



Operating Systems

Thread Libraries & Signal handling

Seyyed Ahmad Javadi

sajavadi@aut.ac.ir

Fall 2022

Thread Libraries

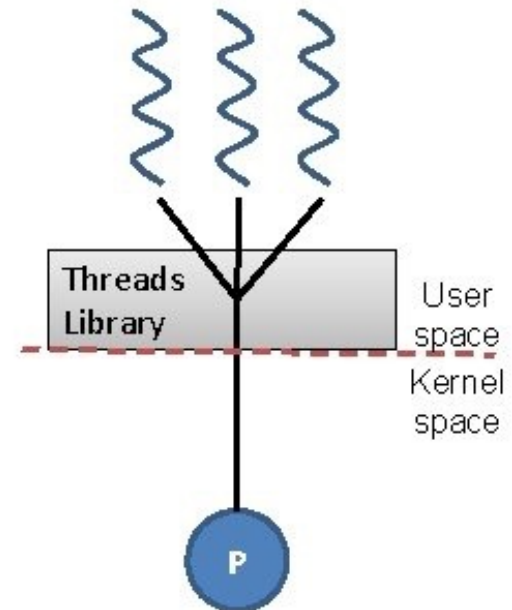
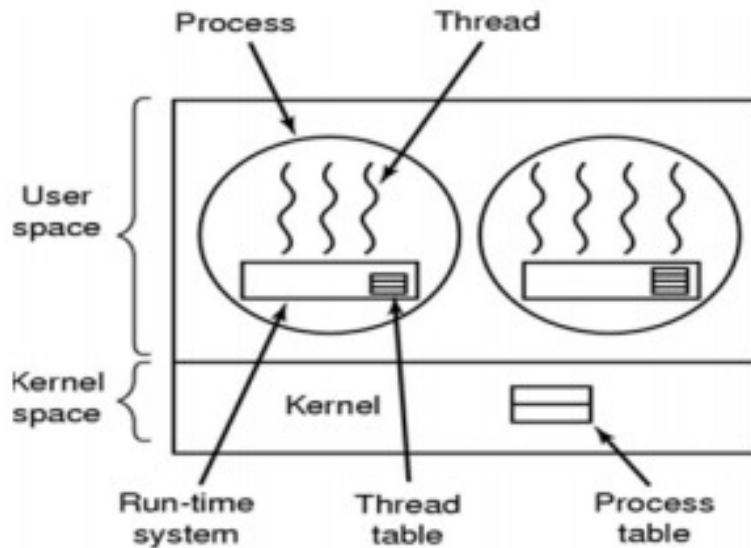
- **Thread library** provides programmer with API for creating and managing threads.
 - Abstract Programming Interface (API)
- Two primary ways of implementing
 - User-space library
 - Kernel-level library

https://www.tutorialspoint.com/operating_system/os_multi_threading.htm



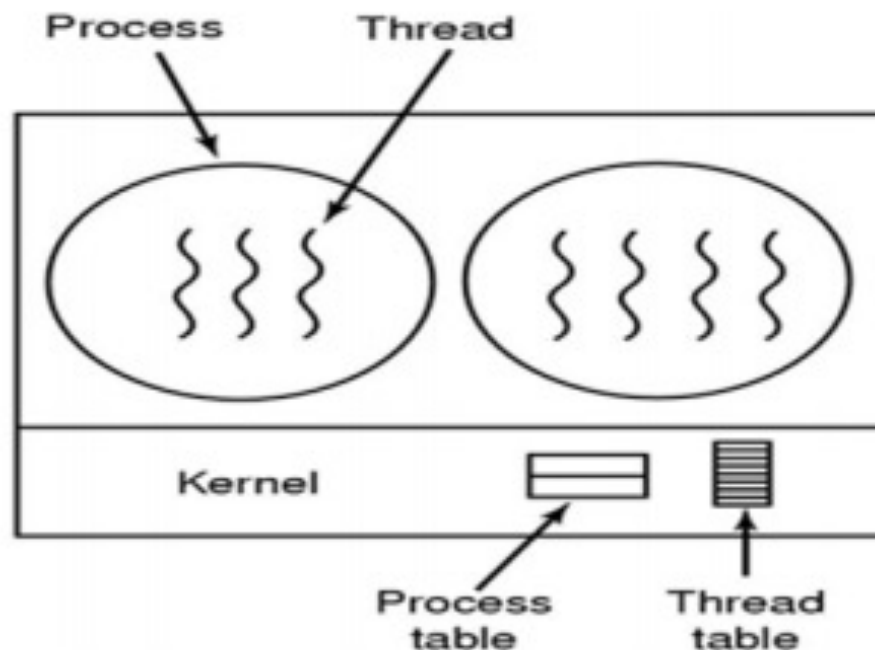
User-Space Library

- All code and data structures for the library *exist in user space*.
- Invoking a function in the library *results in a local function call* in user space and *not a system call*.



