姓名：吴涵 学号：202101000720 成绩：

# 实验28父子进程线程异步性

1、创建父子进程，观察父子进程执行的顺序，了解进程执行的异步行为

源程序：

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

#include <stdlib.h>

int main()

{

pid\_t pid;

char\*msg;

int k;

printf("观察父子进程执行的先后顺序，了解调度算法的特征\n");

pid=fork();

switch(pid)

{

case 0:

msg="子进程在运行";

k=3;

break;

case -1:

msg="进程创建失败";

break;

default:

msg="父进程在运行";

k=5;

break;

}

while(k>0)

{

puts(msg);

sleep(1);

k--;

}

exit(0);

}

编译链接命令：gcc parent-child-fork.c -o parent-child-fork

运行命令：./parent-child-fork

交互与结果：



2、创建主线程和子线程，观察多线程执行的顺序，了解线程执行的异步行为

源程序：

#include <stdio.h>

#include <pthread.h>

#include <unistd.h>

static int run=1;

static int retvalue;

void \*threadfunc(void\*arg)

{

int\*running=arg;

printf("子线程初始化完毕，传入参数为：%d\n",\*running);

while(\*running)

{

printf("子线程正在运行\n");

usleep(1);

}

printf("子线程退出\n");

retvalue=8;

pthread\_exit((void\*)&retvalue);

}

int main()

{

pthread\_t pt;

int ret=-1;

int times=3;

int i=0;

int \*ret\_join=NULL;

ret=pthread\_create(&pt,NULL,(void\*)threadfunc,&run);

if(ret!=0)

{

printf("建立线程失败\n");

return 1;

}

printf("主线程创建子线程后在运行...\n");

usleep(1);

printf("主线程调用usleep(1)...\n");

for(;i<times;i++)

{

printf("主线程打印i=%d\n",i);

usleep(1);

}

run=0;

pthread\_join(pt,(void\*)&ret\_join);

printf("线程返回值为：%d\n",\*ret\_join);

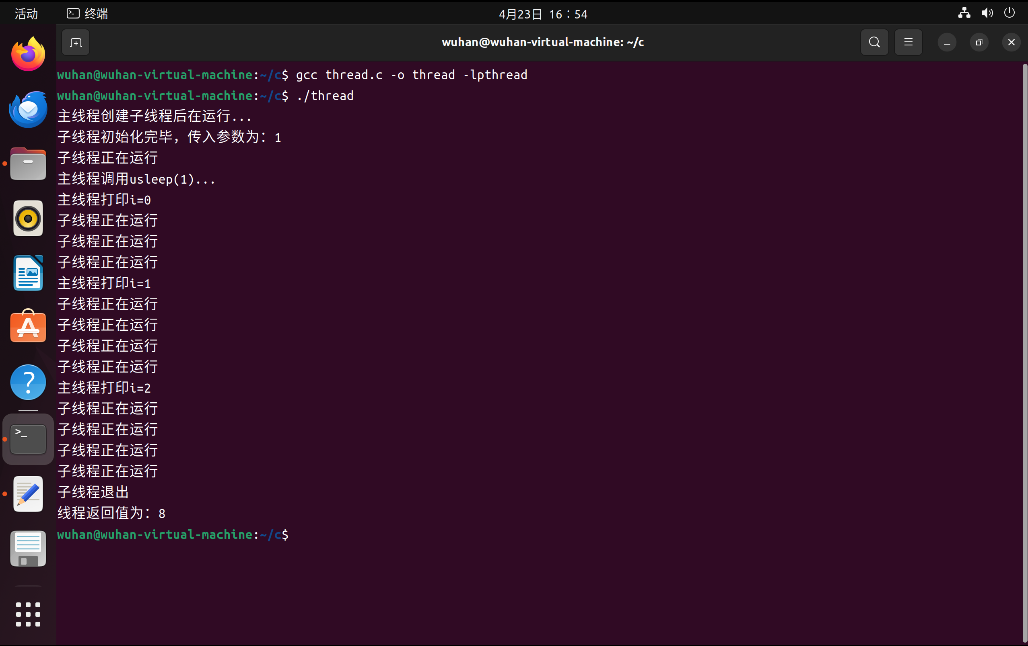
return 0;

