# 操作系统课程设计

# 一个用户态线程的实现

## 设计目标

本次课程设计需要开发一个用户级线程库，功能包括完整的线程创建/删除/joining，互斥锁，条件变量，以及一个基于优先级的调度器。完成此用户级线程库，你可以使用你实现的线程库写多线程的应用，替代linux中的 pthreads.

这个线程库的骨架代码见：

https://gitee.com/scut\_wuyimin/uthread.git

## 课程设计内容

uthreads 是一个用户级线程库，与 pthreads 接口大致类似。这个线程库的功能包括线程的创建，线程可以是 joined 或 detached，一个简单基于优先级的调度器，互斥锁和条件变量。你需要写 uthreads 代码的主要部分，但我已经把这个线程库的骨架部分写好了，你只需要写uthread的关键部分。当然，你也需要仔细阅读已有的代码。以下这些函数你可以直接使用：

// uthread.c

char \*alloc\_stack();

void free\_stack(char\*stack);

void make\_reapable(uthread\_t \*uth);

// uthread\_ctx.c

void uthread\_makecontext(uthread\_ctx\_t \*ctx, char \*stack,

int stacksz, void (\*func)(),

long arg1, void \*arg2);

void uthread\_swapcontext(uthread\_ctx\_t \*oldctx,

uthread\_ctx\_t \*newctx);

以及 uthread\_queue.h 定义的所有函数和 list.h定义的所有宏.

你需要实现的 uthreads API ：

*// uthread.c*

*void uthread\_init();*

*int uthread\_create(uthread\_id\_t \*uidp, uthread\_func\_t func,*

*long arg1, void \*arg2, int prio);*

*void uthread\_exit(void \*status);*

*int uthread\_join(uthread\_id\_t uid, void \*\*return\_value);*

*uthread\_id\_t uthread\_alloc();*

*void uthread\_destroy (uthread\_t \*uth);*

*// uthread\_sched.c*

*void uthread\_yield();*

*void uthread\_block();*

*void uthread\_wake(uthread\_t \*uthr);*

*int uthread\_setprio(uthread\_id\_t id, int prio);*

*void uthread\_switch();*

*void clock\_interrupt(int sig);*

*// uthread\_mtx.c*

*void uthread\_mtx\_init(uthread\_mtx\_t \*mtx);*

*void uthread\_mtx\_lock(uthread\_mtx\_t \*mtx);*

*int uthread\_mtx\_trylock(uthread\_mtx\_t \*mtx);*

*void uthread\_mtx\_unlock(uthread\_mtx\_t \*mtx);*

*// uthread\_cond.c*

*void uthread\_cond\_init(uthread\_cond\_t \*cond);*

*void uthread\_cond\_wait(uthread\_cond\_t \*cond,*

*uthread\_mtx\_t \*mtx);*

*void uthread\_cond\_broadcast(uthread\_cond\_t \*cond);*

*void uthread\_cond\_signal(uthread\_cond\_t \*cond);*

看起来工作很多，但大多数函数都很短。需要你修改或实现的函数在代码中被标为 Function\_you\_need\_to\_implement宏。在命令行中输入： make nyi，你会看到所有你需要修改或实现的函数：

*wuyimin@EULER:~/uthread$ make nyi*

*./uthread\_cond.c:31: uthread\_cond\_init() UTHREADS*

*./uthread\_cond.c:42: uthread\_cond\_wait() UTHREADS*

*./uthread\_cond.c:53: uthread\_cond\_broadcast() UTHREADS*

*./uthread\_cond.c:63: uthread\_cond\_signal() UTHREADS*

*./uthread\_mtx.c:28: uthread\_mtx\_init() UTHREADS*

*./uthread\_mtx.c:38: uthread\_mtx\_lock() UTHREADS*

*./uthread\_mtx.c:48: uthread\_mtx\_trylock() UTHREADS*

*./uthread\_mtx.c:58: uthread\_mtx\_unlock() UTHREADS*

*./uthread.c:66: uthread\_init() UTHREADS*

*./uthread.c:94: uthread\_create() UTHREADS*

*./uthread.c:113: uthread\_exit() UTHREADS*

*./uthread.c:133: uthread\_join() UTHREADS*

*./uthread.c:156: uthread\_alloc() UTHREADS*

*./uthread.c:166: uthread\_destroy() UTHREADS*

*./uthread\_sched.c:43: uthread\_yield() UTHREADS*

*./uthread\_sched.c:54: uthread\_wake() UTHREADS*

*./uthread\_sched.c:67: uthread\_setprio() UTHREADS*

*./uthread\_sched.c:83: uthread\_switch() UTHREADS*

上述要实现的函数在源代码中有丰富的注释，详细解释了你要做的事情。以下简略说明一下。

使用线程库前首先要调用 uthread\_init()。这个函数只执行一次，负责初始化数据结构，比如全局 uthreads 矩阵和 ut\_curthr (这是当前正在执行的线程)。 uthread\_init() 中有一些特殊的代码，处理当前正在执行的进程的上下文 (即调用 uthread\_init()的进程)使其成为一个有效的 uthread 并设置 ut\_curthr为这个uthread，这个uthread称之为主线程，0号线程。详见 create\_first\_thr() 附近的注释信息。所有事情初始化后，即uthread\_init完成后，你可以使用线程 uthread\_create()创建线程。

