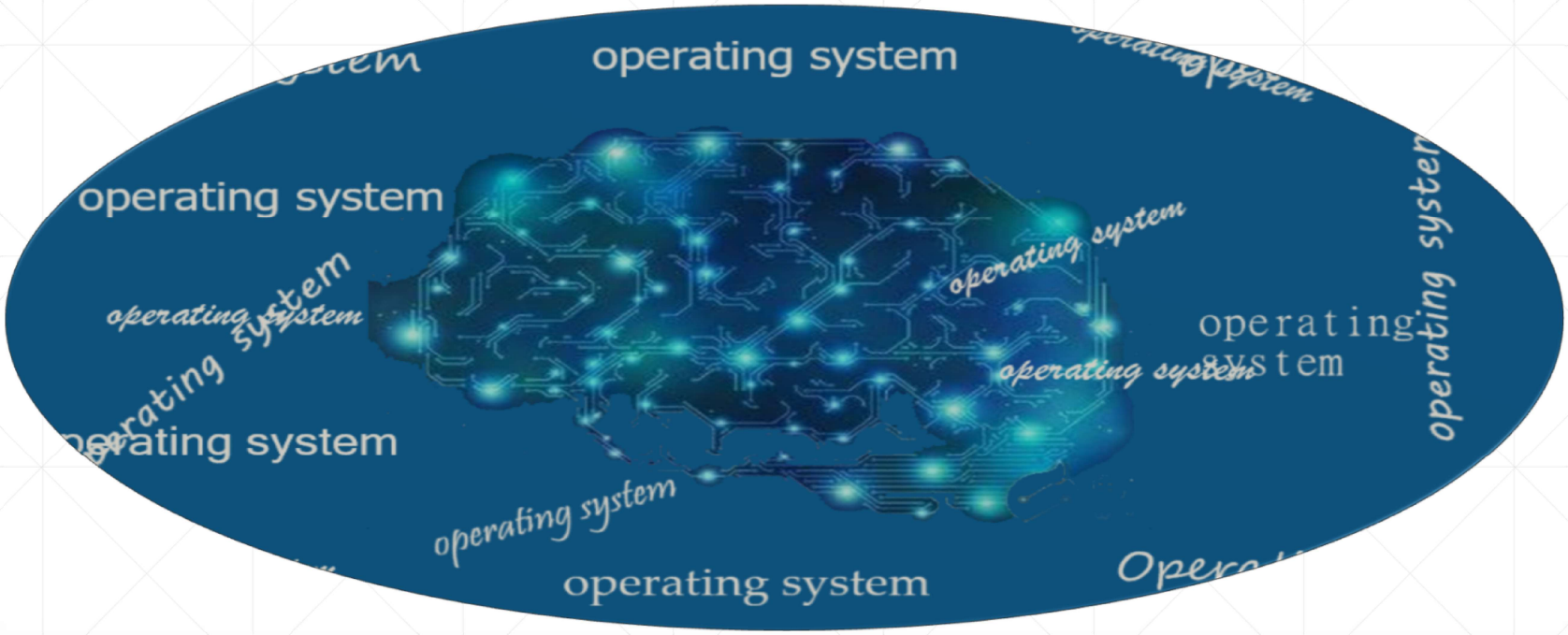


第十三章 线程与线程控制

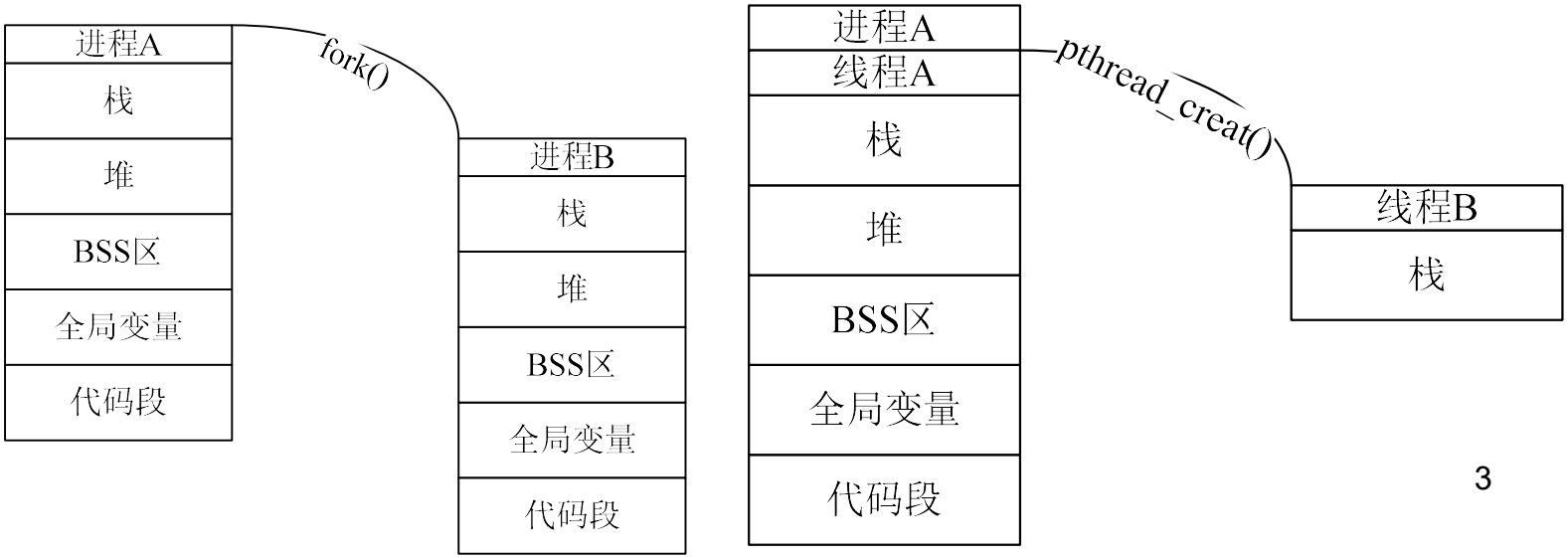
授课教师 电子邮箱:

主要内容

- 线程概述
- 线程与进程的比较
- 线程的控制

线程的基本概念

- **进程的概念体现出两个特点：资源（代码和数据空间、打开的文件等）以及调度执行。线程是进程内的独立执行代码的实体和调度单元**



