CSCI315 – Operating Systems Design

Department of Computer Science
Bucknell University

Threads, Multi-threads, and Processes

This set of notes is based on notes from the textbook authors, as well as L. Felipe Perrone, Joshua Stough, and other instructors.
Xiannong Meng, Fall 2021.

Ch 4.1 – 4.4

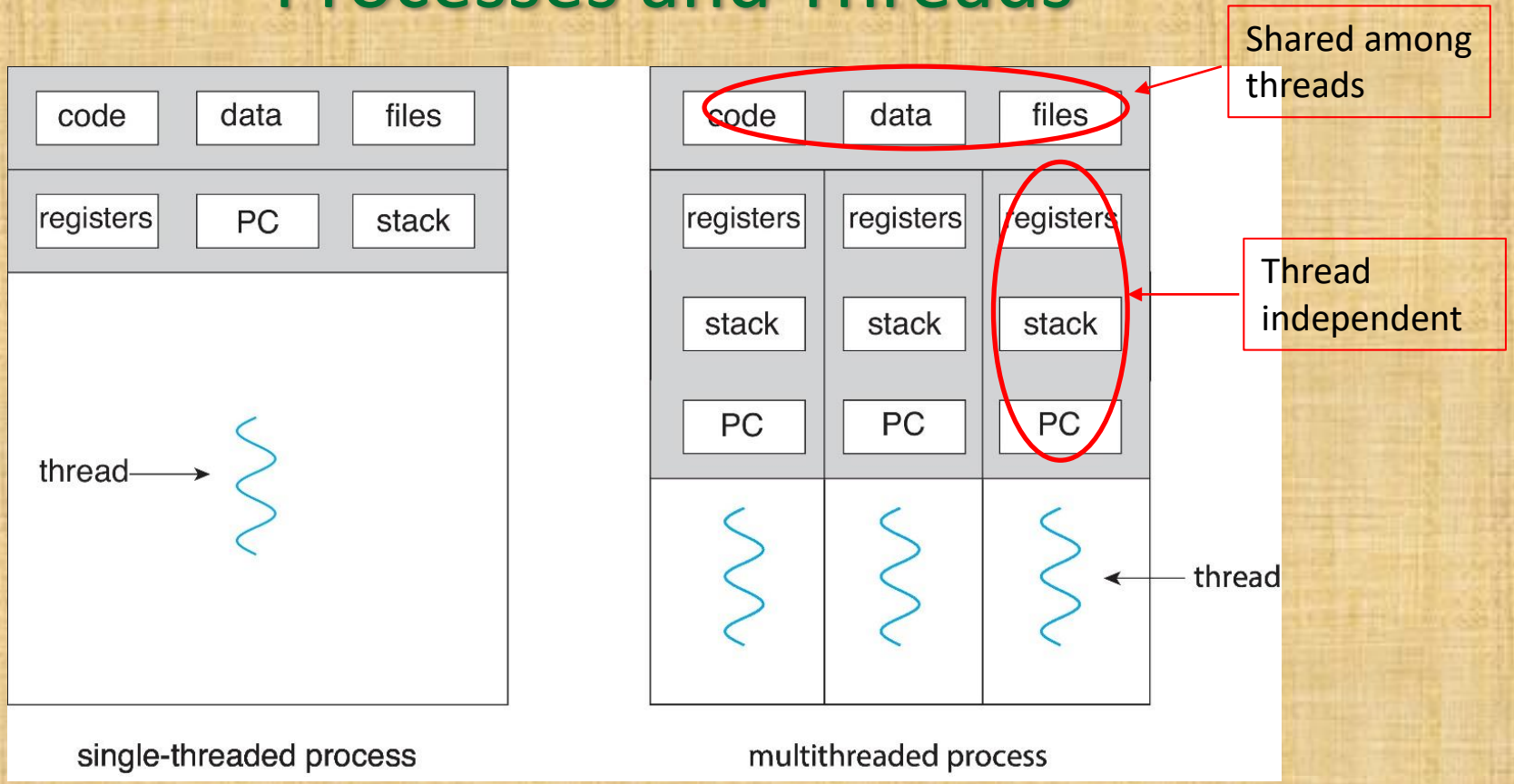
Now that we have seen how **pthreads** work in the Linux system, let’s look back at the bigger picture of threads and their relation with processes.

