**操作系统大作业（一）**

**1、线程死锁**

使用Mutex互斥锁，导致线程死锁。

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <unistd.h>

pthread\_mutex\_t m1 = PTHREAD\_MUTEX\_INITIALIZER;

pthread\_mutex\_t m2 = PTHREAD\_MUTEX\_INITIALIZER;

void \*thread1(void \*args)

{

pthread\_mutex\_lock(&m1);

sleep(1);

pthread\_mutex\_lock(&m2);

pthread\_mutex\_unlock(&m1);

pthread\_mutex\_unlock(&m2);

printf("thread1 finish\n");

}

void \*thread2(void \*args)

{

pthread\_mutex\_lock(&m2);

sleep(1);

pthread\_mutex\_lock(&m1);

pthread\_mutex\_unlock(&m1);

pthread\_mutex\_unlock(&m2);

printf("thread2 finish\n");

}

int main(void)

{

int rc,t = 0;

pthread\_t t1, t2;

printf("Creating thread...\n");

pthread\_create(&t1, NULL, thread1, (void \*)t);

pthread\_create(&t2, NULL, thread2, (void \*)t);

pthread\_join(t1, NULL);

pthread\_join(t2, NULL);

printf("Create thread finish\n");

pthread\_mutex\_destroy(&m1);

pthread\_mutex\_destroy(&m2);

return EXIT\_SUCCESS;

}

**2、死锁解决**

通过在线程间使用信号量解决死锁问题。

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

pthread\_mutex\_t m1 = PTHREAD\_MUTEX\_INITIALIZER;

pthread\_mutex\_t m2 = PTHREAD\_MUTEX\_INITIALIZER;

sem\_t s1;

void \*thread1(void \*args)

{

while(1) {

sem\_wait(&s1);

pthread\_mutex\_lock(&m1);

pthread\_mutex\_lock(&m2);

printf("thread1\n");

pthread\_mutex\_unlock(&m2);

pthread\_mutex\_unlock(&m1);

sem\_post(&s1);

sleep(1);

}

}

void \*thread2(void \*args)

{

while(1) {

sem\_wait(&s1);

pthread\_mutex\_lock(&m2);

pthread\_mutex\_lock(&m1);

printf("thread2\n");

pthread\_mutex\_unlock(&m1);

pthread\_mutex\_unlock(&m2);

sem\_post(&s1);

sleep(1);

}

}

int main(void)

{

int rc,t = 0;

pthread\_t t1, t2;

sem\_init(&s1, 0, 1);

printf("Creating thread...\n");

pthread\_create(&t1, NULL, thread1, (void \*)t);

pthread\_create(&t2, NULL, thread2, (void \*)t);

pthread\_join(t1, NULL);

pthread\_join(t2, NULL);

printf("Create thread finish\n");

pthread\_mutex\_destroy(&m1);

pthread\_mutex\_destroy(&m2);

sem\_destroy(&s1);

return EXIT\_SUCCESS;

}

**3、四个线程之间的通信**

通过使用信号量，生产者消费者模型，实现了要求的功能。

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

char s1[4] = "xxxx"; // thread1向其中写入abcd

char s2[4] = "xxxx"; // thread2向其中写入1234

int len1 = -1; // 当前已写入s1的位置(用于thread1,thread3通信)

int len2 = -1; // 当前已写入s2的位置(用于thread2,thread4通信)

int out = 0; // 为0时thread3可写，为1时thread4可写，两者交替写入buf(用于thread3,thread4通信)

int buf\_pos = 0; // 最终输出数组下标

char buf[10] = "xxxxxxxxxx"; // 最终输出数组(a1b2c3d4)

void writeChar(char c)

{

buf[buf\_pos++] = c;

printf(" write char:%c\n", c);

}

void \*thread1(void \*args)

{

char s[] = "abcd";

while(1) {

len1++;

if(len1 >= 4) {

break;

}

s1[len1] = s[len1];

printf("thread 1 write(%c)\n", s[len1]);

sleep(1);

}

printf("thread 1 finished.\n");

}

void \*thread2(void \*args)

{

char s[] = "1234";

while(1) {

len2++;

if(len2 >= 4) {

break;

}

s2[len2] = s[len2];

printf("thread 2 write(%c)\n", s[len2]);

sleep(1);

}

printf("thread 2 finished.\n");

}

void \*thread3(void \*args)

{

int i;

char c;

for(i = 0; i < 4; ++i) {

while(i > len1);

c = s1[len1];

printf("thread 3 read(%c)\n", c);

while(out == 1);

writeChar(c);

out = 1;

sleep(1);

}

printf("thread 3 finished.\n");

}

void \*thread4(void \*args)

{

int i;

char c;

for(i = 0; i < 4; ++i) {

while(i > len2);

c = s2[len2];

printf("thread 4 read(%c)\n", c);

while(out == 0);

writeChar(c);

out = 0;

sleep(1);

}

printf("thread 4 finished.\n");

}

int main(void)

{

pthread\_t t1, t2, t3, t4;

printf("creating thread...\n");

pthread\_create(&t1, NULL, thread1, NULL);

pthread\_create(&t2, NULL, thread2, NULL);

pthread\_create(&t3, NULL, thread3, NULL);

pthread\_create(&t4, NULL, thread4, NULL);

pthread\_join(t1, NULL);

pthread\_join(t2, NULL);

pthread\_join(t3, NULL);

pthread\_join(t4, NULL);

printf("all thread finished.\n");

buf[buf\_pos] = 0;

printf("%s\n", buf);

return 0;

}

程序执行结果如下：

creating thread...

thread 2 write(1)

thread 4 read(1)

thread 1 write(a)

thread 3 read(a)

write char:a

write char:1

thread 2 write(2)

thread 1 write(b)

thread 3 read(b)

write char:b

thread 4 read(2)

write char:2

thread 2 write(3)

thread 1 write(c)

thread 4 read(3)

thread 3 read(c)

write char:c

write char:3

thread 2 write(4)

thread 1 write(d)

thread 3 read(d)

write char:d

thread 4 read(4)

write char:4

thread 2 finished.

thread 1 finished.

thread 3 finished.

thread 4 finished.

all thread finished.

a1b2c3d4

执行结果与预期一致。