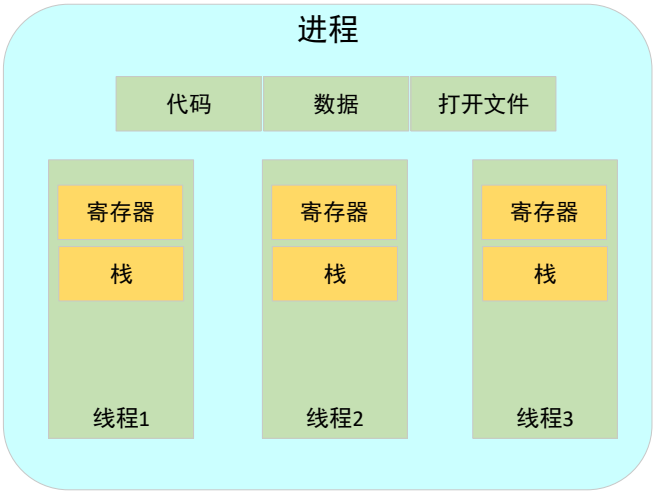
线程的基本概念

- 进程内的所有线程共享进程的很多资源（这种共享又带来了同步问题）

线程间共享	线程私有
进程指令	线程ID
全局变量	寄存器集合（包括PC和栈指针）
打开的文件	栈（用于存放局部变量）
信号处理程序	信号掩码
当前工作目录	优先级
用户ID	

