Operating Systems

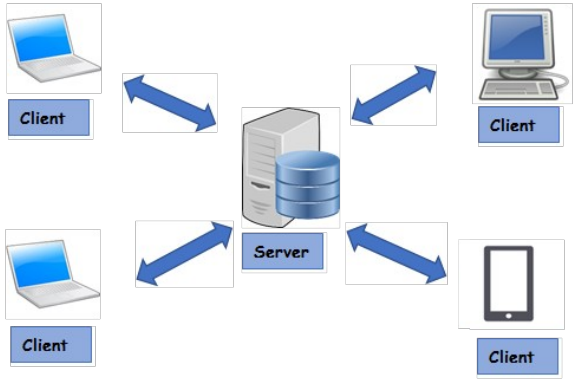
Isfahan University of Technology
Electrical and Computer Engineering Department
1401 semester

Zeinab Zali

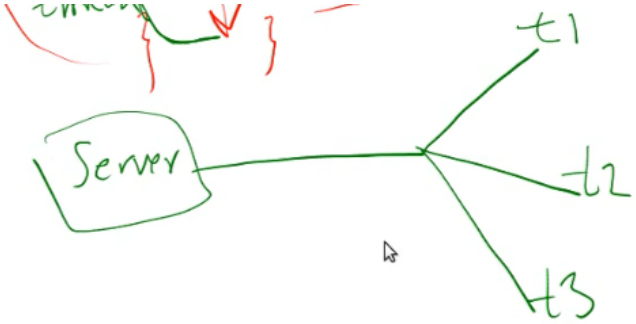
Threads Concepts and API

A Client-Server program

Assume you have a server that is responsible for responding some clients. The clients frequently ask the server to send a requested large file. How does server manage the requests from the clients? What is the problem? What is the solution?



-
چرا نیاز داریم به تردها؟ یا برنامه مالتی ترد؟
برای اینکه بفهمیم یک مسئله رو اینجا می گیم:
یک فایل سروری داریم که قراره به یکسری ریکوست های کلاینت ها جواب بده و یکسری فایل براشون بفرسته پس یک سرور داریم و یک تعدادی کلاینت به این سرور وصل هستند ینی یک تعداد کلاینت ممکنه متصل بشن هرازگاهی و ریکوستشون رو بدن و جواب رو بگیرن و برن و بعد دوباره کلاینت های دیگه وصل بشن یا همزمان وصل بشن
توی این شکل 4 تا کلاینت داریم ولی تعداد کلاینت ها خیلی زیاد هست در واقعیت و همزمان اینا ممکنه بخوان وصل بشن به یک سرور
فرض کنیم ریکوستشون یک فایل بزرگ است که انتقالش روی شبکه زمان می بره ینی transfer اش زمان می بره
حالا این سرور چجوری باید ریکوست های این کلاینت ها رو همزمان پاسخ بده؟
با استفاده از ترد می تونه این کارو بکنه ینی برای هر کدوم از کلاینت ها یک ترد باز میشه و داریم این یکی از راه حل ها است:

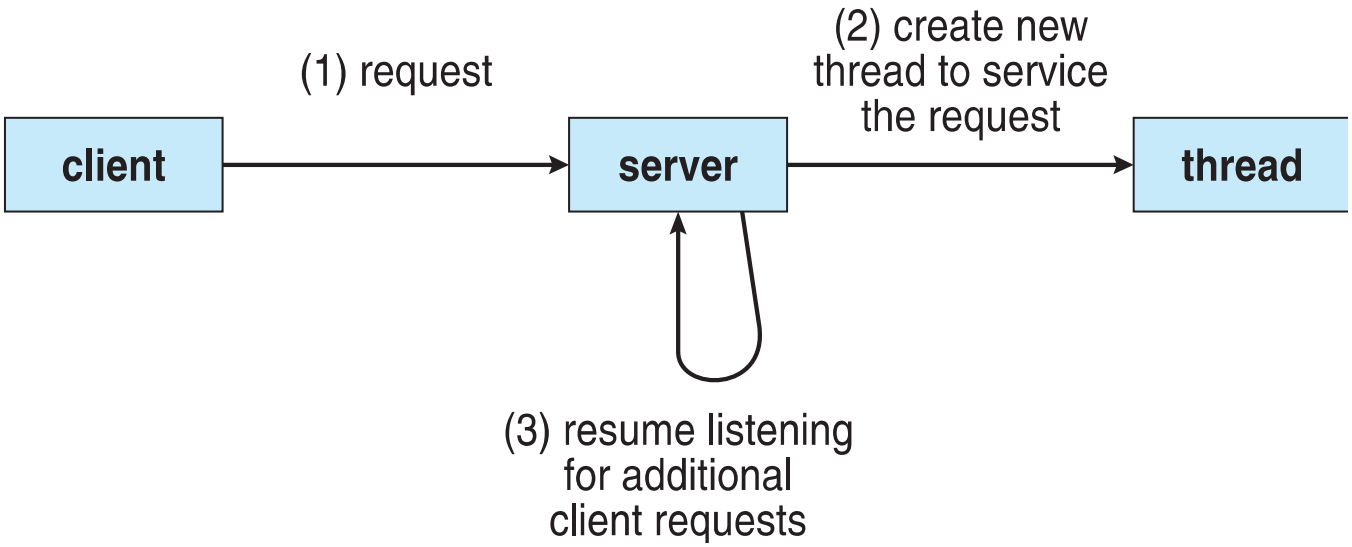


Motivation

- Most modern applications are multithreaded
- Threads run within application
- Multiple tasks with the application can be implemented by separate threads
  - Update display
  - Fetch data
  - Spell checking
  - Answer a network request
- Process creation is heavy-weight while thread creation is light-weight
- Can simplify code, increase efficiency
- Examples of multi-thread applications: basic sorting, trees, and graph algorithms, programmers who must solve contemporary CPU-intensive problems in data mining, graphics, and artificial intelligence can leverage the power of modern multicore systems by designing solutions that run in parallel.

