



# **Operating Systems**

## **Thread Libraries & Signal handling**

Seyyed Ahmad Javadi

[sajavadi@aut.ac.ir](mailto:sajavadi@aut.ac.ir)

Fall 2021

# Thread Libraries

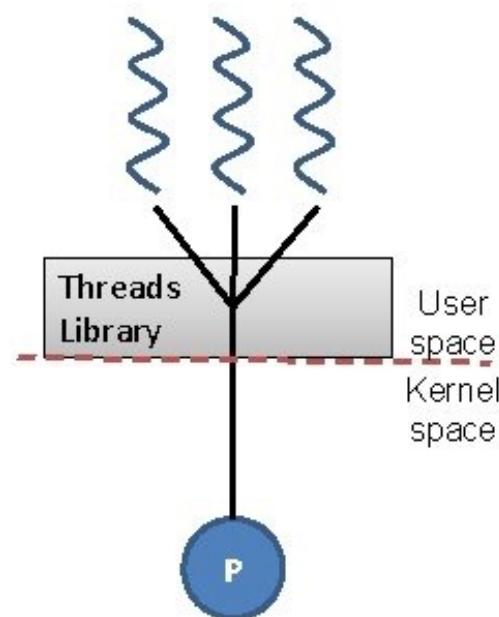
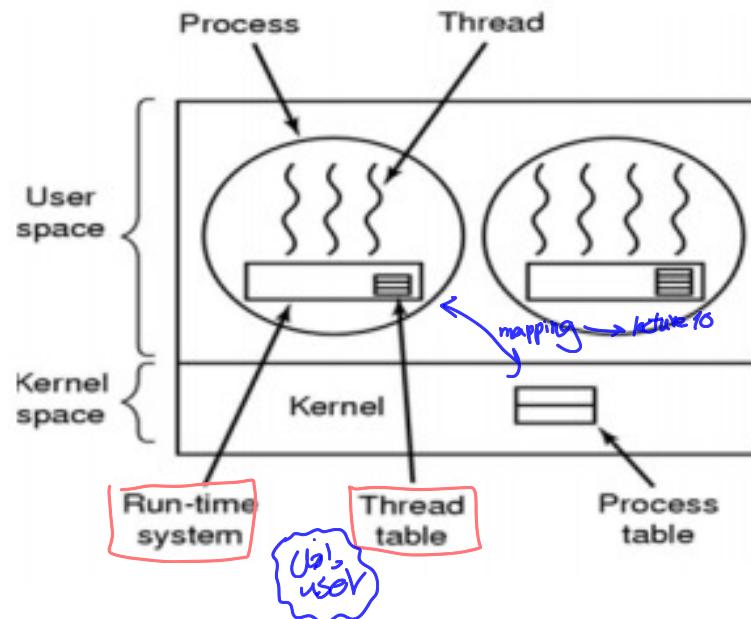
---

- Thread library provides programmer with API for creating and managing threads.
- Two primary ways of implementing
  - User-space library → *با کمک سایر کتابخانه ها*
  - Kernel-level library → *با کمک از سیستم عامل*

[https://www.tutorialspoint.com/operating\\_system/os\\_multi\\_threading.htm](https://www.tutorialspoint.com/operating_system/os_multi_threading.htm)