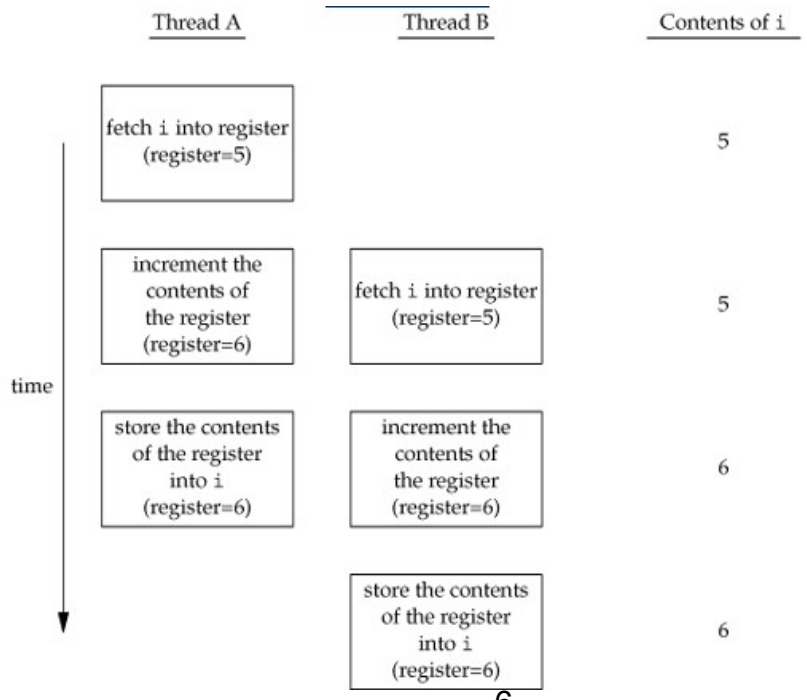
线程的数据共享

- 线程间共享的数据和资源：进程代码段、进程中的全局变量、进程打开的文件……
- 每个线程私有的数据和资源：线程ID、线程上下文（一组寄存器值的集合）、线程局部变量（存储在栈中）



线程的互斥问题

- 全局变量在数据段中（内存单元中）通常对一个全局变量的访问，要经历三个步骤
  - 将内存单元中的数据读入寄存器
  - 对寄存器中的值进行运算
  - 将寄存器中的值写回内存单元



主要内容

- 线程概述
- 线程与进程的比较
- 线程的控制

线程与进程的对比

- 线程只拥有少量在运行中必不可少的资源
  - PC指针：标识当前线程代码执行的位置
  - 寄存器：当前线程执行的上下文环境
  - 栈：用于实现函数调用、局部变量
  - 线程局部变量和私有数据（在栈中申请的数据）
  - 线程信号掩码（可以设置每个线程阻塞的信号）
- 进程占用资源多，线程占用资源少，使用灵活
- 线程不能脱离进程而存在，线程的层次关系，执行顺序并不明显，会增加程序的复杂度
- 没有通过代码显示创建线程的进程，可以看成是只有一个线程的进程

线程ID

- 同进程一样，每个线程也有一个线程ID
- 进程ID在整个系统中是唯一的，线程ID只在它所属的进程环境中唯一
- 线程ID的类型是pthread_t，在Linux中的定义如下：
  - typedef unsigned long int pthread_t
    (/usr/include/bits/pthreadtypes.h)

获取线程ID

- pthread_self函数可以让调用线程获取自己的线程ID
- 函数原型
  - 头文件：pthread.h
  - pthread_t pthread_self();
- 返回调用线程的线程ID

比较线程ID

- Linux中使用整型表示线程ID，而其他系统则不一定
- FreeBSD 5.2.1、Mac OS X 10.3用一个指向pthread结构的指针来表示pthread_t类型。
- 为了保证应用程序的可移植性，在比较两个线程ID是否相同时，建议使用pthread_equal函数

pthread_equal函数

- 该函数用于比较两个线程ID是否相同
- 函数原型
  - 头文件：pthread.h
  - int pthread_equal(pthread_t tid1, pthread_t tid2);
- 若相等则返回非0值，否则返回0

进程/线程控制操作对比

应用功能	线程	进程
创建	pthread_create	fork,vfork
退出	pthread_exit	exit
等待	pthread_join	wait、waitpid
取消/终止	pthread_cancel	abort
读取ID	pthread_self()	getpid()
同步互斥/通信机制	互斥锁、条件变量、读写锁	无名管道、有名管道、信号、消息队列、信号量、共享内存