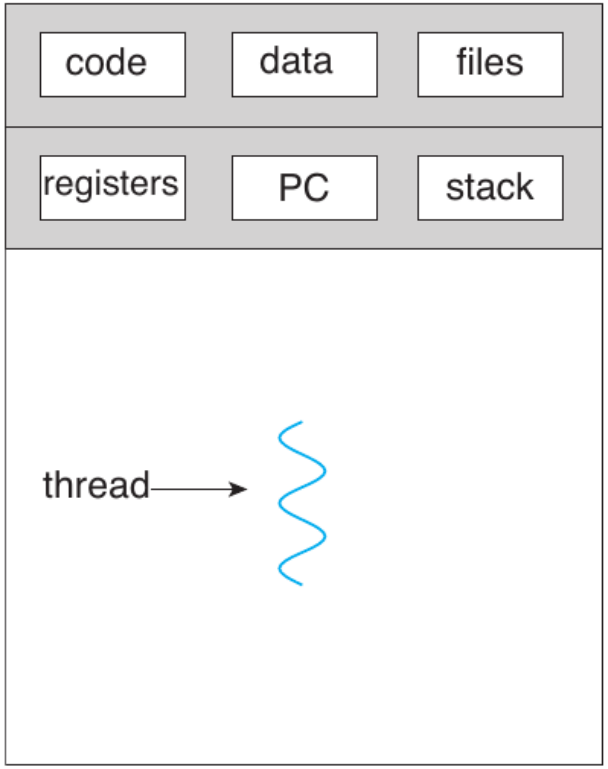
-
چرا یسری جاها نیاز داریم که برنامه مالتی ترد داشته باشیم؟
اکثر اپلیکیشن های مدرن مالتی ترد هستند : پس معمولا اپلیکیشن های interative که با یوزر کار دارن باید مالتی ترد باشند تا تمام نیازهای یوزرها رو بتونن پاسخ بدن
یک برنامه مثل برنامه ورد باید همزمان بتونه تغییراتی که اعمال میکنیم توی برنامه نمایش داده بشه ینی display داشته باشه و همزمان ممکنه Spell checking داشته باشیم (ینی اینا همزمان انجام میشه ینی زمانی که داریم توی ورد تایپ میکنیم یه برنامه دیگه هم داره چک میکنه که املاش صحیح باشه) یا برنامه ای مثل کلاینت و سرور باید ریکوست های روی شبکه رو همزمان پاسخ بدن یا داده های زیادی رو فچ کنن همزمان از جاهای مختلف --> این ها همه نیاز به یک مالتی ترد دارن
بحث اصلی اینه که اگر بخوایم اینارو با مالتی پروسس بنویسیم چه اتفاقی می افته؟ ساخت پروسس خیلی کار سنگینی است (چون برای هر پروسسی یک مموری اسپیس داشتیم) و اگر بخوایم برای Spell checking هر پروسس جدید بازکنیم یا برای اپدیت کردن display ینی داریم هی فضاهای ادرس های جدا جدا ایجاد میکنیم و همین ایجاد خودش یک کار سنگینی است ینی heavy-weight است ولی ترد light-weight است
همینطور کدنویسی تردها ساده تر است پس کارامد است توی یکسری مواقع
مثال هایی از مالتی ترد: مثل basic sorting ینی بعضی از نمونه های سورت می تونه به صورت همزمان یه قسمت هایی سورت بشه و بعدا مرج بشه یا مواردی از این دسته..

Multithreaded Server Architecture

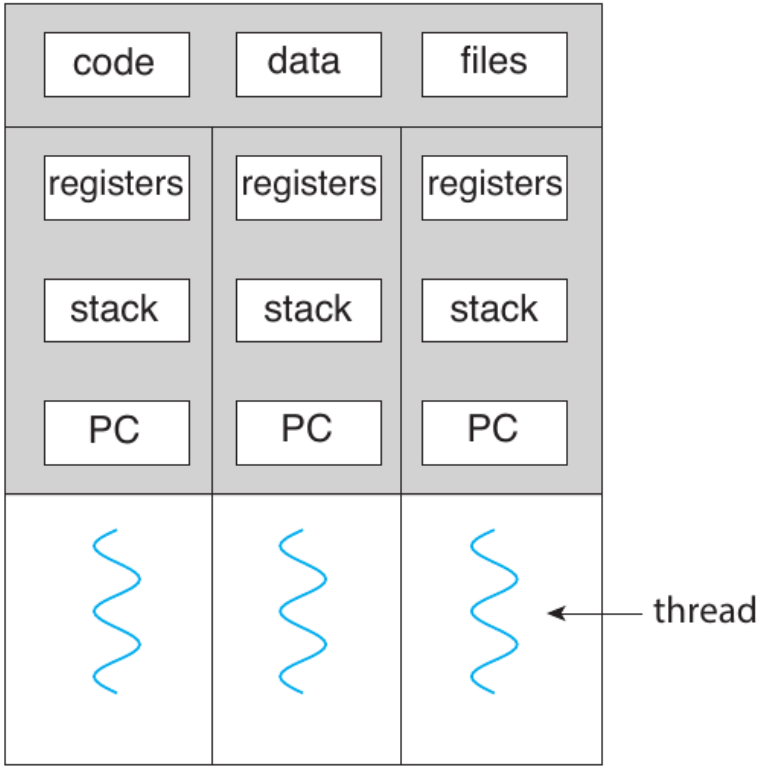


-
کلاینت به سرور میشه و سرور هر بار به ازای هر کلاینتی یک ترد جدید ایجاد میکنه
این خطی که برگشته به سمت سرور ینی این دوباره می تونه کلاینت های بعدی رو اکسپت کنه ینی باز به گوش دادن بپردازه تا کلاینت بعدی بهش وصل بشه

Single and Multithreaded Processes



single-threaded process

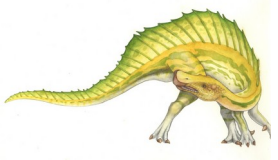


multithreaded process

-
تفاوت سینگل ترد با مالتی ترد:
سمت چپ یک پروسس داریم توی این حالت ما یک ترد داریم
ولی اگر چندتا ترد توی یک پروسس داشته باشیم (سمت راست): کد و دیتا و فایل های گلوبال که باز شدند این ها مشترکه بین همه تردها ولی به ازای هر ترد ما رجیستر مختلف، استک مختلف و pc مختلف داریم و این باعث میشه که امکان اینه که این تردهای مختلف روی core های مختلف cpu اجرا کنیم به صورت همزمان وجود داره چون هر کدوم pc , stack , registers خودشو داره که از طریقش می تونه اون core مستقلا این ترد رو اجرا کنه ولی همزمان این ترد به اطلاعات اون پروسس دسترسی داره ولی این حالت رو مالتی پروسس نداشتیم توی مالتی پروسس وقتی یک پروسس جدید ایجاد میشد کاملا یک فضای ادرس جدید و اطلاعات جدیدی و pcb جدیدی تولید می شد ولی اینجا نه همون پروسس داره به چند بخش داره تقسیم می شه

Benefits

- **Responsiveness –** may allow continued execution if part of process is blocked, especially important for user interfaces (if the time-consuming operation is performed in a separate, asynchronous thread, the application remains responsive to the user)
- **Resource Sharing –** threads share the memory and the resources of the process to which they belong by default, so easier than shared memory or message passing between processes
- **Economy –** thread creation consumes less time and memory than process creation. Additionally, context switching is typically faster between threads than between processes
- **Scalability –** a single process can take advantage of multiprocessor architectures



-
مزایای مالتی ترد:
** پاسخگویی: یکجورایی با یک تایم شیرینگی بین این تردهای مختلف ما پاسخگویی رو بالا می بریم چون هر تردی داره Responsiveness یک قسمتی رو همزمان می ده (توضیحشو بالای صفحه هم بخون)
** اشتراک گذاری منابع: توی بحث ترد این قسمت خیلی راحتر است نسبت به بحث مالتی پروسس ها
** اقتصادی: ساخت تردها زمان کمتری و مموری کمتری می بره و context switching بین تردها هم سریعتر از context switching بین پروسس هاست
** مقیاس پذیری: وقتی که یک برنامه مالتی ترد داریم ما می تونیم از اون معماری های مالتی پروسس و مالتی core مون تازه الان به صورت بهینه استفاده کنیم

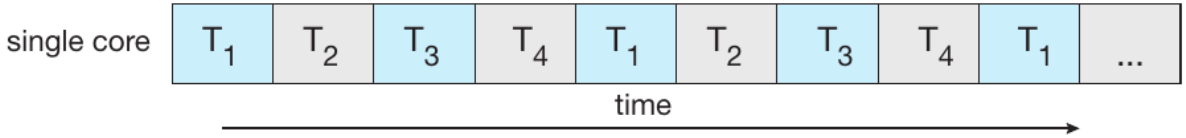
Multi-thread kernel

- Most operating system kernels are also typically multithreaded
- The command ps -ef can be used to display the kernel threads on a running Linux system
  - Examining the output of this command will show the kernel thread kthreadd (with pid = 2), which serves as the parent of all other kernel threads.

