SNS COLLEGE OF TECHNOLOGY

(Autonomous)

COIMBATORE-35

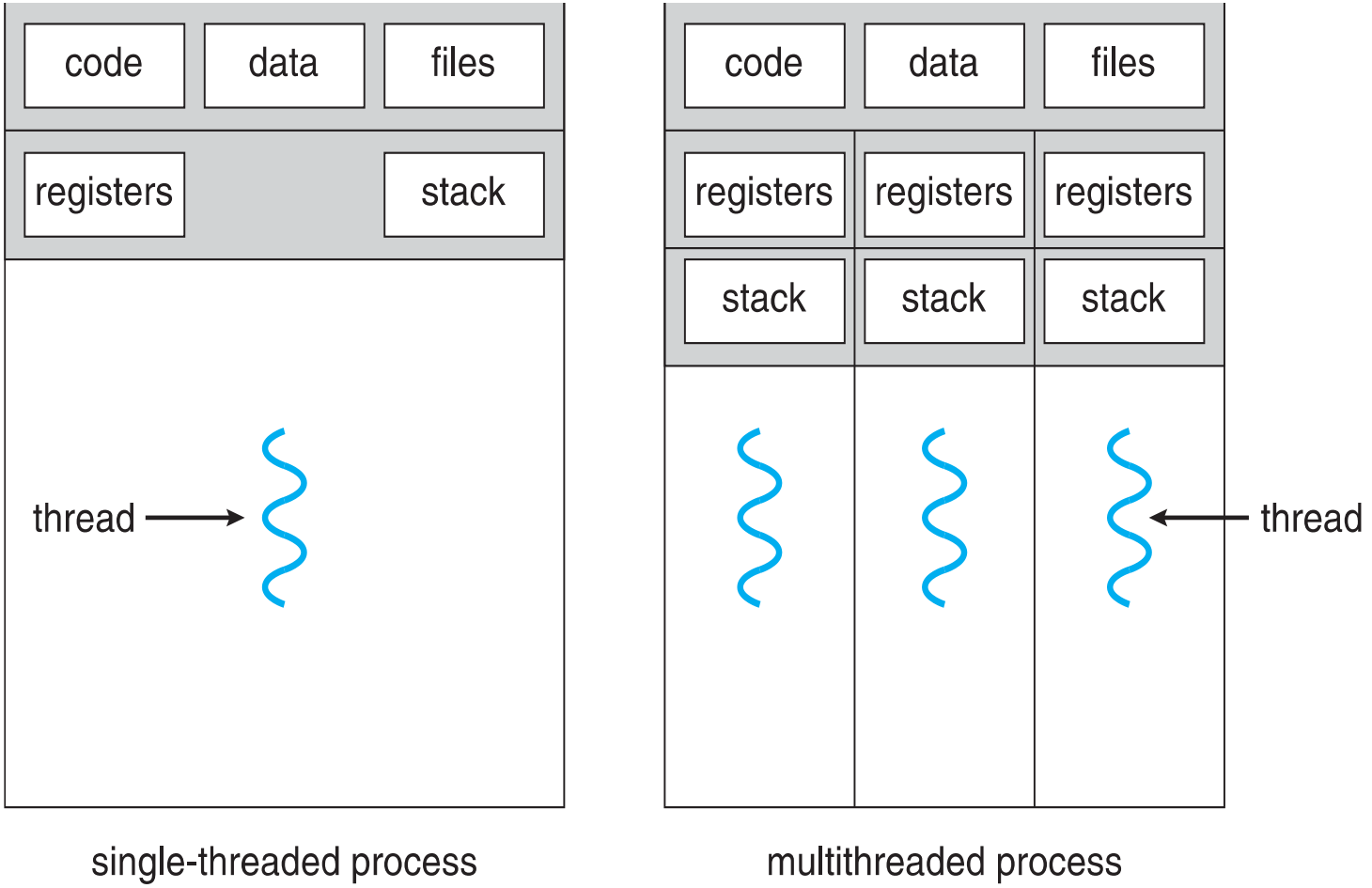
23CST201 Operating Systems

Threading Issues

Threads

- A thread is a basic unit of CPU utilization; **it comprises a thread ID, a program counter, a register set, and a stack**.
- It shares with other threads belonging to the same process its code section, data section, and other operating-system resources, such as open files and signals.
- A traditional (or heavyweight:) process has a single thread of control.
- If a process has multiple threads of control, it can perform more than one task at a time.