主要内容

- 线程概述
- 线程与进程的比较
- 线程的控制

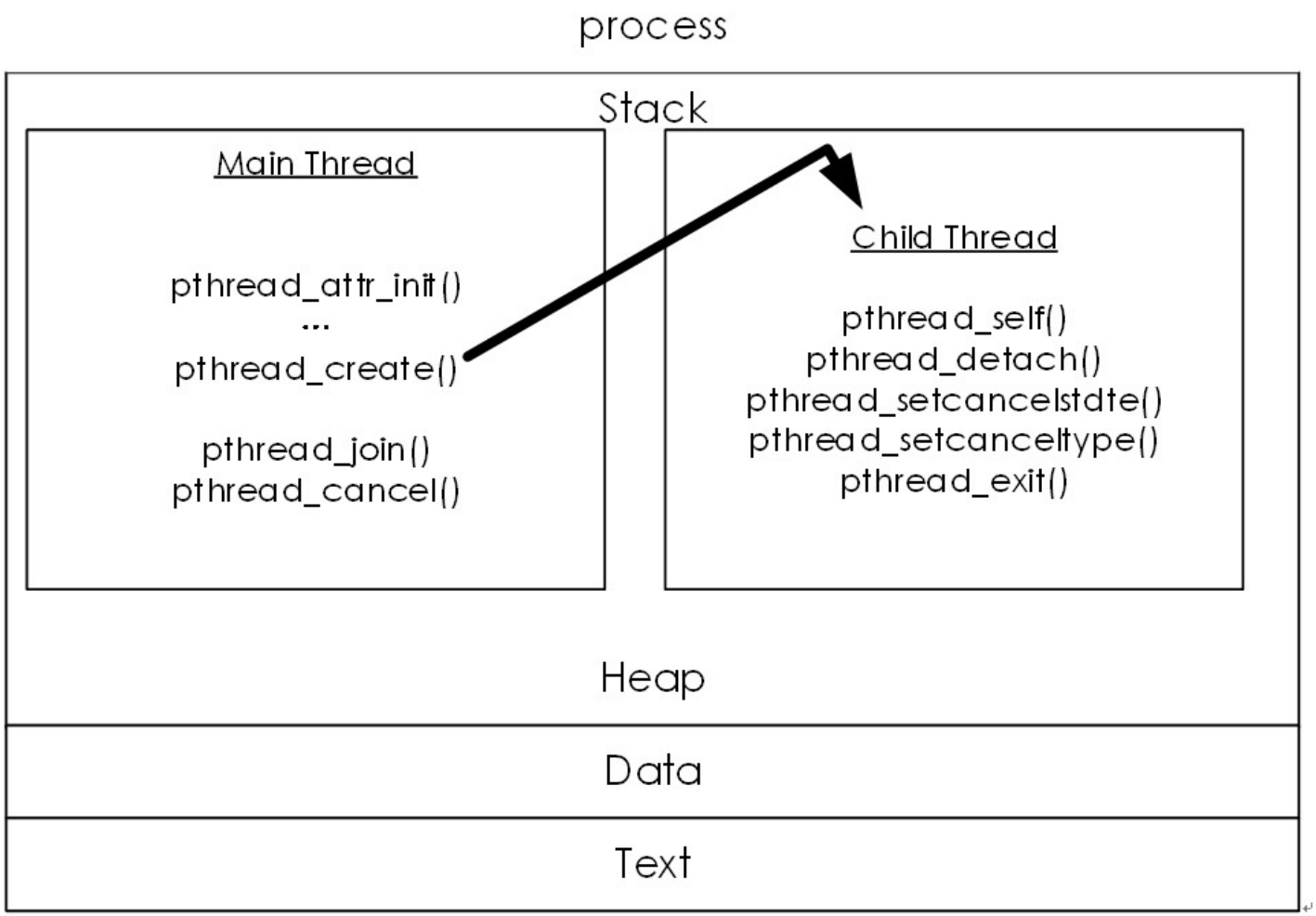
线程的创建

- pthread_create函数用于创建一个线程
- 函数原型
  - 头文件：pthread.h
  - int pthread_create(pthread_t *restrict tidp,
    const pthread_attr_t *restrict attr,
    void *(*start_rtn)(void *),
    void *restrict arg);
- 调用pthread_create函数的线程是所创建线程的父线程