# Library entirely in user space

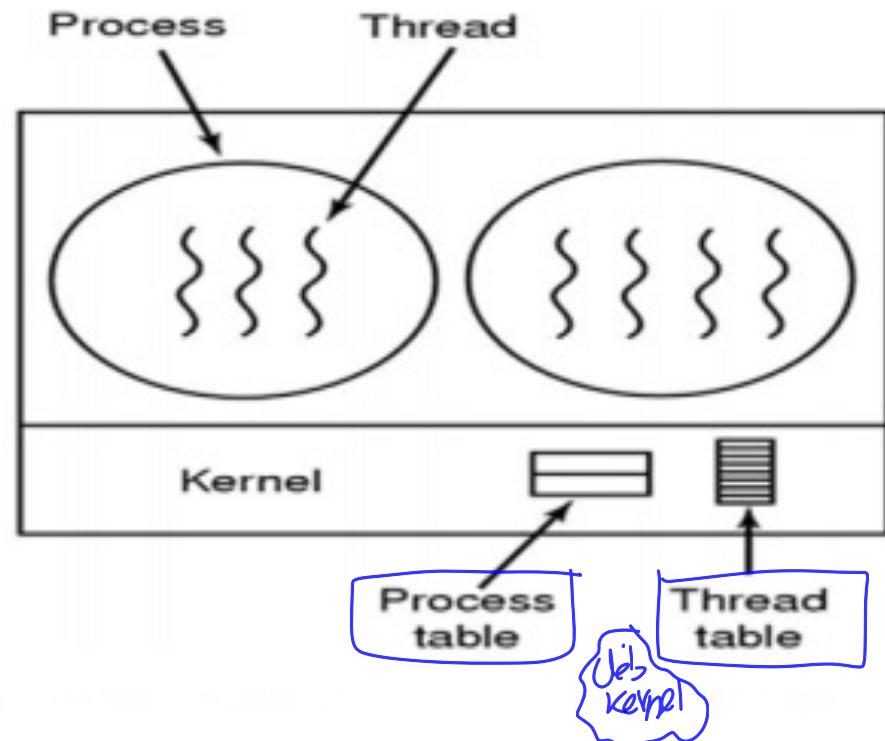
- All code and data structures for the library **exist in user space.**  
از تابع رسانی کردن از فایل library در فضای کاربری این کار را می‌دانم
- Invoking a function in the library **results in a local function call** in user space and **not a system call.**



# Kernel-level library supported directly by the OS

- Code and data structures for the library exist in kernel space.
- Invoking a function in the API for the library typically **results in a system call** to the kernel.

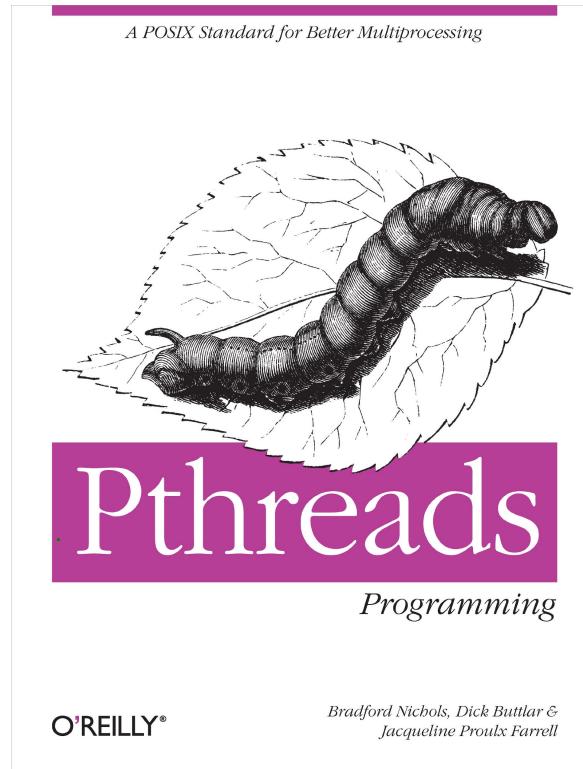
lik system "in  
wro wle")



# PThreads

- May be provided either as user-level or kernel-level.
- A POSIX standard API for thread creation and synchronization.

Thread (جای خود را در سیستم  
user space (سیستم) را در میان چند جای خود را در  
kernel space (کرنل) را در میان چند جای خود را در  
کرنل کرد، روکنل space (کرنل) را در میان چند جای خود را در



# Pthreads (cont.)

*(cont)*

- ***Specification, not implementation.***
- API specifies behavior of the thread library
  - Implementation is up to development of the library.
- Common in UNIX operating systems
  - Linux & Mac OS X

## Optional reading:

<https://users.cs.cf.ac.uk/Dave.Marshall/C/node30.html>

<https://stackoverflow.com/questions/43219214/where-is-the-value-of-the-current-stack-pointer-register-stored-before-context-s>