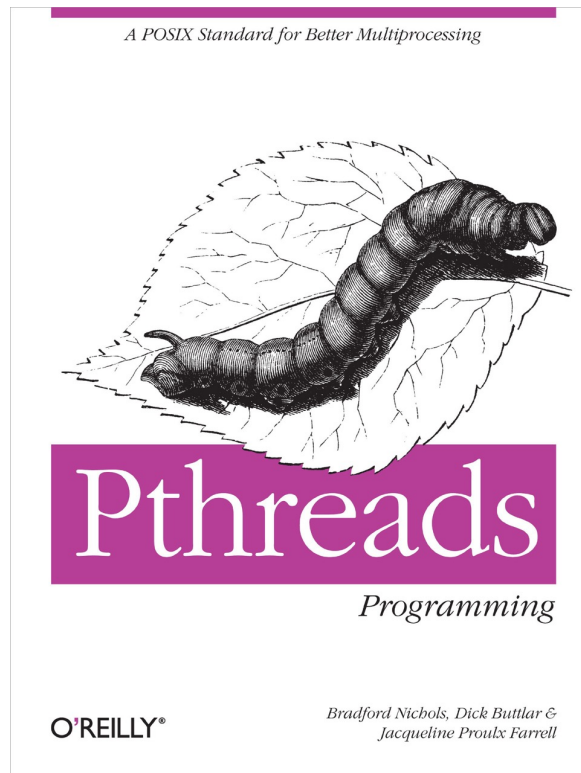
Kernel-Level Library

- 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.



Pthreads

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



Pthreads (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);
}
```



Pthreads Example (Cont.)

```
/* The thread will execute 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);
}
```



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);
```



Threading Issues

- Semantics of **fork()** and **exec()** system calls
- Signal handling
 - Synchronous and asynchronous



Semantics of fork() and exec()

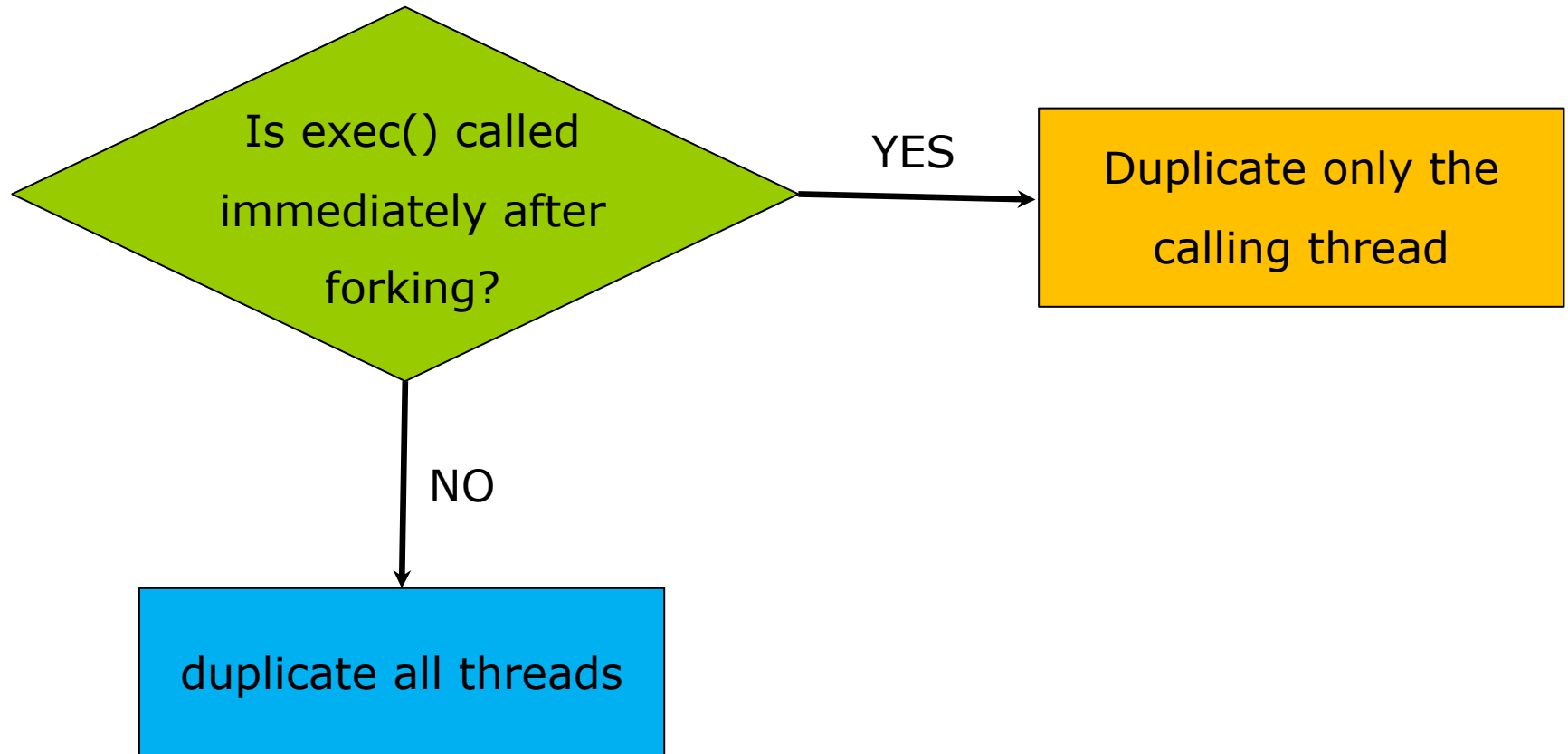
- Does fork() duplicate only the calling thread or all threads?
 - Some UNIXes have **two versions of fork**
 - ▶ One that duplicates all threads
 - ▶ Another that duplicates only the thread that invoked the fork()