线程的创建

- 参数
  - tidp：指向线程ID的指针，当函数成功返回时将存储所创建的子线程ID
  - attr：用于指定线程属性（一般直接传入空指针NULL，采用默认线程属性）
  - start_rtn：线程的启动例程函数指针，创建的线程首先执行该函数代码（可以调用其他函数）
  - arg：向线程的启动例程函数传递信息的参数
- 返回值
  - 成功返回0，出错时返回各种错误码

线程的创建


process
Stack
Main Thread
pthread_attr_init()
...
pthread_create()
pthread_join()
pthread_cancel()
Child Thread
pthread_self()
pthread_detach()
pthread_setcancelstdte()
pthread_setcanceltype()
pthread_exit()
Heap
Data
Text

创建子线程代码示例

```
void *childthread(void){
  int i;
  for(i=0;i<10;i++){
    printf( “childthread message\n” );
    sleep(100);}}
int main(){
  pthread_t tid;
  printf( “create childthread\n” );
   pthread_create(&tid,NULL,(void *) childthread,NULL);
   sleep(3);
   printf( “process exit\n” ); }
```

线程的终止

- 线程的三种终止方式
  - 线程从启动例程函数中返回，函数返回值作为线程的退出码
  - 线程被同一进程中的其他线程取消
  - 线程在任意函数中调用pthread_exit函数终止执行

线程终止函数

- 函数原型
  - 头文件：pthread.h
  - void pthread_exit(void *rval_ptr);
- 参数
  - rval_ptr：该指针将传递给pthread_join函数（与exit函数参数用法类似）

父线程等待子线程终止

- 函数原型
  - 头文件：pthread.h
  - int pthread_join(pthread_t thread,void **rval_ptr);
- 调用该函数的父线程将一直被阻塞，直到指定的子线程终止
- 返回值
  - 成功返回0，否则返回错误编号

pthread_join函数

- 参数
  - thread：需要等待的子线程ID
  - rval_ptr：（若不关心线程返回值，可直接将该参数设置为空指针NULL）
    - 若线程从启动例程返回，rval_ptr将包含返回码
    - 若线程被取消，rval_ptr指向的内存单元值置为PTHREAD_CANCELED
    - 若线程通过调用pthread_exit函数终止，rval_ptr就是调用pthread_exit时传入的参数

创建并等待子线程代码示例

```
void *childthread(void){
  int i;
  for(i=0;i<10;i++){
    printf( "childthread message\n" );
    sleep(100);}}
int main(){
  pthread_t tid;
  printf( "create childthread\n" );
   pthread_create(&tid,NULL,(void *) childthread,NULL);
  pthread_join(tid,NULL);
  printf( "childthread exit process exit\n" ); }
```

取消线程

- 线程调用该函数可以取消同一进程中的其他线程（即让该线程终止）
- 函数原型
  - 头文件： pthread.h
  - int pthread_cancel(pthread_t tid);
- 参数与返回值
  - tid：需要取消的线程ID
  - 成功返回0， 出错返回错误编号

取消线程

- 在默认情况下，pthread_cancel函数与线程ID等于tid的线程自身调用pthread_exit函数（参数为PTHREAD_CANCELED）效果等同
- 线程可以选择忽略取消方式或者控制取消方式
- pthread_cancel并不等待线程终止，它仅仅是提出请求

线程清理处理函数

- 当线程终止时，可以调用自定义的线程清理处理函数，进行资源释放等操作。类似于atexit函数。
- 线程可以注册多个清理处理函数，这些函数被记录在栈中，它们的执行顺序与它们的注册顺序相反
- 线程清理处理函数的注册
  - 头文件：pthread.h
  - void pthread_cleanup_push(void (*rtn)(void *),void *arg);

线程清理处理函数

- 参数
  - rtn：清理函数，无返回值，包含一个类型为指针的参数
  - arg：当清理函数被调用时，arg将被传递给清理函数
- 清理函数被调用的时机
  - 线程自身调用pthread_exit时
  - 线程响应取消线程请求时
  - 以非0参数调用pthread_cleanup_pop时

线程清理处理函数

- pthread_cleanup_push必须和pthread_cleanup_pop成对出现，而且出现的地方必须在同一个作用域内
- 函数原型
  - void pthread_cleanup_pop(int execute);

    (pthread.h)