Some Common Questions

- Pthread is a **specification** that meets the POSIX (Portable Operating Systems Interface) standard.
- How a thread library is implemented varies and we don't need to worry about it at this level, though we will discuss some high level concepts.
- There are other popular thread libraries, e.g., Windows Threads and Java Threads.
- Kernel threads vs user threads.
  - A kernel thread is managed by the OS directly, a user thread is managed by a user program (e.g., in a library).
  - The pthread library we use on the Linux system is a collection of API; each pthread is a user thread.

https://stackoverflow.com/questions/8639150/is-pthread-library-actually-a-user-thread-solution

Processes and Threads



Advantages of Threading

- **Responsiveness:** multiple threads can be executed in parallel (in multi-core machines)
- **Resource sharing:** multiple threads have access to the same data, sharing made easier
- **Economy:** the overhead in creating and managing threads is smaller
- **Scalability:** more processors (or cores), more threads running in parallel

Challenges in Parallel and Multithreaded Programming

- **Identifying tasks**
- **Load balance**
- **Data splitting**
- **Data dependency**
- **Testing and debugging**

Multithreading Models

User threads

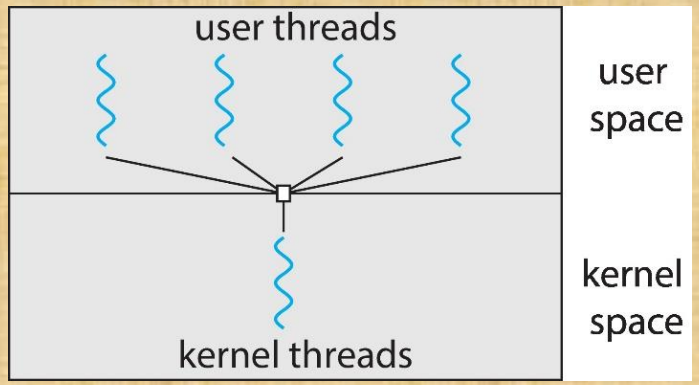
- Managed by a user library
- Runs at user level

Kernel threads

Managed directly by the operating system, thus called “kernel threads”

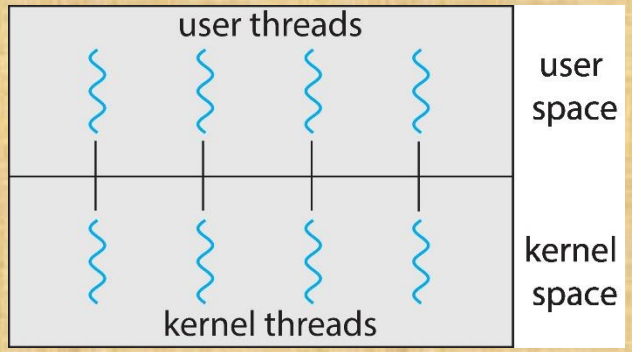
Many-to-One

- Many user-level threads mapped to single kernel thread
- One user thread blocking causes all to block
- Multiple threads may not run in parallel on multi-core system because only one may be in kernel at a time
- Few systems currently use this model
- Examples:
  - **Solaris Green Threads**
  - **GNU Portable Threads**