}

编译链接命令：gcc thread.c -o thread -lpthread

运行命令：./thread

交互与结果：



3、多线程对共享变量的非互斥访问

源程序：

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <unistd.h>

#include <string.h>

int num = 30, count = 10;

void \*sub1(void \*arg)

{

    int i = 0, tmp;

    for (; i < count; i++)

    {

        tmp = num - 1;

        usleep(13);

        num = tmp;

        printf("线程1 num减1后值为: %d\n", num);

    }

    return ((void \*)0);

}

void \*sub2(void \*arg)

{

    int i = 0, tmp;

    for (; i < count; i++)

    {

        tmp = num - 1;

        usleep(31);

        num = tmp;

        printf("线程2 num减1后值为: %d\n", num);

    }

    return ((void \*)0);

}

int main(int argc, char \*\*argv)

{

    pthread\_t tid1, tid2;

    int err, i = 0, tmp;

    void \*tret;

    err = pthread\_create(&tid1, NULL, sub1, NULL);

    if (err != 0)

    {

        printf("pthread\_create error:%s\n", strerror(err));

        exit(-1);

    }

    err = pthread\_create(&tid2, NULL, sub2, NULL);

    if (err != 0)

    {

        printf("pthread\_create error:%s\n", strerror(err));

        exit(-1);

    }

    for (; i < count; i++)

    {

        tmp = num - 1;

        usleep(5);

        num = tmp;

        printf("main num减1后值为: %d\n", num);

    }

    printf("两个线程运行结束\n");

    err = pthread\_join(tid1, &tret);

    if (err != 0)

    {

        printf("can not join with thread1:%s\n", strerror(err));

        exit(-1);

    }

    printf("thread 1 exit code %d\n", (int)tret);

    err = pthread\_join(tid2, &tret);

    if (err != 0)

    {

        printf("can not join with thread1:%s\n", strerror(err));

        exit(-1);

    }

    printf("thread 2 exit code %d\n", (int)tret);

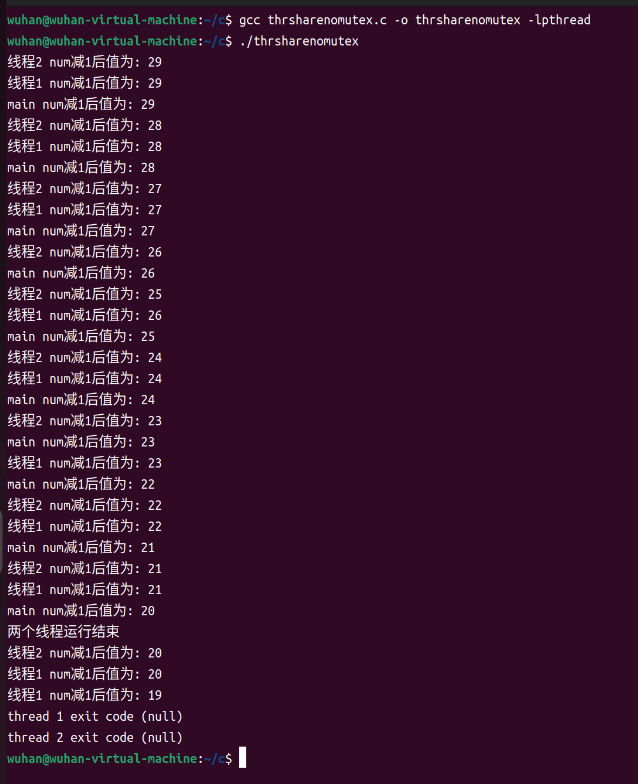
    return 0;