线程退出与清理示例

```
void cleanup(){
  printf( “cleanup\n” );}
void *test_cancel(void){
  pthread_clean_push(cleanup,NULL);
  printf( “test_cancel\n” );
  while(1)
  {
    printf( “test message\n” );
    sleep(1);
  }
  pthread_cleanup_pop(1);}
int main(){
  phtread_t tid;
  phtread_create(&tid,NULL,(void *)test_cancel,NULL);
  sleep(2);
  phread_cancel(tid);
  pthread_join(tid,NULL);}
```

pthread_detach函数

- 在任何一个时间点上，线程是可结合的（joinable）或者是分离的（detached）
  - 可结合的线程能够被父线程回收其资源和杀死。在被父线程回收之前，它的存储器资源（例如栈）是不释放的
  - 分离的线程是不能被父线程回收或杀死的，它的存储器资源在它终止时由系统自动释放
- 若线程已经处于分离状态，线程的底层存储资源可以在线程终止时立即被收回
- 当线程被分离时，并不能用pthread_join函数等待它的终止状态，此时pthread_join返回EINVAL
- pthread_detach函数可以使线程进入分离状态

pthread_detach函数

- 函数原型
  - 头文件：pthread.h
  - int pthread_detach(pthread_t tid);
- 参数与返回值
  - tid：进入分离状态的线程的ID
  - 成功返回0，出错返回错误编号

线程属性

- 前面讨论pthread_create时，针对线程属性，传入的参数都是NULL。
- 实际上，可以通过构建pthread_attr_t结构体，设置若干线程属性
- 要使用该结构体，必须首先对其进行初始化；使用完毕后，需要销毁它

线程属性

■POSIX规定的一些线程属性

```
/* Attributes for threads.   */
typedef struct __pthread_attr_s{
   int __detachstate;                    //是否可以被等待
   int __schedpolicy;                    //调度策略
   struct __sched_param __schedparam;    //某调用策略的参数
   int __inheritsched;                   //是否继承创建线程的调度策略
   int __scope;                          //争用范围
   size_t __guardsize;                   //栈保护区大小
   int __stackaddr_set;                  //
   void *__stackaddr;                    //栈起始地址
   size_t __stacksize;                   //栈大小
} pthread_attr_t;
```

初始化和销毁

- 函数原型

```
#include<pthread.h>
int pthread_attr_init(pthread_attr_t *attr);
int pthread_attr_destroy(pthread_attr_t *attr);
```

- 参数与返回值
  - 成功返回0，否则返回错误编号
  - attr：线程属性，确保attr指向的存储区域有效
  - 为了移植性，pthread_attr_t结构对应用程序是不可见的，应使用设置和查询等函数访问属性

线程属性操作示例代码

```
#include <pthread.h>
#include <sched.h>
int main(void)
{
  int ret;
  pthread_t pid;   /* 线程ID */
  pthread_attr_t pattr;  /* 线程属性结构体 */
  struct sched_param param; /* 线程优先级结构体 */
  pthread_attr_init(&pattr);  /* 初始化线程属性对象，这时是默认值 */
  pthread_attr_setscope(&pattr, PTHREAD_SCOPE_SYSTEM); /* 设置线程绑定 */
  pthread_attr_getschedparam(&pattr, &param); /* 修改线程优先级 */
  param.sched_priority = 20;
  pthread_attr_setschedparam(&pattr, &param);
  /* 使用设置好的线程属性来创建一个新的线程 */
  ret = pthread_create(&pid, &pattr, (void *)thread, NULL);
  ……
```

初始化线程属性对象

属性	缺省值	描述
scope	PTHREAD_SCOPE_PROCESS	新线程与进程中的其他线程发生竞争
detachstate	PTHREAD_CREATE_JOINABLE	线程可以被其它线程等待
stackaddr	NULL	新线程具有系统分配的栈地址
stacksize	0	新线程具有系统定义的栈大小
priority	0	新线程的优先级为0
inheritsched	PTHREAD_EXPLICIT_SCHED	新线程不继承父线程调度优先级
schedpolicy	SCHED_OTHER	新线程使用优先级调用策略

获取线程栈属性

- 函数原型

```
#include<pthread.h>
int pthread_attr_getstack(
        const pthread_attr_t *attr,
         void **stackaddr, size_t *stacksize);
```

