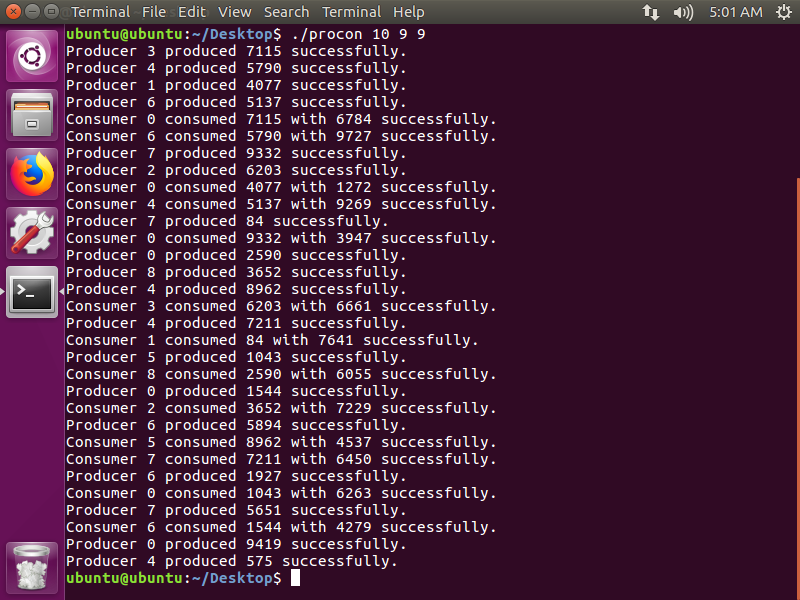
}

sleep(x);

return 0;

}

* 1. 测试结果



1. 心得体会

本实验加深了对线程及线程间资源共享的理解。

由于此前已接触过pthread库，本实验在实现上相对更加简单。