Single and Multithreaded Processes



Threading Issues

- Semantics of **fork()** and **exec()** system calls
- Thread Cancellation
- Signal handling
  - Synchronous and asynchronous
- Thread specific data
- Thread pool
- Scheduler Activations

1.Semantics of fork() and exec()

- The fork() system call is used to create a separate, duplicate process
- The semantics of the fork() and exec() system calls change in a multithreaded program.
- Does **fork()**duplicate only the calling thread or all threads?
  - Some UNIX systems have two versions of fork()

✓ one that duplicates all threads and

✓ another that duplicates only the thread that invoked the fork()

- **exec()** usually works as normal – replace the running process including all threads

2. Thread Cancellation

- **Thread Cancellation** is the task of terminating a thread before it has completed.

Example:

✓ If multiple threads are concurrently searching through a database and one thread returns the result, the remaining threads might be cancelled.

- A thread that is to be cancelled is often referred to as the **target thread**
- **Asynchronous cancellation -** One thread immediately terminates the target thread.
- **Deferred cancellation -**The target thread periodically checks whether it should terminate, allowing it an opportunity to terminate itself in an orderly fashion.

```
pthread_t tid;

/* create the thread */
pthread_create(&tid, 0, worker, NULL);

. . .

/* cancel the thread */
pthread_cancel(tid);
```

3.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 the occurrence of a particular event
  2. Signal is delivered to a process
  3. Once delivered, the signal must be handled.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

- 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

4.Thread Pools

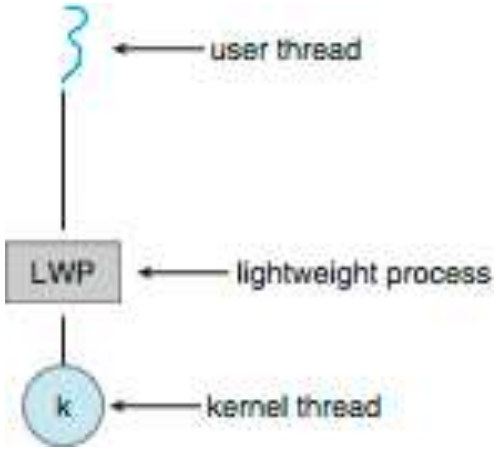
- Create a number of threads in a pool where they **await work**
- Advantages:
  - Usually slightly faster to service a request with an existing thread than create a new thread
  - Allows the number of threads in the application(s) to be bound to the size of the pool
  - Tasks could be scheduled to run periodically
- Windows API supports thread pools

5.Thread Specific Data

- **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(Thread Local Storage) visible across function invocations
- Similar to **static** data
  - TLS is unique to each thread

6.Scheduler Activations

- Scheme for **communication** between the **user-thread library and the kernel** is known as scheduler activation
- Both M:M and Two-level models require communication to maintain the appropriate number of kernel threads allocated to the application
- Typically use an intermediate data structure between **user and kernel threads** – **lightweight process** (**LWP**)
  - Appears to be a virtual processor on which process can schedule **user thread to run**
  - Each LWP attached to kernel thread
  - How many LWPs to create?



6.Scheduler Activations

- Scheduler activations provide **upcalls** - a communication mechanism from the kernel to the **upcall handler** in the thread library
- This communication allows an application to maintain the correct number kernel threads

References

1. Silberschatz, Galvin, and Gagne, “Operating System Concepts”, Tenth Edition, Wiley India Pvt Ltd, 2018
2. Andrew S. Tanenbaum, “Modern Operating Systems”, Fourth Edition, Pearson Education, 2010.
3. William Stallings, “Operating Systems – Internals and Design Principles”, 7th Edition, Prentice Hall, 2011

Summarization