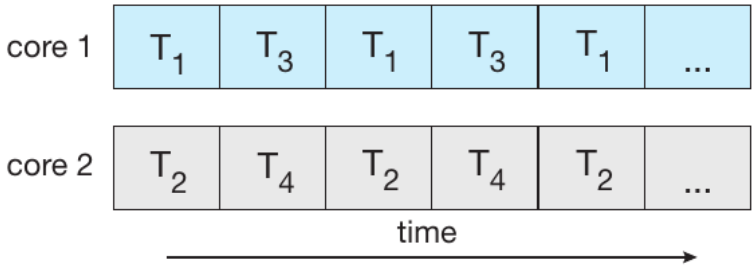
-
از طرفی خود کرنل هم معمولا کدش مالتی ترد نوشته میشه چون کرنل هم باید همزمان یکسری کارها رو انجام بده
با دستور ps -ef می توان تردها رو هم نشون داد
نکته: تردهای کرنل هم PID می گیرند و PID های متفاوت هم دارند

Concurrency vs. Parallelism

- ***Concurrency*** *supports more than one task making progress*
  - Single processor / core, scheduler providing concurrency
- ***Parallelism*** implies a system can perform more than one task simultaneously
- **Concurrent execution on single-core system:**



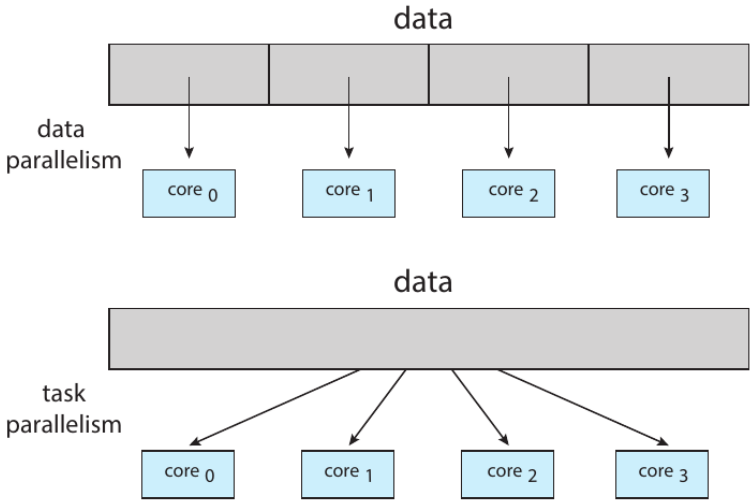
- **Parallelism on a multi-core system:**



-
توی یک سیستمی که سینگل core است یه دونه سی پی یو بیشتر نداره همینم می تونه برنامه هارو همزمان اجرا بکنه به چه صورت؟ با تایم شیرینگ طبق شکل روبرو یه دونه core توی هر بازه زمانی یکی از تردهارو اجرا می کنه پس بحث ما اینجا اجرای همزمان تردهایی است که توی پروسس های مختلف ایجاد میشن --> پس توی این حالت تایم شیرینگ داریم
ولی اگر چندتا core داشته باشیم که چندتا سی پی یو داشته باشیم توی این حالت تردها غیر ازاینکه می تونن با تایم شیرینگ همزمان اجرا بشن توی core های مختلف هم می تونن همزمان اجرا بشن مثلا core1 ترد T1 , T3 اجرا میکنه و core2 ترد T2 , T4 اجرا میکنه ولی قبلا که یک core داشتیم بین همه این 4 تا ترد باید این تایم شیرینگ انجام میشد ولی توی حالت 2 تا core یکیشون داره روی T1 , T3 تایم شیرینگ میکنه و یکی دیگشون روی T2 , T4 و این باعث میشه که T1 , T2 که روی دوتا core مختلف داره اجرا میشن امکان اینکه واقعا همزمان اجرا بشن وجود داره
پس اینکه میگیم concurrent هستند ترد ها توی حالت سینگل core توی این حالت مجازا concurrent هستند یا اصطلاحا میگیم concurrent همزمان واقعی نیستند ولی توی حالت دوتا core همزمان واقعی هستند ینی T1 ,T2 همزمان واقعی هستند ولی T1 , T3 همزمان واقعی نیستند ولی ما بهشون میگیم همزمان چون کاربر متوجه نمیشه که سی پی یو داره بین T1 , T3 سوییچ می کنه و به این حالت دوم که دوتا core داره موازی سازی یا parallelism میگیم

Multicore Programming (Cont.)

- Types of parallelism
  - **Data parallelism** – distributes subsets of the same data across multiple cores, same operation on each
    - Ex: calculating an array sum or matrix multiplication
  - **Task parallelism** – distributing threads across cores, each thread performing unique operation
    - Ex: calculating different statistical operation on the array of elements
  - **Hybrid**



-
اگر بخوایم توی یک برنامه ای از parallelism استفاده بکنیم و قصدمون اینه که سرعت اجرای این برنامه بالا بره و بهش اجازه بدیم که بتونه روی core های مختلف هم همزمان اجرا بشه چجوری این کارو بکنیم؟
دو حالت داره این کار
1- Data parallelism: یک دیتای بزرگ داریم و این دیتا رو می شکونیم به بخش های مختلف و توی این حالت تسکی که قراره روی همه دیتا انجام بشه یکسان یا مشابه و یا ربطی به همدیگه داره پس می تونیم هر قسمت دیتارو به یکی از core ها بدیم و اون عملیات روی اون دیتا رو انجام بده
2- Task parallelism: بعضی وقت ها دیتای بزرگی نداریم ولی تسک هامون زیاده در این حالت یک دیتای واحد رو به core های مختلف می دیم ولی تسک های مختلف روی هرکدوم از این core ها انجام میشه مثلا یکیشون قراره میانگین گیری رو این دیتا بکنه و یکی دیگشون مثلا جمع و... پس به این حالت میگیم Task parallelism یعنی تسک رو شکستیم به تسک های مختلف یعنی یک تسک بزرگ داریم و اینو شکستیم به تسک های کوچکتر
یه موقعی ها هست که از هر دو حالت استفاده میکنیم که بهش میگیم Hybrid یعنی هم دیتا رو می شکنیم بعد خود اون دیتای شکسته خودش به تسک های مختلف شکسته میشه یعنی همزمان می خواد تسک های مختلفی روش اجرا بشه