}

编译链接命令：gcc thrsharenomutex.c -o thrsharenomutex -lpthread

运行命令：./thrsharenomutex

交互与结果：



# 实验29同步与互斥

1、并发线程同步与互斥

源程序：

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <unistd.h>

#include <string.h>

int num = 30, count = 10;

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

void \*sub1(void \*arg)

{

int i = 0, tmp;

for (; i < count; i++)

{

pthread\_mutex\_lock(&mylock);

tmp = num - 1;

usleep(13);

num = tmp;

pthread\_mutex\_unlock(&mylock);

printf("线程1 num减1后值为: %d\n", num);

}

return ((void \*)0);

}

void \*sub2(void \*arg)

{

int i = 0, tmp;

for (; i < count; i++)

{

pthread\_mutex\_lock(&mylock);

tmp = num - 1;

usleep(31);

num = tmp;

pthread\_mutex\_unlock(&mylock);

printf("线程2 num减1后值为: %d\n", num);

}

return ((void \*)0);

}

int main(int argc, char \*\*argv)

{

pthread\_t tid1, tid2;

int err, i = 0, tmp;

void \*tret;

err = pthread\_create(&tid1, NULL, sub1, NULL);

if (err != 0)

{

printf("pthread\_create error:%s\n", strerror(err));

exit(-1);

}

err = pthread\_create(&tid2, NULL, sub2, NULL);

if (err != 0)

{

printf("pthread\_create error:%s\n", strerror(err));

exit(-1);

}

for (; i < count; i++)

{

pthread\_mutex\_lock(&mylock);

tmp = num - 1;

usleep(5);

num = tmp;

pthread\_mutex\_unlock(&mylock);

printf("main num减1后值为: %d\n", num);

}

printf("两个线程运行结束\n");

err = pthread\_join(tid1, &tret);

if (err != 0)

{

printf("can not join with thread1:%s\n", strerror(err));

exit(-1);

}

printf("thread 1 exit code %d\n", (int)tret);

err = pthread\_join(tid2, &tret);

if (err != 0)

{

printf("can not join with thread1:%s\n", strerror(err));

exit(-1);

}

printf("thread 2 exit code %d\n", (int)tret);

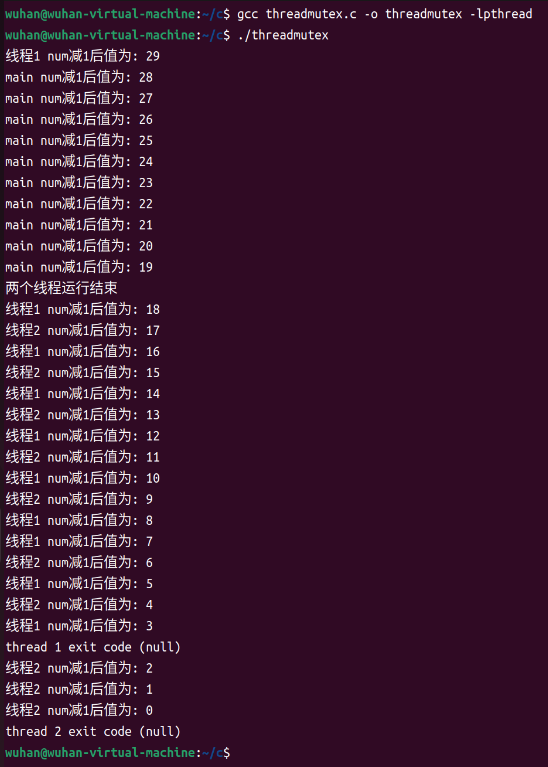
return 0;