One-to-One

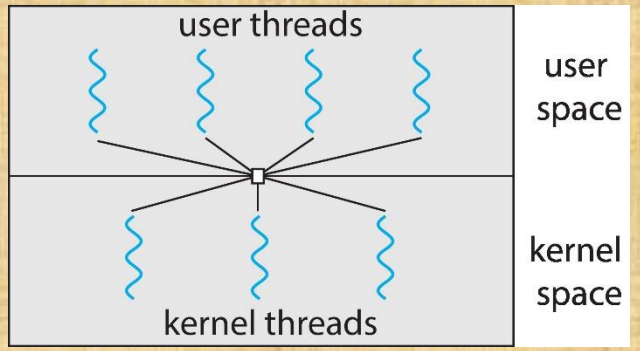
- Each user-level thread maps to kernel thread
- Creating a user-level thread creates a kernel thread
- More concurrency than many-to-one
- Number of threads per process sometimes restricted due to overhead, imagining that your program tries to create 100s of threads…
- Examples
  - **Windows**
  - **Linux**



Many-to-Many Model

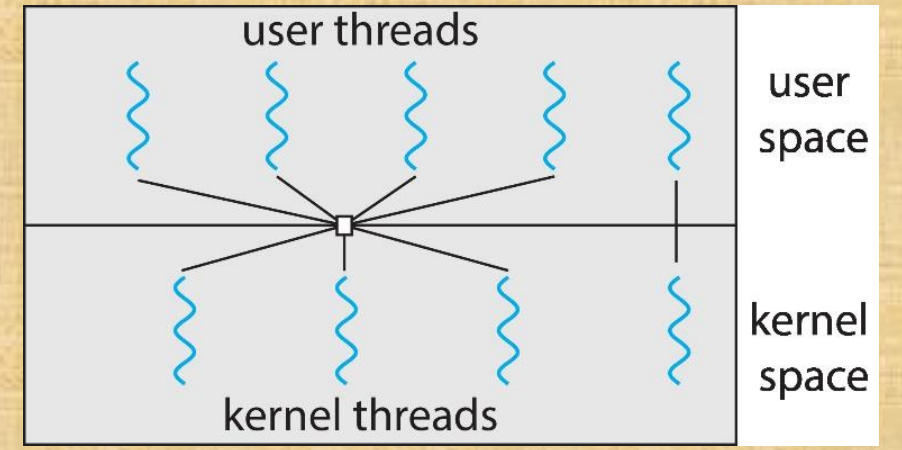
- Allows many user level threads to be mapped to many kernel threads
- Allows the operating system to create a sufficient number of kernel threads
- Windows with the *ThreadFiber* package
- Otherwise not very common

https://docs.microsoft.com/en-us/windows/win32/procthread/fibers

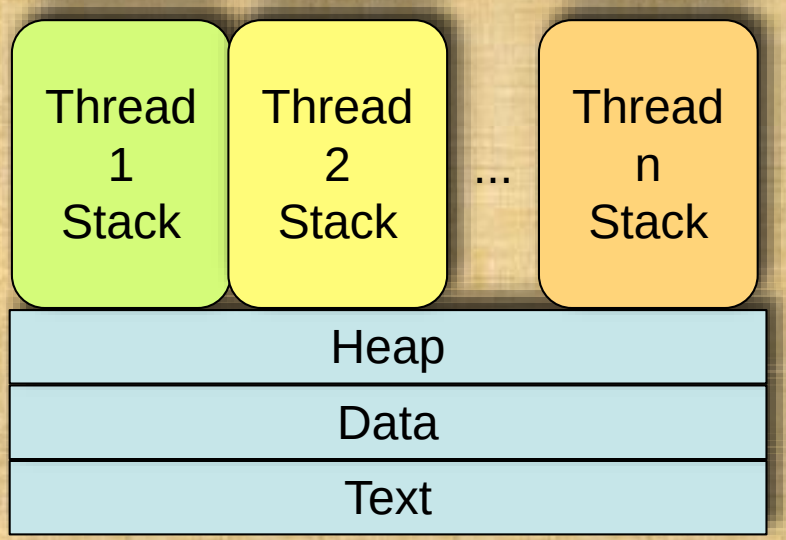


Two-level Model

- Similar to M:M, except that it allows a user thread to be **bound** to kernel thread