# Pthreads Example

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

#include <stdlib.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 */
    /* set the default attributes of the thread */
    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);
}
```

Handwritten notes and annotations:

- A blue box highlights "attributes" in `pthread_attr_init(&attr);`. Above it, a green arrow points to "الإيات من" (Attributes of) and "يُعرف بالجذار" (Known as).
- A yellow box highlights "pthread\_create(&tid, &attr, runner, argv[1]);". Above it, a green arrow points to "برمجة" (Programming) and "المفهوم" (The concept). Below it, arrows point to "thread id" (thread identifier), "attributes", "function", and "parameters".
- A yellow box highlights "pthread\_join(tid, NULL);". Above it, a green arrow points to "برمجة" (Programming) and "المفهوم" (The concept).
- A yellow box highlights "printf("sum = %d\n", sum);". Below it, a green arrow points to "to thread" (to thread) and "sum new value" (new value of sum). To the right, a green arrow points to "sum new value" (new value of sum) and "runner" (runner).
- A blue box highlights "sum" in "printf("sum = %d\n", sum);". Above it, a green arrow points to "أو سائدة" (Or master).
- A blue box highlights "runner" in "printf("sum = %d\n", sum);". Above it, a green arrow points to "نحو دفعات" (Like loops) and "الطبقة الأولى" (First level).
- A blue box highlights "argv[1]" in "pthread\_create(&tid, &attr, runner, argv[1]);". Above it, a green arrow points to "أمثلة على دفعات" (Examples of loops) and "أمثلة على طبقات" (Examples of levels).
- A blue box highlights "main" in "int main(int argc, char \*argv[])".
- A blue box highlights "argc" in "int main(int argc, char \*argv[])".
- A blue box highlights "argv" in "int main(int argc, char \*argv[])".



# Pthreads Example (Cont.)

```
/* The thread will execute in this function */
void *runner(void *param)
{
    int i, upper = atoi(param);
    sum = 0;
    *global*
    for (i = 1; i <= upper; i++)
        sum += i;

    pthread_exit(0); → آرخه های آخر بود
}                                         exit (-1) ب
```

Combination of  
pthread and  
fork

# 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);
```

thread پر join کی جو اور

کند و تاریخ کریم خواہی

نہیں

join کی مالیت اور قبائل خیلی اجنبی



کوئی thread کو join کر کر جو ایام  
مرد تاریخ کرست اس سے جویں را ایام  
صدمت. پھر اینکے دونوں درجہ خیلی خوب تسلیک کر جوں کم کرنا میتوں اور for بالا استفادہ کرنے

join  
با استفاده از  
دریم صبر مردمی  
bothread  
از



# Threading Issues

---

- Semantics of **fork()** and **exec()** system calls

والسُّنْدِيَّةُ مُسْلِمٌ لِّتَمَّ  
copy مُسْلِمٌ  
وَلِلْأَنَّ كَمْ  
مُسْلِمٌ thread?

- Signal handling
  - Synchronous and asynchronous



# Semantics of fork() and exec()

- Does fork() duplicate only the calling thread or all threads?