}

编译链接命令：gcc threadmutex.c -o threadmutex -lpthread

运行命令：./threadmutex

交互与结果：



2、生产者-消费者同步与互斥试验

源程序：

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <pthread.h>

#include <unistd.h>

#include <signal.h>

#include <semaphore.h>

#define Maxbuf 10 // 缓冲单元数目

#define TimesOfOp 10 // 生产者、消费者循环读写缓冲区的次数

#define true 1

#define false 0

#define historynum 100 // 生产者、消费者读写历史记录数目

struct Circlebuf // 循环缓冲队列结构

{

int read; // 读指针

int write; // 写指针

int buf[Maxbuf]; // 缓冲区

} circlebuf;

sem\_t mutex; // 互斥信号量

sem\_t empty; // 空白缓冲区同步信号量

sem\_t full; // 满缓冲区同步信号量

char writehistory[historynum][30]; // 写历史

char readhistory[historynum][30]; // 读历史

int writehistorycount = 0; // 写历史计数器

int readhistorycount = 0; // 读历史计数器

char history[historynum][30]; // 缓冲区操作历史

int historycount = 0; // 缓冲区操作历史计数器

void writeCirclebuf(struct Circlebuf \*circlebuf, int \*value) // 向缓冲区中写一个值

{

circlebuf->buf[circlebuf->write] = (\*value);

sleep(1);

circlebuf->write = (circlebuf->write + 1) % Maxbuf;

}

int readCirclebuf(struct Circlebuf \*circlebuf)

{

int value = 0;

value = circlebuf->buf[circlebuf->read];

sleep(1);

circlebuf->buf[circlebuf->read] = 0;

circlebuf->read = (circlebuf->read + 1) % Maxbuf;

return value;

}

void sigend(int sig)

{

exit(0);

}

void \*productThread(void \*i)

{

int \*n = (int \*)i;

int t = TimesOfOp;

int writeptr;

while (t--)

{

sem\_wait(&empty);

sem\_wait(&mutex);

writeCirclebuf(&circlebuf, n);

if (circlebuf.write > 0)

writeptr = circlebuf.write - 1;

else

writeptr = Maxbuf - 1;

sprintf(writehistory[writehistorycount++], "生产者%d:缓冲区%d=%d", \*n, writeptr, \*n);

sprintf(history[historycount++], "生产者%d:缓冲区%d=%d\n", \*n, writeptr, \*n);

sem\_post(&mutex);

sem\_post(&full);

sleep(1);

}

}

void \*consumerThread(void \*i)

{

int \*n = (int \*)i;

int t = TimesOfOp;

int value = 0;

int readptr;

while (t--)

{

sem\_wait(&full);

sem\_wait(&mutex);

value = readCirclebuf(&circlebuf);

if (circlebuf.read > 0)

readptr = circlebuf.read - 1;

else

readptr = Maxbuf - 1;

sprintf(readhistory[readhistorycount++], "消费者%d:缓冲区%d=%d\n", \*n, readptr, value);

sprintf(history[historycount++], "消费者%d:缓冲区%d=%d\n", \*n, readptr, value);

sem\_post(&mutex);

sem\_post(&empty);

sleep(1);

}

}

int main()

{

int i, max;

int ConsNum = 0, ProdNum = 0, ret;

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

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

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

signal(SIGINT, sigend);

signal(SIGTERM, sigend);

circlebuf.read = circlebuf.write = 0;

for (i = 0; i < Maxbuf; i++)

circlebuf.buf[i] = 0;

printf("请输入生产者线程的数目 :");

scanf("%d", &ProdNum);

int \*pro = (int \*)malloc(ProdNum \* sizeof(int));

pthread\_t \*proid = (pthread\_t \*)malloc(ProdNum \* sizeof(pthread\_t));

printf("请输入消费者线程的数目 :");

scanf("%d", &ConsNum);

int \*con = (int \*)malloc(ConsNum \* sizeof(int));

pthread\_t \*conid = (pthread\_t \*)malloc(ConsNum \* sizeof(pthread\_t));

for (i = 1; i <= ConsNum; i++)

{

con[i - 1] = i;

ret = pthread\_create(&conid[i], NULL, consumerThread, (void \*)&con[i - 1]);

if (ret != 0)