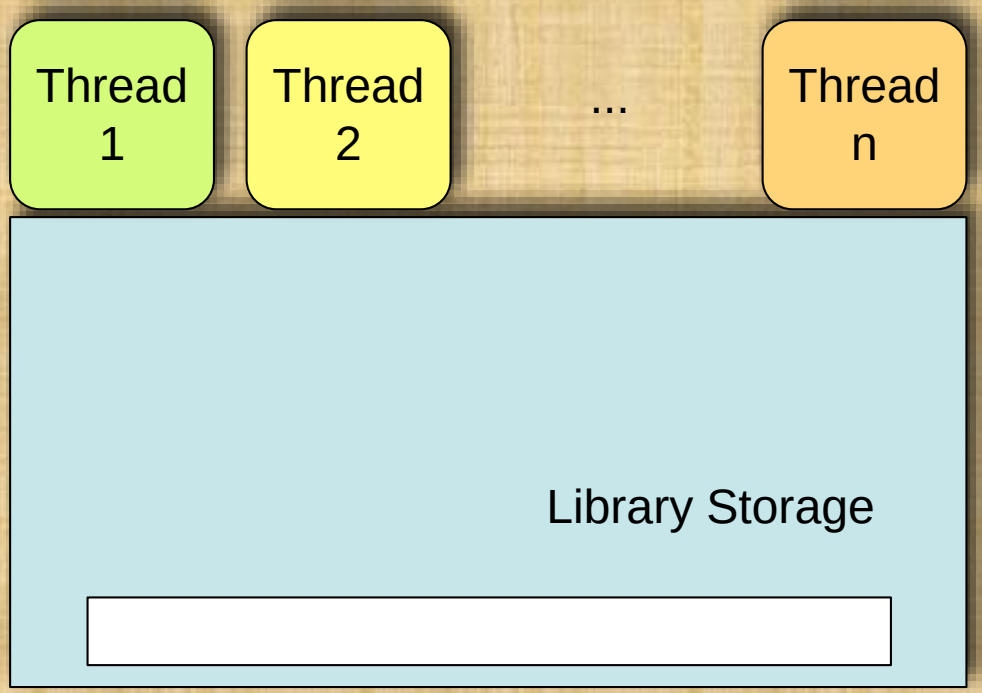


Shared Memory Model

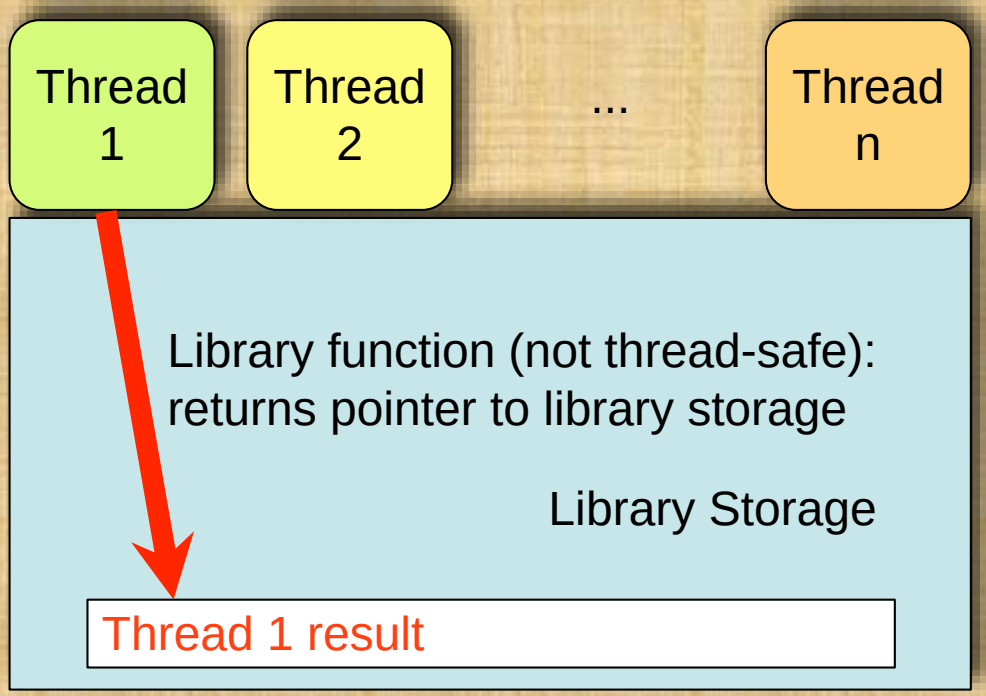


- All threads have access to the same global, shared memory
- Threads also have their own private data (how? where?)
- Programmers are responsible for protecting globally shared data