- 参数与返回值
  - attr：线程属性
  - stackaddr：该函数返回的线程栈的最低地址
  - stacksize：该函数返回的线程栈的大小
  - 成功返回0，否则返回错误编号

设置线程栈属性

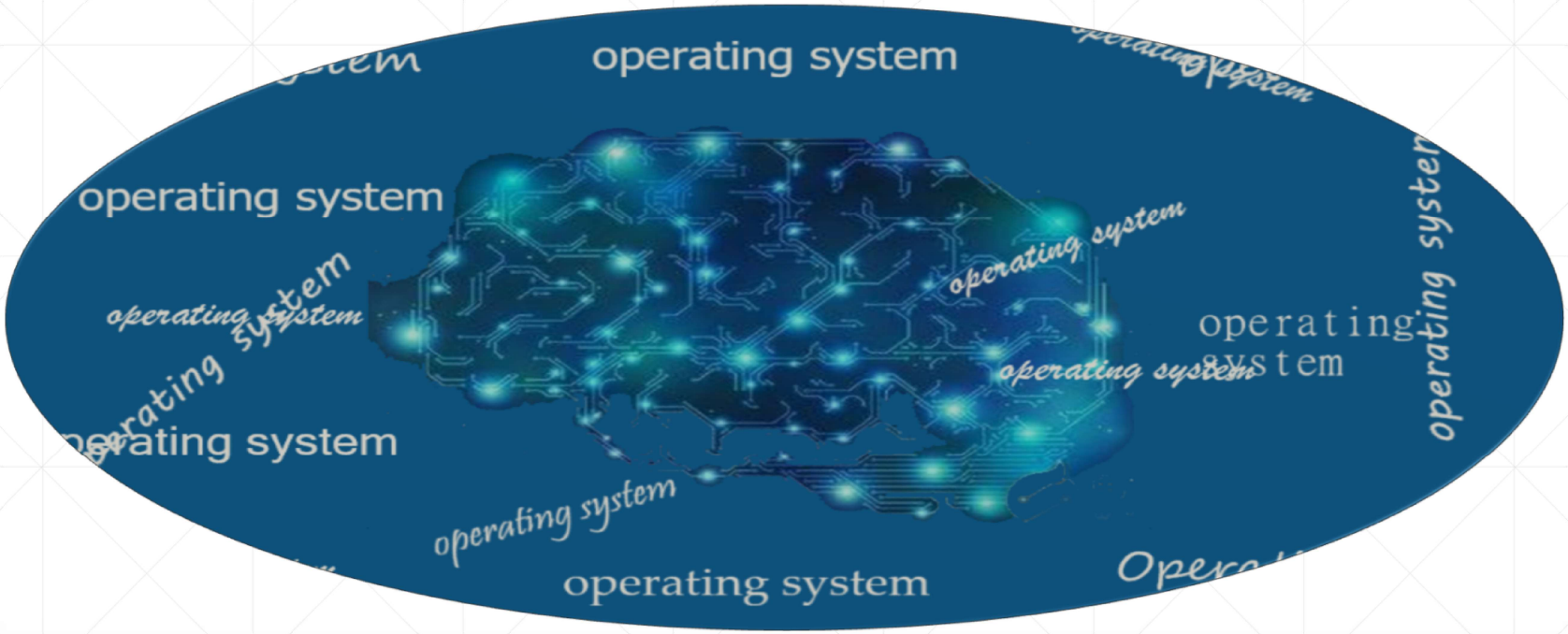
- 函数原型

```
#include<pthread.h>
int pthread_attr_setstack(
        const pthread_attr_t *attr,
         void *stackaddr, size_t *stacksize);
```

- 当用完线程栈时，可以再分配内存，并调用本函数设置新建栈的位置

设置线程栈属性

- 参数与返回值
  - attr：线程属性
  - stackaddr：新栈的内存单元的最低地址，通常是栈的开始位置；对于某些处理器，栈是从高地址向低地址方向伸展的，stackaddr就是栈的结尾
  - stacksize：新栈的大小
  - 成功返回0，否则返回错误编号



感谢观看！

授课教师：

电子邮箱：