- exec() usually works as normal
 - Replace the running process including all threads.



Which Version of Fork() to use?

Depends on the application.



Which Version of Fork() to use?

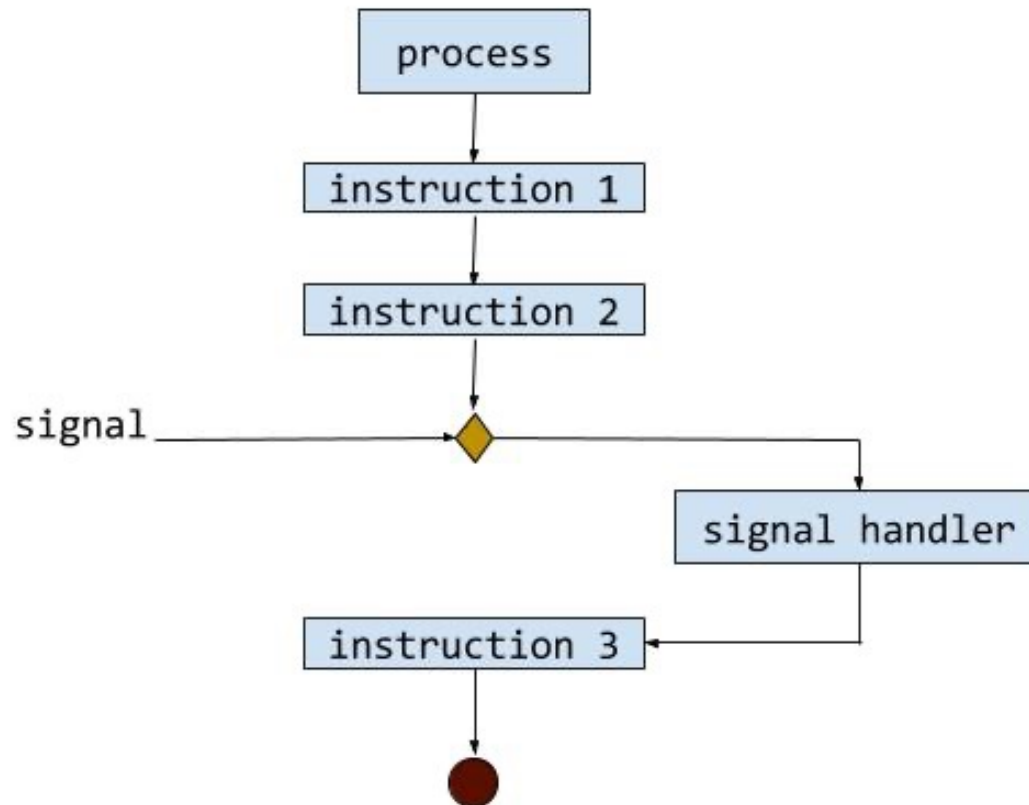
- **If exec() is called immediately after forking**
 - Duplicating only the calling thread is appropriate.
 - Since the program specified in the parameters to exec() will replace the process.

- **If the child process does not call exec() after forking**
 - The **child process should duplicate all threads.**



Signal Handling

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



Signal Handling

- 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**



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



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.)

- 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 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.

- 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>)



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)`

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

- POSIX Pthreads function

`pthread_kill(pthread_t tid, int signal)`

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