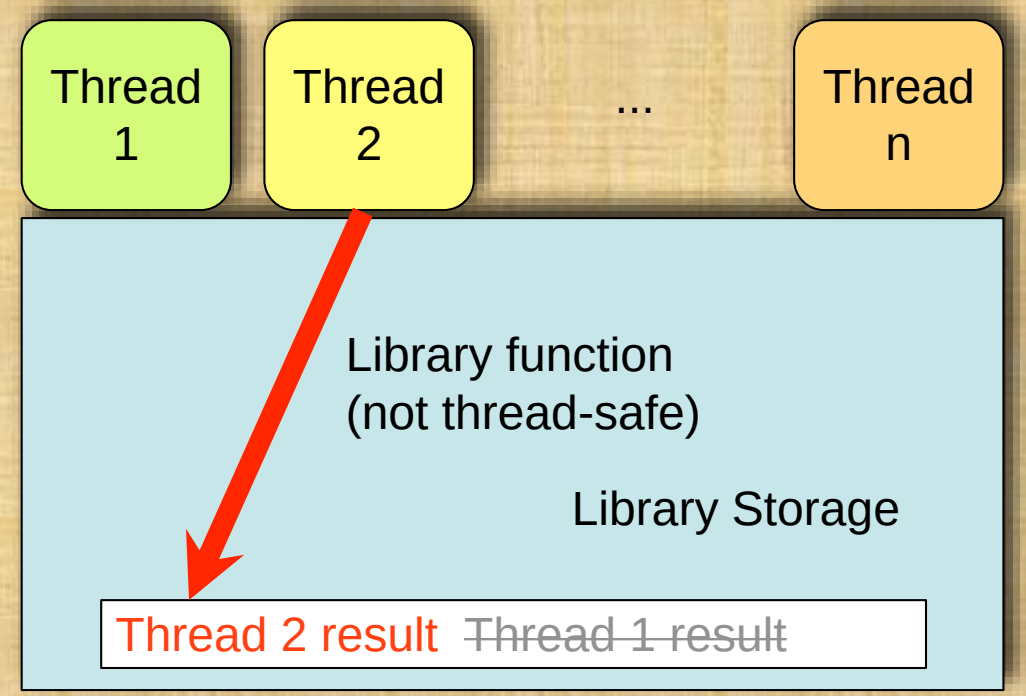
Thread Safeness



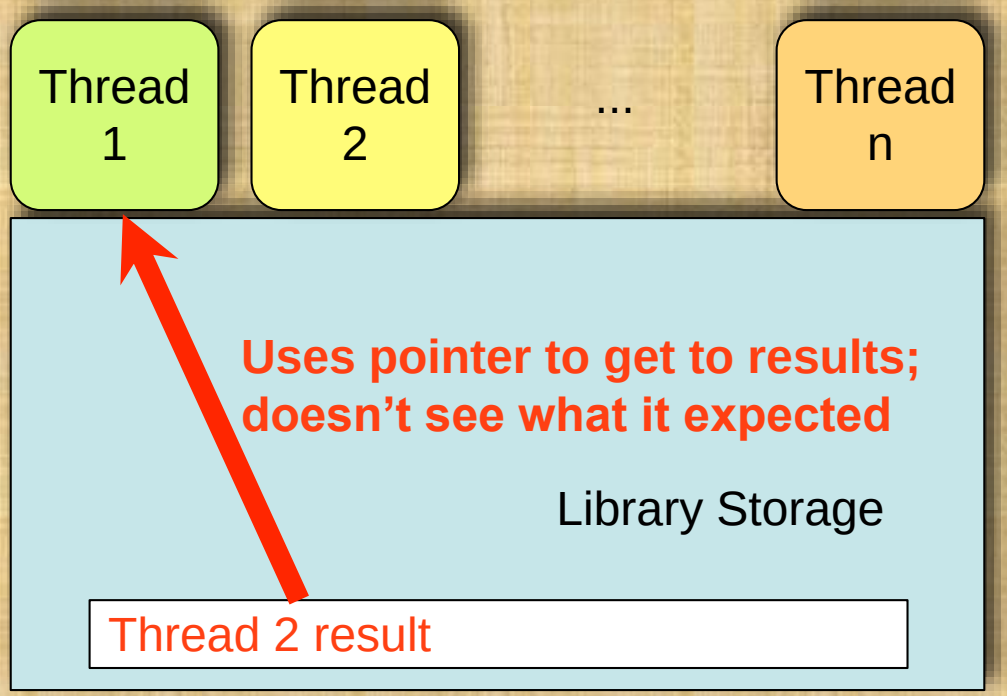
Thread Safeness



Thread Safeness



Thread Safeness



Remember this example …

```
#define  NUM_THREADS  5000   // everything else is the same
```

```
[xmeng@polaris thread]$ ./trd-share
main() reporting that all 5000 threads have terminated
v should be 5000, it is 4998
[xmeng@polaris thread]$
```

Who stole the two counts from me?!!

Implicit Threads - OpenMP

```
sum = 0;
for (auto i = 0; i < 100; i++){
    sum += a[i]
}
```

```
sum = 0;
#pragma omp parallel for shared(sum, a) reduction(+: sum)
for (auto i = 0; i < 100; i++) {
    sum += a[i]
}
```

```
sumloc_1 = a[0]  + ⋯ + a[33]
sumloc_2 = a[34] + ⋯ + a[66]
sumloc_3 = a[67] + ⋯ + a[99]

sum = sumloc_1 + sumloc_2 + sumloc_3
```

http://jakascorner.com/blog/2016/06/omp-for-reduction.html

A Simple Example

```
/* To compile, enter:
 *            gcc -fopenmp openmp.c -o openmp
 */
#include <omp.h>
#include <stdio.h>
int main(int argc, char *argv[]){
            /* sequential code */
            printf("I am in a sequential area 1\n");
            #pragma omp parallel
            {
                        printf("I am a parallel region\n");
            }
            /* sequential code */
            printf("I am in a sequential area 2\n");
            return 0;
}
```

http://www.eg.bucknell.edu/~cs315/F2021/meng/code/thread/openmp.c