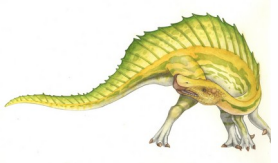
Multicore Programming challenges

- **Dividing activities:** examining applications to find areas that can be divided into separate, concurrent tasks
- **Balance:** ensure that the tasks perform equal work of equal value.
- **Data splitting:** the data accessed and manipulated by the tasks must be divided to run on separate cores
- **Data dependency:** When one task depends on data from another, programmers must ensure that the execution of the tasks is synchronized to accommodate the data dependency
- **Testing and debugging:** When a program is running in parallel on multiple cores, many different execution paths are possible making debugging difficult

-
چالش هایی که داریم اینجا چیه؟
**Dividing activities: از اون ابتدا برنامه نویس باید فکر کنه چه activities هایی رو تقسیم بکنه مثلا دیتا رو باید تقسیم بکنه یا تسک رو باید تقسیم بکنه؟ اصلا کدوم یکی از تسک ها امکان موازی سازی دارند کدومشون ندارند؟
**بعد وقتی که تسک رو تقسیم میکنیم یا دیتا رو تقسیم میکنیم یک dependency هایی وجود داره
**بعد خود این دیتارو چجوری بشکنیم به بخش های مختلفی که هر کدومش مستقلا بتونه عملیات روش انجام بشه ینی به عبارتی splitting کردن دیتا همیشه ساده نیست و گاهی هست که می بینیم نمی تونیم splitting اش بکنیم با توجه به کارکردی که از برنامه انتظار داریم
**Testing and debugging یک برنامه مالتی ترد خیلی سخته

User Threads and Kernel Threads

- **User threads** - management done by user-level threads library
- Three primary thread libraries:
  - POSIX **Pthreads**
  - Windows threads
  - Java threads
- **Kernel threads** - Supported by the Kernel
  - Examples – virtually all general purpose operating systems, including:
    - Windows
    - Solaris
    - Linux
    - Tru64 UNIX
    - Mac OS X



-
پیاده سازی تردها از لحاظ سیستم: ینی سیستم اینارو چجوری پیاده میکنه؟
ما دو نوع ترد داریم:
User threads: تردهایی که توی سطح یوزر اجرا میشن- اگر توی سطح یوزر باشن منظورمون اینه که در واقع توسط کتابخونه های ترد که توی سطح یوزر هستند دارن مدیریت میشن اینکه مثلا چجوری سوییچ بشه بینشون و اینکه چجوری اجرا بشن اینا همه مربوط به اون کتابخونه ای میشه که اون چجوری مدیریت بکنه
Kernel threads: ترد هایی که توی سطح کرنل اجرا میشن
توی تردهایی سطح یوزر مثلا شاید سوییچ کردن بین تردها مستقیما سیستم کال نداره ولی تردهای سطح کرنل سیستم کال داره ینی مثلا وقتی تولیدشون میکنیم سیستم کال داریم فراخوانی میکنیم ولی وقتی تردهایی سطح یوزر را داریم تولید میکنیم اون کتابخونه سطح یوزر است که داره تولیدش میکنه و سیستم کالی وجود نداره
ما سه تا ترد پایه داریم که توی یوزر ترد هستن:

- POSIX Pthreads
- Windows threads
- Java threads

بیشتر با posix تردها اینجا اشنا میشیم
در ادامه می خوایم بگیم این ارتباط تردهای سطح یوزر و سطح کرنل به چه صورت است؟

تحقیق

- با بررسی در هدر فایل sched.h و جستجو، تحقیق کنید آیا برای مدیریت threadها نیز ساختاری مشابه task_struct در سیستم عامل برای هر thread استفاده میشود؟ یا روش دیگری وجود دارد؟ تفاوتها را مشخص کنید
- مدل threadها در زبانهای c، جاوا و پایتون را مقایسه کنید.
- نحوه ساخت kernel thread و user thread با استفاده از system callهایی شبیهfork به چه صورت است؟

Thread Libraries

- **Thread library** provides programmer with API for creating and managing threads
- Two primary ways of implementing
  - Library entirely in user space
  - Kernel-level library supported by the OS

-
کتابخونه های ترد از لحاظ از اینکه خودشون چجوری پیاده سازی شدند می تونن به دو دسته تقسیم بشن:
1- اون هایی که به طور کامل توی یوزر اسپیس پیاده سازی شدند
2- یا کتابخونه هایی که اواس ساپورت می کنه از طریق سطح کرنل هستند

منظور از کتابخونه ترد همون api هست ک در اختیار پروگرمر قرار میگیره که بتونه باهاش ترد بسازه

Pthreads

- A POSIX standard (IEEE 1003.1c) API for thread creation and synchronization
- ***Specification***, not ***implementation***
- API specifies behavior of the thread library, implementation is up to development of the library
- Common in UNIX operating systems (Solaris, Linux, Mac OS X)

-
در این جلسه می خواهیم در مورد تردهای POSIX صحبت میکنیم که api استاندارد لینوکس هم هست و اسمش Pthreads است
api ویندوز متفاوت است
کاری نداریم که Pthreads خودش چجوری پیاده سازی شده و ما می خواهیم در مورد اینکه api چجوریه و چجوری به وسیله اون ترد بسازیم صحبت می کنیم که اصطلاحا می گیم
specification اش است و به implementation اش کاری نداریم
توی سیستم های مختلف این Pthreads خودش فانکشن هاش متفاوت می تونه پیاده سازی بشه

Pthread.h

- get the default attributes
  - Pthread_attr_init(pthread_attr_t * attr)
- create the thread
  - pthread_create(pthread_t *tid, pthread_attr_t *attr_t,
    void* thread_runner, void *thread_runner_args):
- wait for the thread to exit
  - pthread_join(pthread_t *tid, void ** thread_runner_ret_val)
- Exit thread
  - pthread_exit(void * pthread_runner_ret_val)

-
این توابع همه مربوط به فایل Pthread.h است
اولین تابعی و تابع اصلی که استفاده میشه pthread_create است و pthread_create اون تردمون رو می سازه و یک هندلری برای ترد داریم که از جنس pthread_t *tid است که این پوینترشو وقتی به pthread_create می دیم و اگر ساختن ترد موفقیت امیز این پوینتر غیر nullمی شه و بعد از اون می تونیم از طریق این به ترد دسترسی داشته باشیم و ارگومان دومش pthread_attr_t *attr_t است که یکسری فیچرهایی است که در مورد ترد داریم می سازیم پس یکسری فیچر هایی توی این هست که بعضی هاشو می تونیم دستی تنظیم کنیم ولی فعلا کاری به این نداریم و وقتی ترد می سازیم با تابع
Pthread_attr_init با مقادیر پیش فرض سیستم اون رو پرش میکنیم
نکته: از طریق pthread_attr_t *attr_t, میشه بعضی از فیچرهای ترد رو هم گرفت مثل استکش و موارد دیگه
پارامتر سوم خیلی مهمه که void *thread_runner است که این پاارامتر ادرس اون فانکشنی است که کد اصلی تردمون است و چهارمین پارامتر که void *thread_runner_args است ارگومان هایی هست که می تونیم به این فانکشن بدیم ینی ما اون تردی که می سازیم می تونه این ارگومان های ورودی رو داشته باشه
دو تابع دیگه رو گفت توی کد توضیح میده

