第十四组 多线程编程

小组成员：刘一丁，黎伟恒，冉然

一、实验目的

1、掌握基本的多线程编程技术

2、掌握基本的线程间同步技术

3、熟练使用pthread线程库调用接口

二、实验内容

读者-写者问题多线程实现

 一个数据集（如数据、文件等）被N个线程读写；

 一些线程只要求读数据集内容，称为读者 (Reader)，实验读者数

不少于6个；

 另些线程要求修改数据集内容，称为写者 (Writer)，实验写者数

不少于3个；

 多个读者可以同时读数据集内容，不需要互斥操作；

 一个写者不能和其他写者或读者同时访问数据集，换句话说，写

者和其他写者或读者之间必须互斥操作！

 读者优先：如果有读者，写者需要等待！用于更新不频繁或更新

影响不显著、读者较多的场合；

 写者优先：如果有写者，读者需要等待！用于更新频繁或更新影

响显著的场合。

1. 实验过程
2. 读者优先

/\*

\* 多线程,读者优先

\*/

#include "stdio.h"

#include <stdlib.h>

#include <pthread.h>

#include<semaphore.h>

#define N\_WRITER 30 //写者数目

#define N\_READER 5 //读者数目

#define W\_SLEEP 1 //控制写频率

#define R\_SLEEP 1 //控制读频率

pthread\_t wid[N\_WRITER],rid[N\_READER];

pthread\_mutex\_t mutex\_write;//同一时间只能一个人写文件,互斥

sem\_t sem\_read;//同一时间只能有一个人访问 readerCnt

int data = 0;

int readerCnt = 0;

void write()

{

int rd = rand();

printf("write %d\n",rd);

data = rd;

}

void read()

{

printf("read %d\n",data);

}

void \* writer(void \* in)

{

// while(1)

// {

pthread\_mutex\_lock(&mutex\_write);

printf("写线程id%d进入数据集\n",pthread\_self());

write();

printf("写线程id%d退出数据集\n",pthread\_self());

pthread\_mutex\_unlock(&mutex\_write);

sleep(W\_SLEEP);

// }

pthread\_exit((void \*) 0);

}

void \* reader (void \* in)

{

// while(1)

// {

sem\_wait(&sem\_read);

readerCnt++;

if(readerCnt == 1){

pthread\_mutex\_lock(&mutex\_write);

}

sem\_post(&sem\_read);

printf("读线程id%d进入数据集\n",pthread\_self());

read();

printf("读线程id%d退出数据集\n",pthread\_self());

sem\_wait(&sem\_read);

readerCnt--;

if(readerCnt == 0){

pthread\_mutex\_unlock(&mutex\_write);

}

sem\_post(&sem\_read);

sleep(R\_SLEEP);

// }

pthread\_exit((void \*) 0);

}

int main()

{

printf("多线程,读者优先\n");

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

sem\_init(&sem\_read,0,1);

int i = 0;

for(i = 0; i < N\_WRITER; i++)

{

pthread\_create(&wid[i],NULL,writer,NULL);

}

for(i = 0; i < N\_READER; i++)

{

pthread\_create(&rid[i],NULL,reader,NULL);

}

sleep(1);

return 0;

}

读者优先

（2）写者优先

#include "stdio.h"

#include <stdlib.h>

#include <pthread.h>

#include<semaphore.h>

#define N\_WRITER 5 //写者数目

#define N\_READER 20 //读者数目

#define W\_SLEEP 1 //控制写频率

#define R\_SLEEP 0.5 //控制读频率

pthread\_t wid[N\_WRITER],rid[N\_READER];

int data = 0;

int readerCnt = 0, writerCnt = 0;

pthread\_mutex\_t sem\_read;

pthread\_mutex\_t sem\_write;

pthread\_mutex\_t mutex\_write;

pthread\_mutex\_t mutex\_read;

void write()

{

int rd = rand();

printf("write %d\n",rd);

data = rd;

}

void read()

{

printf("read %d\n",data);

}

void \* writer(void \* in)

{

// while(1)

// {

sem\_wait(&sem\_write);

{//临界区,希望修改 writerCnt,独占 writerCnt

writerCnt++;

if(writerCnt == 1){

//阻止后续的读者加入待读队列

pthread\_mutex\_lock(&mutex\_read);

}

}

sem\_post(&sem\_write);

pthread\_mutex\_lock(&mutex\_write);

{//临界区，限制只有一个写者修改数据

printf("写线程id%d进入数据集\n",pthread\_self());

write();

printf("写线程id%d退出数据集\n",pthread\_self());

}

pthread\_mutex\_unlock(&mutex\_write);

sem\_wait(&sem\_write);

{//临界区,希望修改 writerCnt,独占 writerCnt

writerCnt--;

if(writerCnt == 0){

//阻止后续的读者加入待读队列

pthread\_mutex\_unlock(&mutex\_read);

}

}

sem\_post(&sem\_write);

sleep(W\_SLEEP);

// }

pthread\_exit((void \*) 0);

}

void \* reader (void \* in)

{

// while(1)

// {

//假如写者锁定了mutex\_read,那么成千上万的读者被锁在这里

pthread\_mutex\_lock(&mutex\_read);//只被一个读者占有

{//临界区

sem\_wait(&sem\_read);//代码段 1

{//临界区

readerCnt++;

if(readerCnt == 1){

pthread\_mutex\_lock(&mutex\_write);

}

}

sem\_post(&sem\_read);

}

pthread\_mutex\_unlock(&mutex\_read);//释放时,写者将优先获得mutex\_read

printf("读线程id%d进入数据集\n",pthread\_self());

read();

printf("读线程id%d退出数据集\n",pthread\_self());

sem\_wait(&sem\_read);//代码段2

{//临界区

readerCnt--;

if(readerCnt == 0){

pthread\_mutex\_unlock(&mutex\_write);//在最后一个并发读者读完这里开始禁止写者执行写操作

}

}

sem\_post(&sem\_read);

sleep(R\_SLEEP);

// }

pthread\_exit((void \*) 0);

}

int main()

{

printf("多线程,写者优先\n");

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

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

sem\_init(&sem\_write,0,1);

sem\_init(&sem\_read,0,1);

int i = 0;

for(i = 0; i < N\_READER; i++)

{

pthread\_create(&rid[i],NULL,reader,NULL);

}

for(i = 0; i < N\_WRITER; i++)

{

pthread\_create(&wid[i],NULL,writer,NULL);

}

sleep(1);

return 0;

}

写者优先

四实验总结和收获

1了解了基本的多线程编程技术

2掌握基本的线程间同步技术

3学会使用pthread线程库调用接口