1进程间pthread\_mutex

A依赖的头文件

#include<pthread.h>

B 函数声明

int pthread\_mutexattr\_destroy(pthread\_mutexattr\_t \*attr);

int pthread\_mutexattr\_setpshared(pthread\_mutexattr\_t \*attr, int pshared);

int pthread\_mutexattr\_init(pthread\_mutexattr\_t \*attr);

关于pshared可供选的参数：

线程锁：PTHREAD\_PROCESS\_PRIVATE

进程锁：PTHREAD\_PROCESS\_SHARED

默认情况下是线程锁

C 案例说明：

|  |
| --- |
| #include <stdio.h>  #include <pthread.h>  #include <unistd.h>  #include <sys/stat.h>  #include <sys/types.h>  #include <fcntl.h>  #include <sys/mman.h>  #include <string.h>  #include <sys/wait.h>  struct mt {  int num;  pthread\_mutex\_t mutex;  pthread\_mutexattr\_t mutexattr;  };  int main(void) {  int fd,i,err;  struct mt \*mm;  pid\_t pid;  fd = open("mt\_test",O\_CREAT|O\_RDWR,0777);  /\*不需要write,文件里初始值为0\*/  ftruncate(fd,sizeof(\*mm));  mm = mmap(NULL,sizeof(\*mm),PROT\_READ|PROT\_WRITE,MAP\_SHARED,fd,0);  close(fd);  memset(mm,0,sizeof(\*mm));  /\*初始化互斥对象属性\*/  pthread\_mutexattr\_init(&mm->mutexattr);  /\*  \* 设置互斥对象为PTHREAD\_PROCESS\_SHARED共享，即可以在多个进程的  \*线程访问，PTHREAD\_PROCESS\_PRIVATE为同一进程的线程共享  \*/  pthread\_mutexattr\_setpshared(&mm->mutexattr,PTHREAD\_PROCESS\_SHARED);  pthread\_mutex\_init(&mm->mutex,&mm->mutexattr);  pid = fork();  if(pid == 0) {  /\*加10次。相当于加10\*/  for(i = 0;i < 10;i++) {  //在子进程上对文件进行上锁  pthread\_mutex\_lock(&mm->mutex);  (mm->num)++;  printf("num++:%d\n",mm->num);  //解锁  pthread\_mutex\_unlock(&mm->mutex);  sleep(1);  }  } else if(pid > 0) {  /\*父进程完成x+2,加10次，相当于加20\*/  for (i = 0;i < 10;i++) {  pthread\_mutex\_lock(&mm->mutex);  mm->num += 2;  printf("num+=2:%d\n",mm->num);  pthread\_mutex\_unlock(&mm->mutex);  sleep(1);  }  wait(NULL);  }  err = pthread\_mutex\_destroy(&mm->mutex);  if(err != 0) {  printf("%s\n",strerror(err));  }  /\* 父子均需要释放 \*/  munmap(mm,sizeof(\*mm));  unlink("mt\_test");  return 0;  } |

运行结果：



总结：

a进程间通信，可以通过内存映射的方式对文件进行操作。

b在上锁的情况下，数字相加后最后得到的是30，加锁后没有出现冲突。

2 文件锁

使用fcntl提供文件锁

struct flock {

….

short l\_type; /\*Type of lock:F\_RDLCK,F\_WRLCK,F\_UNLCK\*/

short l\_whence; /\*How to interpret l\_start:SEEK\_SET,SEET\_CUR,SEEK\_END\*/

off\_t l\_start; /\*Starting offset for lock\*/

off\_t l\_len; /\*Number of bytes to lock\*/

pid\_t l\_pid; /\*PID of process blocking our lock(F\_GETLK only)\*/

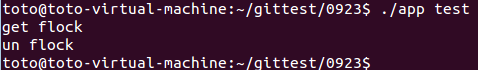
….

};

案例说明：

|  |
| --- |
| #include<stdio.h>  #include<sys/types.h>  #include<sys/stat.h>  #include<fcntl.h>  #include<unistd.h>  #include<stdlib.h>  void sys\_err(char \*str) {  perror(str);  exit(1);  }  int main(int argc,char \*argv[]) {  int fd;  struct flock f\_lock;  if(argc < 2) {  printf("./a.out filename\n");  exit(1);  }  if((fd = open(argv[1],O\_RDWR)) < 0) {  sys\_err("open");  }  //f\_lock.l\_type = F\_WRLCK  f\_lock.l\_type = F\_WRLCK;  f\_lock.l\_whence = SEEK\_SET;  f\_lock.l\_start = 0;  f\_lock.l\_len = 0; //0表示整个文件加锁    fcntl(fd,F\_SETLKW,&f\_lock);  printf("get flock\n");  sleep(10);  f\_lock.l\_type = F\_UNLCK;  fcntl(fd,F\_SETLKW,&f\_lock);  printf("un flock\n");  close(fd);  return 0;  } |

运行结果：



现象说明：当一个终端在运行的时候，同时打开另外一个终端，同时也执行./app test时发现开始的10秒内不允许操作，现象是test文件被锁住了。也就是说不能同时操作这个文件。