{

printf("Create thread error");

exit(1);

}

}

for (i = 1; i <= ProdNum; i++)

{

pro[i - 1] = i;

ret = pthread\_create(&proid[i], NULL, productThread, (void \*)&pro[i - 1]);

if (ret != 0)

{

printf("Create thread error");

exit(1);

}

}

sleep((ConsNum + ProdNum) \* 10);

if (writehistorycount > readhistorycount)

max = writehistorycount;

else

max = readhistorycount;

for (i = 0; i < max; i++)

if ((i < writehistorycount) && (i < readhistorycount))

printf("%s | %s\n", writehistory[i], readhistory[i]);

else if (i < writehistorycount)

printf("%s | %s\n", writehistory[i], " ");

else

printf("%s | %s\n", " ", readhistory[i]);

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*缓冲池的操作历史为：\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

for (i = 0; i < historycount; i++)

printf("%s", history[i]);

sem\_destroy(&mutex);

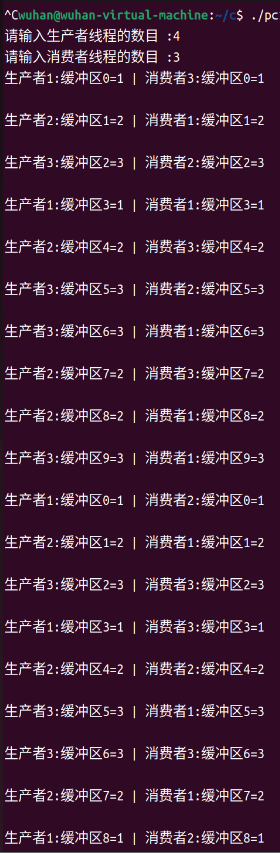
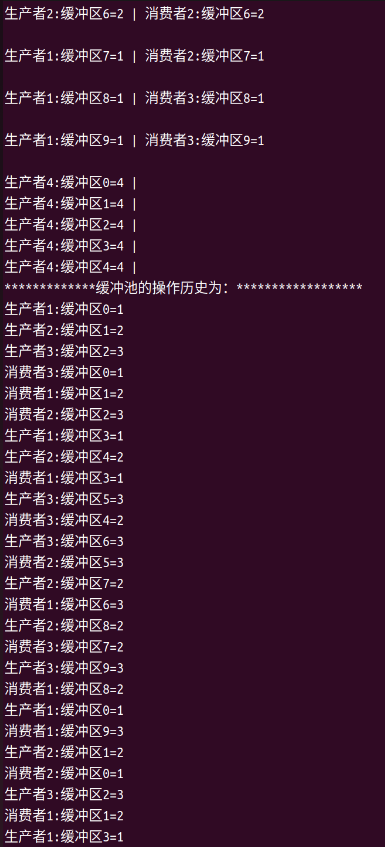
sem\_destroy(&empty);

sem\_destroy(&full);

}

编译链接命令：gcc pc1.c -o pc1 -lpthread

运行命令：./pc1

交互与结果：（非全部截图）

3、生产者-消费者未加同步与互斥机制的运行试验

源程序：

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <unistd.h>

#define Maxbuf 10 // 缓冲单元数目

#define TimesOfOp 10 // 生产者、消费者循环读写缓冲区的次数

#define historynum 100 // 生产者、消费者读写历史记录数目

struct Circlebuf // 循环缓冲队列结构

{

int read; // 读指针

int write; // 写指针

int buf[Maxbuf]; // 缓冲区

} circlebuf;

char writehistory[historynum][30]; // 写历史

char readhistory[historynum][30]; // 读历史

int writehistorycount = 0; // 写历史计数器

int readhistorycount = 0; // 读历史计数器

char history[historynum][30]; // 缓冲区操作历史

int historycount = 0; // 缓冲区操作历史计数器

void writeCirclebuf(struct Circlebuf \*circlebuf, int \*value) // 向缓冲区中写一个值

{

circlebuf->buf[circlebuf->write] = (\*value);

sleep(1);

circlebuf->write = (circlebuf->write + 1) % Maxbuf;