- Some UNIXes have **two versions of fork**



One that **duplicates all threads**



Another that **duplicates only the thread that invoked the fork()**

→ replace

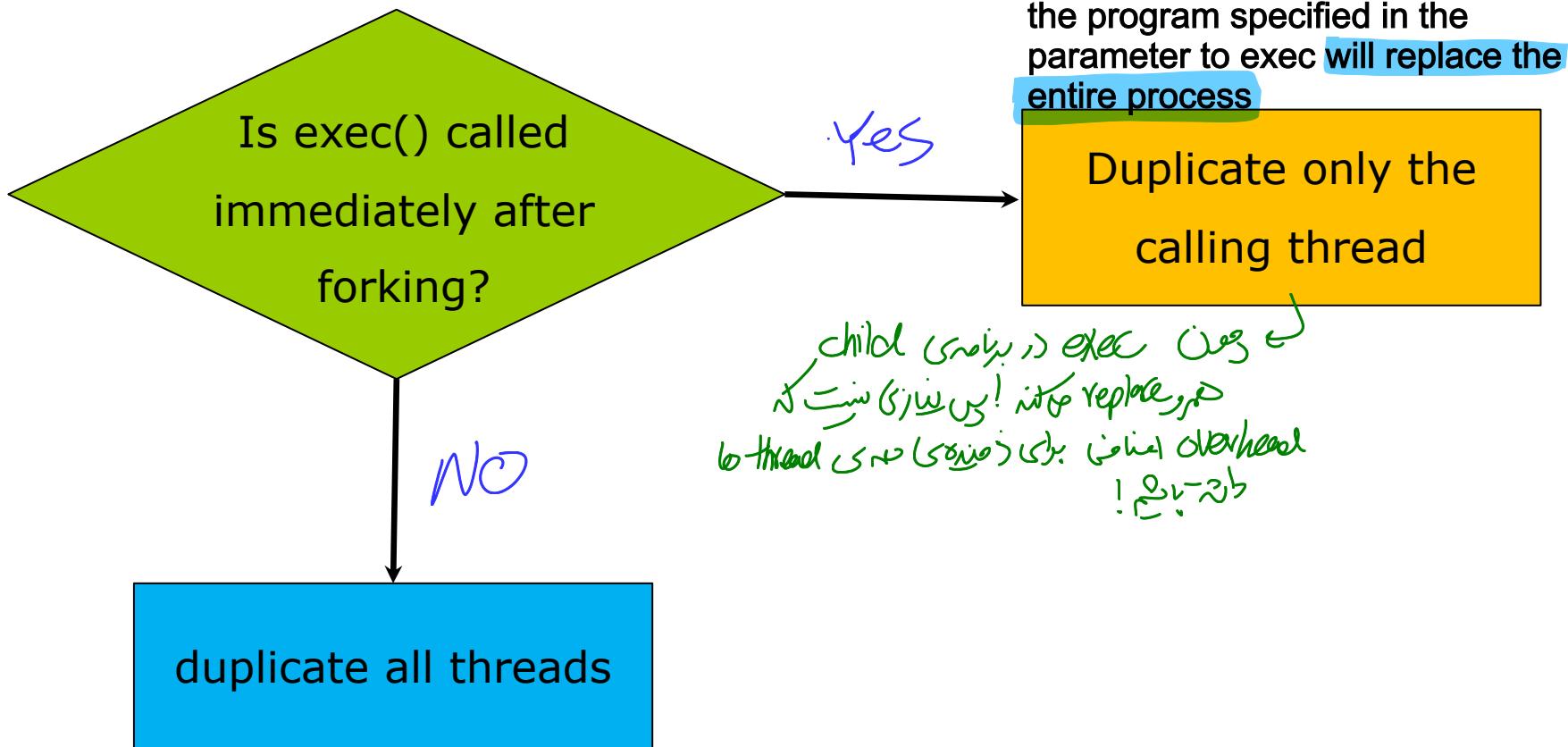
- **exec()** usually works as normal

- Replace the running process including all threads.

! info about exec چیزی که exec دارد  
! info replace چیزی که replace می‌کند  
! info about multi-thread چیزی که متا-تread دارد

# Which Version of Fork() to use?

Depends on the application.



# Which Version of Fork() to use?

---

## ■ If exec() is called immediately after forking

- Duplicating only the calling thread is appropriate.
- Since the program specified in the parameters to exec() will replace the process.

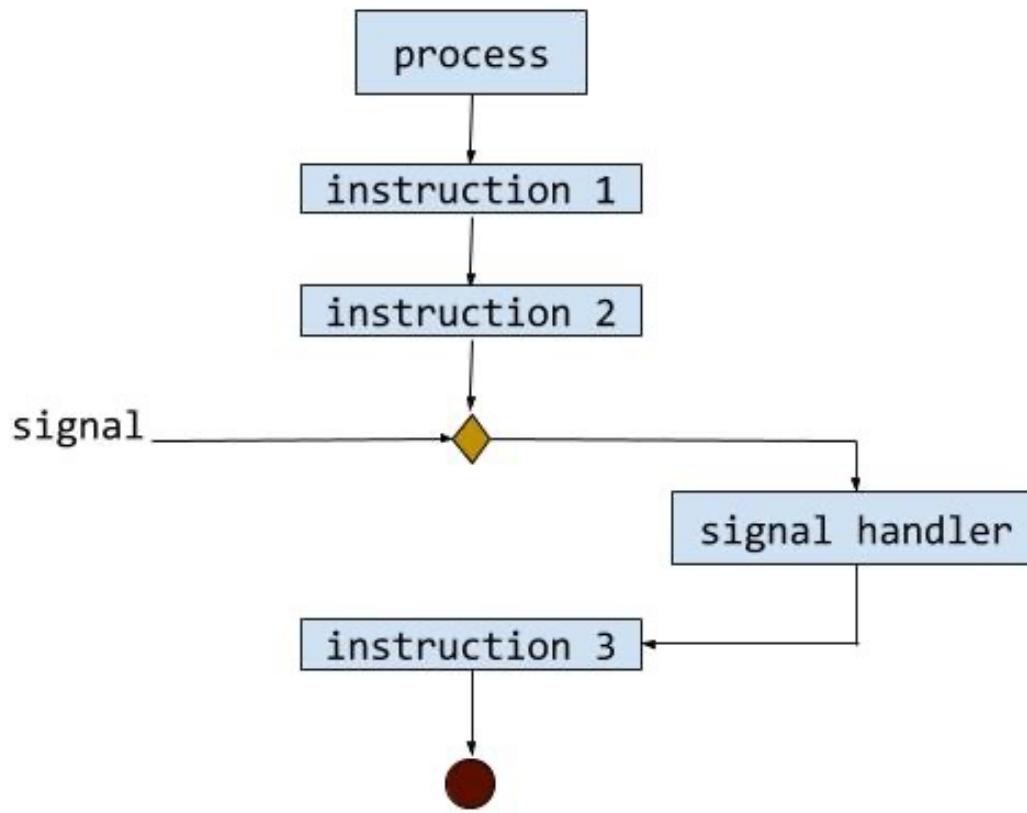
## ■ If the child process does not call exec() after forking