一旦创建好线程，线程库需要能够调度这些线程。线程需要暂时 yield 处理器给另一个线程 (仍然是可执行的，即处于就绪态)，则需要调用 uthread\_yield()。 线程可以通过 uthread\_block() 进入睡眠态，也可以通过 uthread\_wake() 被唤醒。选择另一个线程运行发生在 uthread\_switch() 函数。调度器调度时，采用优先级调度算法，线程优先级可以由 uthread\_setprio()设置，数字越大优先级越高，相同优先级的线程则轮流使用CPU。每个优先级一个队列，相关的数据结构见 uthread\_sched.c 中的runq\_table .

本课程设计所设计实现的 uthreads 行为上与linux pthread 类似。因此你要实现的函数或api 与linux pthread 基本相同。实现时请参见pthread的说明。

Uthreads中的线程有6种状态，定义如下：

*typedef enum {*

*UT\_NO\_STATE, /\* 无效的线程状态 \*/*

*UT\_ON\_CPU, /\* 线程正在执行 \*/*

*UT\_RUNNABLE, /\* 线程可运行，就绪 \*/*

*UT\_WAIT, /\* 线程被阻塞 \*/*

*UT\_ZOMBIE, /\* 线程处于 zombie 状态，即已结束，但需要回收资源 \*/*

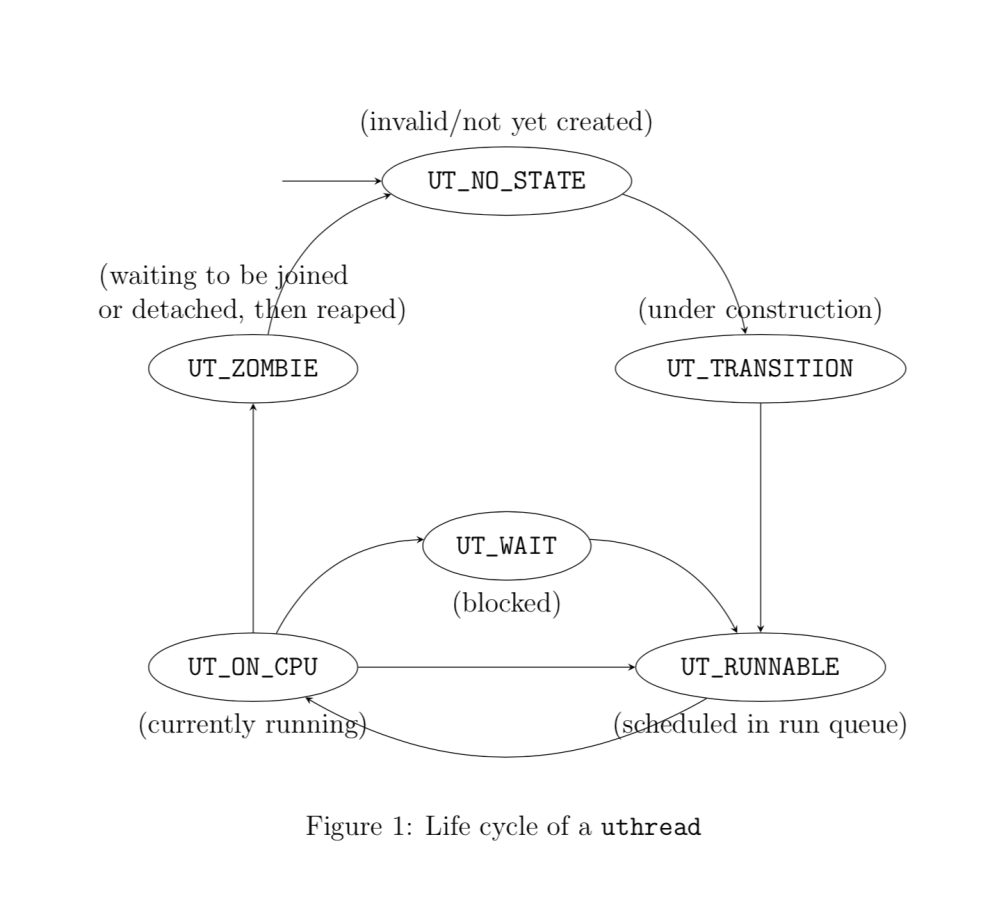
*UT\_TRANSITION, /\* 线程处于创建状态 \*/*

*UT\_JOINABLE, //线程结束时需要一个线程回收其资源*

*UT\_DETACHABLE, //线程结束时，不需要一个线程回收其资源*

*UT\_NUM\_THREAD\_STATES //线程状态数目*

*} uthread\_state\_t;*



Uthread 线程转换图

在uthreads 中永远不会有多于一个线程同时执行，但线程可被抢占。在 uthreads中，使用 uthread\_makecontext() 创建线程的机器上下文。如果需要改变当前正在执行线程 (在调度器)，需要调用 uthread\_swapcontext()。这使得当前 CPU 上下文被保存，新的上下文被执行。保存的上下文作为newctx 参数在之后调用 uthread\_swapcontext() 时被重新执行。

Uthread中有一个线程称之为Reaper 线程，也就是1号线程，负责清理已结束线程占用的资源。值得注意的是 reaper 并不清理已经结束但还没有joined的non-detached 线程，而是让 uthread\_join()去完成。我们在 uthread.c中给出了一个完整的 reaper 作为 reaper，仔细查看其实现，以理解其意义.