}

int readCirclebuf(struct Circlebuf \*circlebuf)

{

int value = 0;

value = circlebuf->buf[circlebuf->read];

sleep(1);

circlebuf->buf[circlebuf->read] = 0;

circlebuf->read = (circlebuf->read + 1) % Maxbuf;

return value;

}

void \*productThread(void \*arg)

{

int \*n = (int \*)arg;

int t = TimesOfOp;

int writeptr;

while (t--)

{

writeCirclebuf(&circlebuf, n);

if (circlebuf.write > 0)

writeptr = circlebuf.write - 1;

else

writeptr = Maxbuf - 1;

sprintf(writehistory[writehistorycount++], "生产者%d:缓冲区%d=%d", \*n, writeptr, \*n);

sprintf(history[historycount++], "生产者%d:缓冲区%d=%d\n", \*n, writeptr, \*n);

sleep(1);

}

}

void \*consumerThread(void \*arg)

{

int \*n = (int \*)arg;

int t = TimesOfOp;

int value = 0;

int readptr;

while (t--)

{

value = readCirclebuf(&circlebuf);

if (circlebuf.read > 0)

readptr = circlebuf.read - 1;

else

readptr = Maxbuf - 1;

sprintf(readhistory[readhistorycount++], "消费者%d:缓冲区%d=%d\n", \*n, readptr, value);

sprintf(history[historycount++], "消费者%d:缓冲区%d=%d\n", \*n, readptr, value);

sleep(1);

}

}

int main()

{

int i, max;

int ConsNum = 0, ProdNum = 0, ret;

circlebuf.read = circlebuf.write = 0;

for (i = 0; i < Maxbuf; i++)

circlebuf.buf[i] = 0;

printf("请输入生产者线程的数目 :");

scanf("%d", &ProdNum);

pthread\_t \*proid = (pthread\_t \*)malloc(ProdNum \* sizeof(pthread\_t));

int \*proArgs = (int \*)malloc(ProdNum \* sizeof(int));

for (i = 0; i < ProdNum; i++)

proArgs[i] = i + 1;

printf("请输入消费者线程的数目 :");

scanf("%d", &ConsNum);

pthread\_t \*conid = (pthread\_t \*)malloc(ConsNum \* sizeof(pthread\_t));

int \*conArgs = (int \*)malloc(ConsNum \* sizeof(int));

for (i = 0; i < ConsNum; i++)

conArgs[i] = i + 1;

for (i = 0; i < ConsNum; i++)

{

ret = pthread\_create(&conid[i], NULL, consumerThread, (void \*)&conArgs[i]);

if (ret != 0)

{

printf("Create thread error");

exit(1);

}

}

for (i = 0; i < ProdNum; i++)

{

ret = pthread\_create(&proid[i], NULL, productThread, (void \*)&proArgs[i]);

if (ret != 0)

{

printf("Create thread error");

exit(1);

}

}

sleep((ConsNum + ProdNum) \* 10);

if (writehistorycount > readhistorycount)

max = writehistorycount;

else

max = readhistorycount;

for (i = 0; i < max; i++)

{

if ((i < writehistorycount) && (i < readhistorycount))

printf("%s | %s\n", writehistory[i], readhistory[i]);

else if (i < writehistorycount)

printf("%s | %s\n", writehistory[i], " ");

else

printf("%s | %s\n", " ", readhistory[i]);

}

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*缓冲池\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

for (i = 0; i < historycount; i++)

printf("%s", history[i]);

free(proid);

free(proArgs);

free(conid);

free(conArgs);

return 0;

}

编译链接命令：gcc pc2.c -o pc2 -lpthread

运行命令：./pc2

交互与结果：（非全部截图）