- The **child process should duplicate all threads.**

لـ thread (وـ)  
لـ child (وـ)  
! لـ child (وـ) > need (وـ)

# Signal Handling

- **Signals** are used in UNIX systems to notify a process that a particular event has occurred.



exception و خطا هایی

# Signal Handling

interrupt و divide by zero  
signal → خطا هایی که باید پردازش شوند

- 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**

→ خطا هایی که Ctrl+C  
(default) است و terminate

2. **user-defined**

→ خطا هایی که بزرگ و کوچک است



# Two Types of Signals

---

- **Synchronous**
- **Asynchronous**

# Synchronous Signals

---

- When a signal is generated by an event internal to a running process.
- Example:
  - illegal memory access
  - divide by 0
- Synchronous signals are delivered **to the same process** that performed the operation that caused the signal
  - that is the reason they are considered synchronous

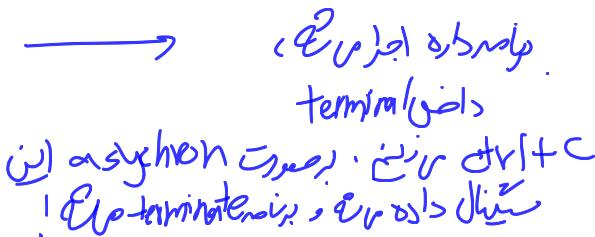
میزان سازمان اجرای برنامه  
نایابی این اتفاق را درین  
سیستم می‌داند  
صدای خالق

# Asynchronous Signals

- Signal is *generated by an event external* to a running process, that process receives the signal asynchronously.

■

- Example: terminating a process with specific keystrokes

- <control><C> → 

- Typically, **an asynchronous signal is sent to another process.**

# Signal Handling (Cont.)

---

- Every signal has default-handler that kernel runs when handling it
  - Some signals are simply ignored
    - ▶ such as changing the size of a window
  - Others are handled by terminating the program.
    - ▶ such as an illegal memory access
- User-defined signal handler can override default.
- For single-threaded, signal delivered to process.



# Signal Handling (Cont.)

*For process! and ctrl+c is*

---

- Where should a signal be delivered for multi-threaded?  
*options to*
  - ① • Deliver the signal to the thread to which the signal applies
  - ② • Deliver the signal to every thread in the process
  - ③ • Deliver the signal to certain threads in the process
  - ④ • Assign a specific thread to receive all signals for the process
- Which one should be used?
  - **Depends on the type of signal generated.**



# Signal Handling (cont.)

- **Synchronous signals** need to be delivered to the thread causing the signal and not to other threads in the process.

پردازش کننده!

- However, the situation with **asynchronous signals** is not as clear.

- Some asynchronous signals should be sent to all threads
    - ▶ Such as a signal that terminates a process (e.g., <control><C>)

نهایت پردازش کننده!  
نهایت پردازش کننده! & to thread

# Signal Handling (cont.)

---

- Most multithreaded versions of UNIX allow a thread to specify which signals it will accept and which it will block.
  - Therefore, in some cases, an asynchronous signal may be delivered only to those threads that are not blocking it.
- However, a signal is typically delivered only to the first thread found that is not blocking it.
  - Because signals need to be handled only once



# Functions for Delivering Signals

- The standard UNIX function

`kill(pid_t pid, int signal)`

*process id n°*

- Specifies the process to which a particular signal is to be delivered.

- POSIX Pthreads function

`pthread_kill(pthread_t tid, int signal)`

*thread id n°  
tid n°*

- Allows a signal to be delivered to a specified thread (tid)