## 编译

编译完整的线程库以及测试程序：

*make*

查看需要实现的函数或代码：

*make nyi*

编译测试程序，使用项目中给定的链接库（主要用于查看正确的运行结果）：

*gcc test.c -luthread -o test*

不使用链接库方式，将所有代码编译为一个可执行文件（这并不是提交代码的正确方式，仅用于方便调试）：

*gcc \*.c -o test*

## 调试

一般来讲，在linux中使用 gdb 调试C程序，当然你也可以将gdb与其他IDE使用配合使用。但是，gdb 有时在调试多线程的程序时遇到有很多问题，关于如何使用gdb调试多线程的程序，请在网上查找相关的资料。在代码的关键处使用 assert()或者在需要调试的地方使用printf输出一些信息有助于理解和调试线程库，但要注意，printf并不是线程安全的函数。

另外，由于 uthreads 被编译为动态库，也给调试带来了不少困难。一个简单的办法是先把这个项目作为单个可执行程序编译调试，即使用下面命令编译：

*gcc -g \*.c -o test*

之后再使用gdb调试。调试没问题后，再make生成动态库。

## 测试代码

我们提供了一个测试程序test.c。你的uthread线程库完成后，编译这个测试程序并运行。这个测试程序运行时调用你的uthread线程库中的API，运行结果如下：

*wuyimin@EULER:~/uthreads and mthreads/uthreading$ ./test*

*joined with thread 2, exited 0.*

*joined with thread 3, exited 0.*

*joined with thread 4, exited 0.*

*joined with thread 5, exited 0.*

*joined with thread 6, exited 0.*

*joined with thread 7, exited 0.*

*joined with thread 8, exited 0.*

*joined with thread 9, exited 0.*

*joined with thread 10, exited 0.*

*joined with thread 11, exited 0.*

*joined with thread 12, exited 0.*

*joined with thread 13, exited 0.*

*joined with thread 14, exited 0.*

*joined with thread 15, exited 0.*

*joined with thread 16, exited 0.*

限于篇幅此处只列出了一部分，全部结果见result.txt文件。为了方便大家理解程序的运行结果，项目也提供了一个已经做好的线程库，文件名为libmyuthread.so。将这个文件重新命名为libuthread.so，使用下面命令编译test.c：

*wuyimin@EULER:~/temp/uthread$ gcc test.c -luthread -o test*

然后运行test程序，也可以得到相同的结果。

## ****设计****实现****步骤****

1. 在实体机或虚拟机上安装linux操作系统，推荐安装openEuler。并安装cscope、git、gcc、gdb、make、vim等软件包。安装根据你的需要，可以安装图形界面、vscode等IDE，并做好配置。
2. 在gitee.com上注册一个账号，把申请的账号和申请账号时用的电子邮箱填在文档中；
3. 课程教师收到大家填好的，含有每位同学gitee账号和电子邮箱的文档后，逐个将各位同学拉入项目；
4. 各位同学会收到一个邮件，请注意查收邮件，并确认加入；
5. 通过浏览器登录你的gitee.com，找到课程设计项目（uthread）,fork为你的项目；
6. 使用git clone 命令将你的项目clone 到你的本地（可以使用命令行，也可以使用其他集成开发环境IDE，比如vscode）；
7. 你的目录下会有一个目录uthread，你的课程设计的代码全部在这里。你在这个目录中完成课程设计的编码；
8. 编码完成后，撰写课程设计报告，并将课程设计报告加入到你的项目中；
9. 使用git commit 命令提交你的代码修改；
10. 使用git push将你的代码推送到gitee.com端；
11. 浏览器登录gitee.com，发起一个pull request将你的项目发送给课程老师，请求合并到项目。注意：发起pull request要指定正确的分支（项目中每一位同学会有一个分支，分支的名字是你的学号）

## 提示

1. **熟悉git的操作，以及git与你的IDE的配合使用，IDE与GDB的配合使用。**
2. **由于我们要实现的线程库与Pthread类似，请仔细研究pthread线程库中语义，完全理解pthread的思想；**
3. **仔细阅读骨架代码中的所有代码，只有充分理解骨架代码中的所有代码，才能实现你的线程库。**
4. **可以使用任何IDE，当然只使用命令行也是可以的。**

## 设计工作**要**求

1. 独立完成课程设计内容及报告，并在规定的时间内提交。提交内容包括：1）课程设计报告书；2）源代码；
2. 设计报告及代码需提交到gitee上指定项目的指定分支（分支名是你的学号）；
3. 你的课程设计不得使用任何POSIX线程库或linux线程库，或者对其进行包装。