Execution Result

Running the program with 4 threads

```
[xmeng@polaris thread]$ ./openmp
I am a sequential region1
I am a parallel region
I am a parallel region
I am a parallel region
I am a parallel region
I am a sequential region2
[xmeng@polaris thread]$
```

The number of threads in OpenMP can be set, either through the call to **`omp_set_threads()`** or through the setting of environment variable **`export OMP_NUM_THREADS=val`**
The default is **16**.

A Example with Shared Data

```
/* To compile, enter:  gcc -fopenmp openmp-m.c  -o oenmp-m */
#include <omp.h>
#include <stdio.h>
int main(int argc, char *argv[]){
        /* sequential code */
        int  v = 0;         int  tid;         int  nthreads;
        #pragma omp parallel shared(v, nthreads) private(tid)
        {
                tid = omp_get_thread_num();
                if (tid == 0)         {
                        nthreads = omp_get_num_threads();
                        printf("Number of threads = %d\n", nthreads);
                }
        #pragma omp critical (addv)
                {
                        v ++;
                }
        printf("I am a parallel region (thread id == %d)\n", tid);
        }
        /* sequential code */
        printf("value of v = %d\n", v);
        return 0;
}
```

http://www.eg.bucknell.edu/~cs315/F2021/meng/code/thread/openmp-m.c

Execution Result