Pthreads Example

```
#include <pthread.h>
#include <stdio.h>

int sum; /* this data is shared by the thread(s) */
void *runner(void *param); /* threads call this function */

int main(int argc, char *argv[])
{
   pthread_t tid; /* the thread identifier */
   pthread_attr_t attr; /* set of thread attributes */

   if (argc != 2) {
      fprintf(stderr,"usage: a.out <integer value>\n");
      return -1;
   }
   if (atoi(argv[1]) < 0) {
      fprintf(stderr,"%d must be >= 0\n",atoi(argv[1]));
      return -1;
   }
```

-
توی این مثال یک عدد رو به عنوان ارگومان ورودی دریافت میکنه و اعداد یک تا اون عددی که دریافت کرده رو جمع می زنه و این جمع زدن رو توی ترد انجام میده

توی برنامه مین برای ساختن ترد دو تا مقدار از دو تا type نیاز است :
pthread_t tid
pthread_attr_t attr
و این دوتا اینجا تعریف شده و ادرسشون به تابع pthread_create داده شده

نکته: حتما به type ها توجه بکنین که درست استفاده بشه و استفاده بکنیم

```
   /* get the default attributes */
   pthread_attr_init(&attr);
   /* create the thread */
   pthread_create(&tid,&attr,runner,argv[1]);
   /* wait for the thread to exit */
   pthread_join(tid,NULL);

   printf("sum = %d\n",sum);
}

/* The thread will begin control in this function */
void *runner(void *param)
{
   int i, upper = atoi(param);
   sum = 0;

   for (i = 1; i <= upper; i++)
      sum += i;

   pthread_exit(0);
}
```

فانکشن ترد که اینجا void *runner است و همیشه ارگومان ورودی این تابع باید void * باشه و داخل اون تابع همون کدی که می خوایم رو می نویسیم و اینجا مثلا یک جمع زده میشه
و از ارگومان ورودی این تابع استفاده کرده برای اینکه تا اون عدد رو جمع بکنه
نکته: اگر هر برنامه ای دیگه توی این تابع داشتیم و هر ورودی رو داشت اون ورودی باید حتما void * باشه ولی خودمون باید اینجا توی بدنه تابع باید cast اش کنیم به هر type که مدنظرمون است مثلا اینجا param رو با تابع atoi به int تبدیل کرده
اگر بیشتر از یک ارگومان ورودی برای یک تابع ترد نیاز داشتیم چه کار کنیم ؟ کافیه از همین void * استفاده بکنیم به این صورت که می تونیم یک استراکچری تعریف بکنیم که فیلد هاش همون پارامترهایی باشه که می خوایم به عنوان ارگومان برای تابع بفرستیم و بعد یک متغییری از نوع استراکچر تعریف بکنیم و ادرسش رو بفرستیم برای این تابع runner و بعد از cast کردن اون ها می تونیم از فیلداشون توی بدنه تابع استفاده بکنیم
از تابع pthread_exit(0) هم استفاده میشه برای اینکه معمولا status اون فانکشن برگردونده بشه که ایا با موفقیت انجام شده یا نه که اینجا به طور کلی صفر را ارسال کرده که به معنی موفقیت امیز بودن است
ارگومان چهارم تابع pthread_create همون عددی است که میخوایم تا اون عدد جمع بکنیم و عدد یک را براش می فرستیم ینی argv[1]
تابع pthread_join : این تابع مترادف wait توی پروسسس ها است و اگر این تابع رو فراخوانی نکنیم ترد بالا create میشه و بعد از اون بقیه برنامه مین اجرا میشه و ترد هم همزمان شروع به اجرا شدن میکنه (مثلا توی این برنامه که خیلی کوتاه است این برنامه اصلی ممکنه تموم بشه در صورتی که اون ترد هنوز تموم نشده باشه اجراش) پس pthread_join رو اینجا فراخوانی میکنیم و این باعث میشه که برنامه سر این تابع صبر بکنه تا تردی که هندلرش ارسال شده به این تابع تموم بشه و این که تموم بشه این ریترن میشه و ارگومان دوم هم برای برگرداندن همون Status تردی است که تمام شده و اینجا بهش نیاز نداشتیم واسه همین null گذاشتیم
نکته: اینجا برای نتیجه ترد از یک متغییر گلوبال استفاده کرده ینی Sum

Pthreads Code for Joining 10 Threads

```
#define NUM_THREADS 10

/* an array of threads to be joined upon */
pthread_t workers[NUM_THREADS];

for (int i = 0; i < NUM_THREADS; i++)
   pthread_join(workers[i], NULL);
```

-
نکته: اگر یکی از تردها دچار یک مشکلی بشه کل برنامه بسته میشه و مشکل پیدا میکنه بخاطر اینکه تردها فضای اردسشون و کدشون مشترکه با پروسس اصلی

Implicit Threading

- Growing in popularity as numbers of threads increase, program correctness more difficult with explicit threads
- Creation and management of threads done by compilers and run-time libraries rather than programmers
- some methods explored
  - Thread Pools
  - OpenMP
  - Fork-Join
  - Grand Central Dispatch
  - Intel Threading Building Blocks

-
Implicit Threading : به صورت غیر مستقیم تردها ساخته میشن ینی ما نیازهایی داریم و با یک مدل کدنویسی امکان این که ترد از کد ما ایجاد بشه فراهم میشه ینی به صورت مشخص اون تردها رو نساختیم
کی این Implicit Threading رو داریم؟ وقتی که تعداد تردهایی که نیاز داریم بره بالا یا یک منظور ویژه ای داریم برای ساخت تردها
پس مدیریت ساخت این تردها از طریق کامپایلر و کتابخونه هایی که در اختیار برنامه نویس قرار میگیره انجام میشه
مثال:
ترد پول و روش هایی دیگری که پایینش گفته و بیشتر در مورد ترد پول حرف می زنیم

Client-Server Example

Remember our own example

What happens if the number of alive threads exceeds the maximum number of concurrent threads that the system can support?

How can we prevent this issue?

-
برای اینکه مشخص کنیم ترد پول چجوری کار میکنه و چی هست اینارو میگیم:
روی مثال برنامه کلاینت و سرور می ریم جلو که ما یک سرور داریم و تعداد زیادی کلاینت ممکنه همزمان بهش وصل بشن - این رو هم میدونیم که هر سیستمی یک ماکزیمم تردی رو می تونه بسازه و هر چقدر میخواد نمیتونه بسازه
مسئله اصلی اینجا اینه که اگر ما اینجا ترد هم استفاده کردیم برای مدیریت پاسخگویی به کلاینت ها ولی تعداد کلاینت ها اینقدر زیاد شد و همزمان تردهاشون در حال اجرا بودند و هنوز پاسخشون کامل داده نشده بود که کلاینت جدیدی که اومد وارد بشه دیگه تردی برای اون نداشتیم ینی وقتی خواستیم براش با pthrea creat براش ترد بسازیم بهمون خطا میده و میگه دیگه توی سیستم نمی تونم ترد جدید باز کنم و این توی run time این خطا اتفاق می افته و اگر این خطا رو مدیریت نکنیم اتصال این کلاینت جدید از بین می ره ینی سرور کامل miss اش کرده
بعضی خطاها هنگام اجرا به وجود میان و اگر ما بتونیم توی کدنویسی اون حالت هارو در نظر بگیریم که اگر اون خطا رخ داد چی بشه این بهترین حالت میشه ینی پیش بینی بکنیم چه خطاهایی ممکنه اتفاق بیوفته و یه جوری سعی کنیم اینو هندل بکنیم و از رخ دادنش جلوگیری بکنیم یا مدیریتش بکنیم
و اینجا مسئله همین است که ما چجوری از این اتفاقی که پیش میاد که یک کلاینت جدید اتصالش از بین میره چچوری جلوگیری بکنیم؟

Thread Pools

- Create a number of threads in a pool where they await work
- Advantages:
  - Usually slightly faster to service a request with an existing thread than create a new thread
  - Allows the number of threads in the application(s) to be bound to the size of the pool
  - Separating task to be performed from mechanics of creating task allows different strategies for running task
    - i.e.Tasks could be scheduled to run periodically
- Windows API supports thread pools:

```
DWORD WINAPI PoolFunction(AVOID Param) {
    /*
    * this function runs as a separate thread.
    */
}
```

-
جواب: ترد پول است ینی یک استخری از تردهاست
ترد پول: یک تعدادی ترد رو توی یک جایی می سازه که اصطلاحا میگیم پول و اون ابتدایی که ساخته میشه ممکنه هیچ تسکی به تردها داده نشده باشه
ترد پول این امکان رو برای ما فراهم میکنه جایی که پولی از تردها می سازیم یک تعدادی ترد توی سیستم ایجاد بشه ولی هنوز تسکی بهشون داده نشده ولی این تردها الان اماده این هستند که تسک بهشون داده بشه و وقتی بهشون تسک داده شد اجرا بشن
در واقع مثل این که فضای ادرس، هندلر و ... و در واقع دیتا استراکچرهایی ک برای مدیریت اون ترد نیاز داریم براشون درنظر گرفته میشه ولی هنوز تسکی مشخص نیست که چه تسکی باید توشون اجرا بشه
همچین ترد پولی رو میتونیم زمان افلاین ایجادش بکنیم
اانلاین به زمانی گفته که میشه سرور داره کانکشن هایی این کلاینت هارو قبول میکنه و شروع میکنه بهشون جواب داده می شه حالت انلاین ولی قبل از اینکه وارد این بشه که بخواد کلاینت هارو قبول بکنه میشه حالت افلاین و زمان افلاین یکبار ایجاد میشه
مزایای ترد پول:
1- چون ساختن ترد توی حالت افلاین است پس باعث میشه ما سریعتر بتونیم پاسخ بدیم به ریکوست ها
2- اون محدودیت هایی که روی تعداد تردها داریم رو این میتونه برامون مدیریت بکنه و از miss شدن یکسری تسک ها جلوگیری بکنه و ما اینجا از طریق پول روی باندی که روی تعداد تردها بوده داریم چون می تونیم خودمون بگیم که اون پولی که داریم می سازیم سازش چقدر باشه و ما اون سایز رو چجوری انتخاب میکنیم که از ماکزیمم تردهای سیستم بالاتر نزنه
3- ساختن اون ترد رو از جدا کردیم از اساین کردن تسک بهش ینی اون تسکی که قراره داده بشه از مکانیزم ایجاد اون تردها جدا شده و این تسک ها میتونن هر جوری که ما خواستیم اسکجول بشن برای اون تردها
برای جاهای مختلف کتابخونه های مختلفی برای ترد پول داریم مثلا توی ویندوز API برای ترد پول امکان
همچین امکاناتی رو به ما میده و توی صفحه روبرو است:

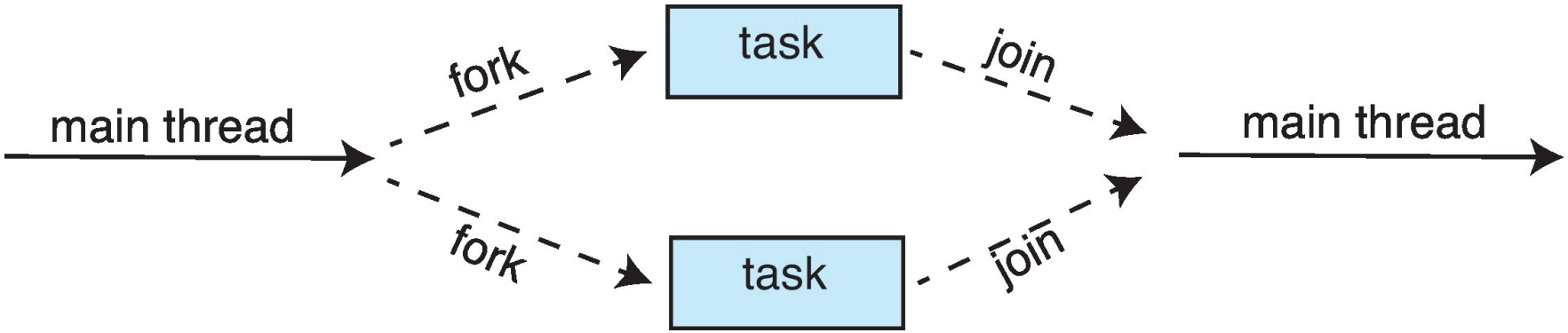
Sample thread pool API

```
void first_task() {…}
void second_task() {…}
void third_task() {…}
main(){
    // Create a thread pool.
    pool tp(2);
    //Add some tasks to the pool.
    tp.schedule(&first_task);
    tp.schedule(&second_task);
    tp.schedule(&third_task);
}
```

-
2 حداکثر تعداد تردها است توی پول

Fork-Join Parallelism

- Multiple threads (tasks) are **forked**, and then **joined**.



-
موازی سازی با فورک-جوین:
شبیه این است که ما ترد بسازیم با pthread و بعد جوین بذاریم روش
ینی یک main thread داریم و بتونه فورک کنه از این تردهای مختلفی رو و بعد بتونیم روش جوین بکنیم

Fork-Join Parallelism

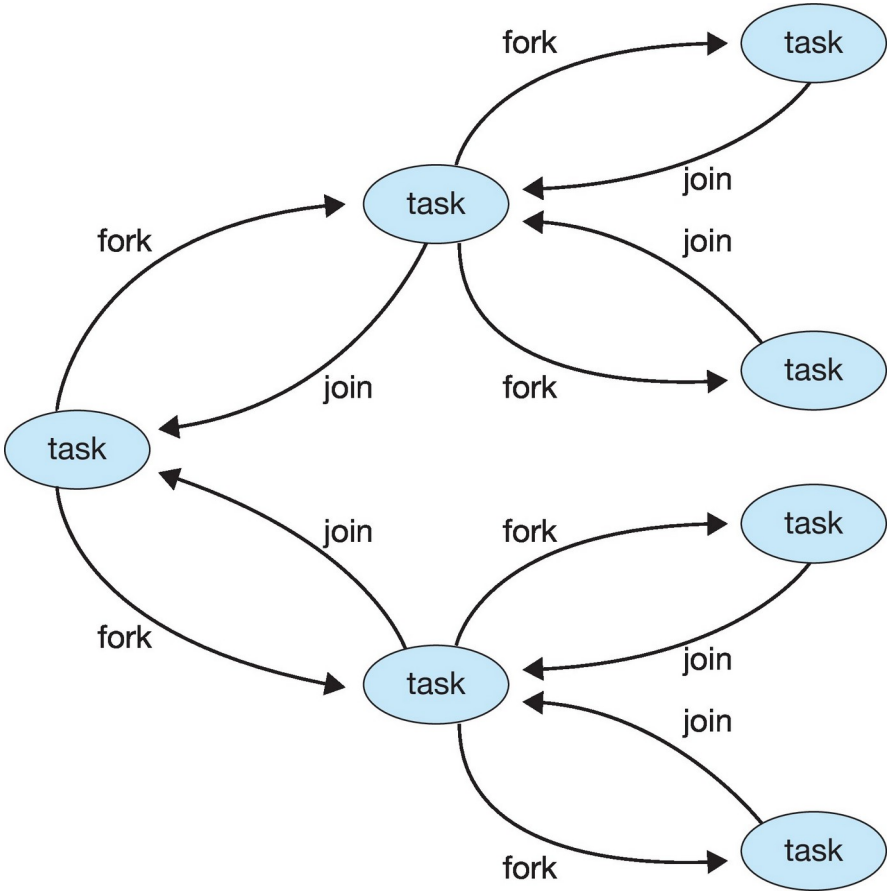
- General algorithm for fork-join strategy:

```
Task(problem)
   if problem is small enough
     solve the problem directly
   else
     subtask1 = fork(new Task(subset of problem)
     subtask2 = fork(new Task(subset of problem)

     result1 = join(subtask1)
     result2 = join(subtask2)

     return combined results
```

Fork-Join Parallelism



این کار معمولا به چه کاری میاد؟ چجور تسک هایی رو ما می تونیم اینجوی حلش بکنیم؟
جاهایی که مسئله با تقسیم و غلبه حل میشه ینی یک مسئله رو بشکنیم به مسئله های ریزتر و از اونا جواب میگیریم و نتیجشون برمیگیرده به اون برنامه اصلی

OpenMP

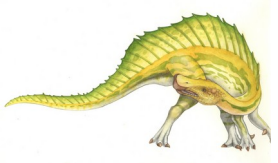
- Set of compiler directives and an API for C, C++, FORTRAN
  provides support for parallel programming
- Identifies **parallel regions** – blocks of code that can run in parallel

```
#pragma omp parallel
```

Create as many threads as there are cores

```
#pragma omp parallel for
for(i=0;i<N;i++) {
    c[i] = a[i] + b[i];
}
```

Run for loop in parallel



-
این یکسری directives به ما میده که directives های کامپایلری هستند که اولش # میاریم
مثلا توی مثال پایین ازش خواسته که اون حلقه ای که نوشته شده رو به صورت موازی بیاد اجرا بکنه
برای کدهای مختلف directives مختلفی برای openMP داریم که استفاده بشه که بتونه کامپایلر اون قسمت رو بهینه تر با توجه به کدی که هست بشکنه و ازش تردهای مختلفی ایجاد بکنه که بتونه موازی اجرا بشه

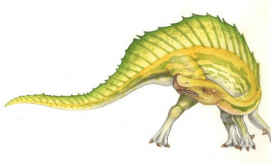
Threading Issues

- Semantics of **fork()** and **exec()** system calls
- Signal handling
  - Synchronous and asynchronous
- Thread cancellation of target thread
  - Asynchronous or deferred

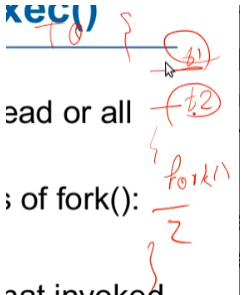
-
مسائل و چالش هایی که باید توی مسئله تردینگ بهش دقت کنیم:
اگر fork() and exec() توی برناممون داشتیم و برنامه مالتی ترد باشه در مجموع چه اتفاقی خواهد افتاد توی اجرا؟
مسئله بعدی Signal handling و Thread cancellation of target thread

Semantics of fork() and exec()

- Does `fork()` duplicate only the calling thread or all threads?
  - Some UNIX systems have two versions of fork():
    - one that duplicates all threads
    - one that duplicates only the thread that invoked the fork() system call.
- `exec()` usually works as normal – replace the running process including all threads
  - So when exec() is called immediately after forking, forking only the calling thread is sufficient



-
اگر برنامه‌مون مالتی ترد باشه وقتی فورک فراخوانی میشه کل تردها میان تکرار میشن یا فقط تردی که اون فورک رو فراخوانی کرده تکرار میشه؟
جواب این بستگی داره به این که این API ها توی سیستم های مختلف چجوری پیاده سازی شده اند
مثلا برای لینوکس میتونه این دو حالت رو داشته باشه:
1- همه تردها رو تکرار کنه
2- فقط تردی که فراخوانی کرده تکرار بشه مثلا برای مثال پایین فقط T0 میشه در حالی که T1 , T2 درون T0 است ولی چون T0 فراخوانی کرده فقط T0 میشه ولی برای بخش اولی سه تاش میشه
بحث exec: اگر exec رو فراخوانی کنیم ایا همه تردهای اون پروسسی که exec روش فراخوانی شده جایگزین میشه با exec یا نه ؟
جوابش اینه که همش جایگزین میشه و این اینو نشون میده که ما اگر exec رو فراخوانی کنیم تمام تردها جایگزین میشه با اون برنامه ای که توی exec گفتیم ینی اگر ما قبلش یک برنامه مالتی ترد رو فورک کرده باشیم و بعد بخوایم exec رو فراخوانی کنیم خب مفید هم نیست برامون که بخوایم تردهای مختلف رو هم فورک کرده باشیم چون میخوایم بلافاصله بعد از فورک exec اجرا بشه



Signal Handling

- **Signals** are used in UNIX systems to notify a process that a particular event has occurred.
- A **signal handler** is used to process signals
  1. Signal is generated by particular event
  2. Signal is delivered to a process
  3. Signal is handled by one of two signal handlers:
     1. default
     2. user-defined
- Every signal has **default handler** that kernel runs when handling signal
  - **User-defined signal handler** can override default
  - For single-threaded, signal delivered to process

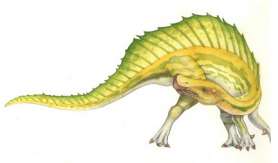
-
Signal Handling: برای حالتی پیش میاد که ما بخوایم یک پروسس رو اگاه بکنیم از رخداد یک اتفاقی و اون پروسس وقتی که اون اتفاق افتاد بخواد در جواب یک کاری انجام بده
پس سیگنال دوتا بحث داره: 1- کی سیگنال ارسال بشه 2- چطور هندل بشه
از یک Signal Handling برای پروسس اون سیگنال استفاده میشه
وقتی که یک سیگنالی ارسال میشه اگر یک برنامه ای چندتا ترد داشته باشه ایا همه تردهای اون برنامه اون سیگنال رو دریافت میکنن یا فقط یکیشون میتونه دریافت کنه؟ که اینا جواب های متفاوتی توی سیستم های مختلف میتونه داشته باشه

Signal Handling (Cont.)

- Where should a signal be delivered for multi-threaded?
  - Deliver the signal to the thread to which the signal applies
  - Deliver the signal to every thread in the process
  - Deliver the signal to certain threads in the process
  - Assign a specific thread to receive all signals for the processes

kill(pid_t, signal)

pthread_kill(thread_t, signal)



-
و ممکنه مثل لینوکس این امکان وجود داشته باشه که ما مستقیما سیگنال رو برای یک ترد مشخصی بفرستیم
وقتی میخوایم سیگنال رو بفرستیم از سیستم کال kill استفاده میکنیم و با ارگومان اولش مشخص میکنیم که برای کی این رو بفرستیم ینی برای چه پروسسی بفرستیم و pid اون پروسس رو میدیم و هرجایی توی برنامه این فراخوانی را داشته باشیم برای پروسسی که pid اش رو دادیم این سیگنالی که مشخص کردیم ارسال میشه حالا اون برنامه باید Signal Handling داشته باشه که اگر این سیگنال اومد چه کاری انجام بده
یک تابع دیگر: pthread_kill است که از طریقش می تونیم به یک ترد مشخص سیگنال بفرستیم

Thread Cancellation

- Terminating a thread before it has finished
  - Suppose multiple threads searching for a record in a database and one of them find it
  - a web page loads using several threads—each image is loaded in a separate thread. When a user presses the stop button on the browser, all threads loading the page are canceled.
- Thread to be canceled is **target thread**
- Two general approaches:
  - **Asynchronous cancellation** terminates the target thread immediately
  - **Deferred cancellation** allows the target thread to periodically check if it should be canceled

-
یکی از سیگنال ها سیگنال کنسل کردن است
کنسل کردن ترد:
مثلا توی زمانی که چندتا ترد دارند با هم اجرا میشن و هر کدوم سرچ خودش رو داره انجام میده و اگر یکی از این تردها موفق بشه و اون چیزی که میخوایم رو پیداش بکنه دیگه نیازی نیست که بقیه تردها کارشون رو ادامه بدن پس اینجا میتونیم همه تردها رو ببندیم و این خروجی که اون ترد به ما داده رو استفاده بکنیم
یا مثلا توی مرورگرها که بالا گفته شده
در این حالت یک سیگنال کنسل ارسال میشه از اون والد تردها به تردها و دو حالت برای هندل کردن این سیگنال کنسل هست:
1- یا Asynchronous کنسل میکنیم ینی به محض اینکه این پیام کنسل ارسال شد برای تردهای مقصد، تردهای مقصد بسته میشن که به این حالت Asynchronous میگیم
2- یا Deferred است ینی اون ترد مقصد میتونه و اجازه داشته باشه که به محض این که دریافت کرد نبنده بلکه امکان این که سیگنال کنسل رو بگیره داشته باشه و بعد از اینکه یک سری عملیاتی رو انجام داد اون ترد خودش رو ببنده ینی ما امکانی توی هر ترد داشته باشیم که چک کنیم که ایا سیگنال کنسل رو گرفتیم یا نه، نه اینکه خود به خود سیستم همیشه وقتی کنسل ارسال بشه اونو ببنده

Thread Cancellation (Cont.)

- Invoking thread cancellation requests cancellation, but actual cancellation depends on thread state

Mode	State	Type
Off	Disabled	–
Deferred	Enabled	Deferred
Asynchronous	Enabled	Asynchronous

- If thread has cancellation disabled, cancellation remains pending until thread enables it
- Default type is deferred
  - Cancellation only occurs when thread reaches **cancellation point**
    - Ex: **`pthread_testcancel()`**
    - `Ex: read function`
- On Linux systems, thread cancellation is handled through signals

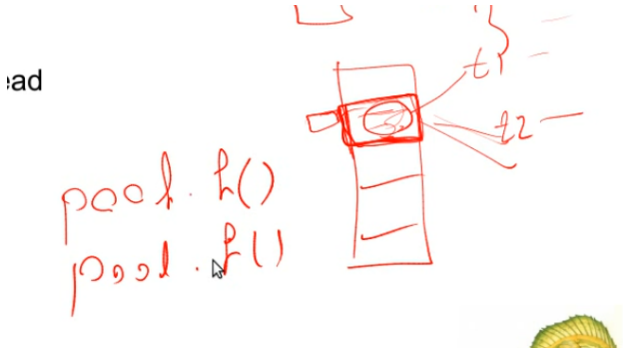
-
off ینی اگر سیگنال رو گرفتیم هیچ کاری باهاش نداریم
Asynchronous که مشخصه که اگر گرفت فوری می بنده خودش رو و کار دیگه ای انجام نمیده
و Deferred اینه که ما یک تابعی باید داشته باشیم که هی چک کنیم که کنسل رو گرفتیم یا نه و
اگر رسیدم به خط pthread_testcancel کنسل بشیم پس توی حالت Deferred وقتی که به
یک نقطه مشخص که خودمون توی کد ترد مشخص کردیم رسید می تونه در صورت دریافت
سیگنال کنسل ببنده خودش رو --> این نوع برخورد مثل تابع read می مونه
توی سیستم های لینوکس این کنسل کردن از طریق سیگنال ها پیاده سازی میشه

Thread-Local Storage

- **Thread-local storage** (**TLS**) allows each thread to have its own copy of data
- Useful when you do not have control over the thread creation process (i.e., when using a thread pool)
- Different from local variables
  - Local variables visible only during single function invocation
  - TLS visible across function invocations
- Similar to `static` data
  - TLS is unique to each thread

-
ممکنه که ما نیاز داشته باشیم که برای ترد یک دیتایی رو بتونیم نگه داریم ینی یک قسمت مموری وجود داشته باشه که همیشه توی این تردمون قابل دسترسی باشه --> این بیشتر توی ترد پول بدردمون میخوره
به این مموری TLS میگیم ینی این TLS بین تمام تردهایی که مربوط به این یک بلاک ترد پول است مشترک است و اگر مقدارش توی اجرای یکی از این تردها عوض شد ترد بعدی که میخواد همین جا اجرا بشه اون مقدار جدیده رو داره

پس بین تمام تردهایی که مربوط به بلاک این ترد پول است مشترکه و این با مقدار لوکال فرق داره



Linux Threads

- Linux refers to them as ***tasks*** rather than ***threads***
- Thread creation is done through **`clone()`** system call
- **`clone()`** allows a child task to share the address space of the parent task (process)
  - Flags control behavior

flag	meaning
CLONE_FS	File-system information is shared.
CLONE_VM	The same memory space is shared.
CLONE_SIGHAND	Signal handlers are shared.
CLONE_FILES	The set of open files is shared.

- **`struct task_struct`** points to process data structures (shared or unique)

```
getconf GNU_LIBPTHREAD_VERSION
```

-
توی سیستم ها مخصوصا سیستم لینوکس این تردهای سطح یوزر یا تردهای سطح کرنل چجوری پیاده سازی شدند؟
دستور فورکی که استفاده می کردیم یک دستور عام تری داره به اسم clone که با clone میتونیم همون فورک هم اجرا بکنیم ولی clone اپشن های دیگری رو هم داره
وقتی که از فورک استفاده می کردیم به محض فورک کردن یک پروسسی عین پروسس والد کپی میشد ولی clone این اپشن رو به ما میده که بهش بگیم اینطوری باشه یا نه ینی پروسس والد رو کامل کپیش بکنه یا نه

End of Chapter 4