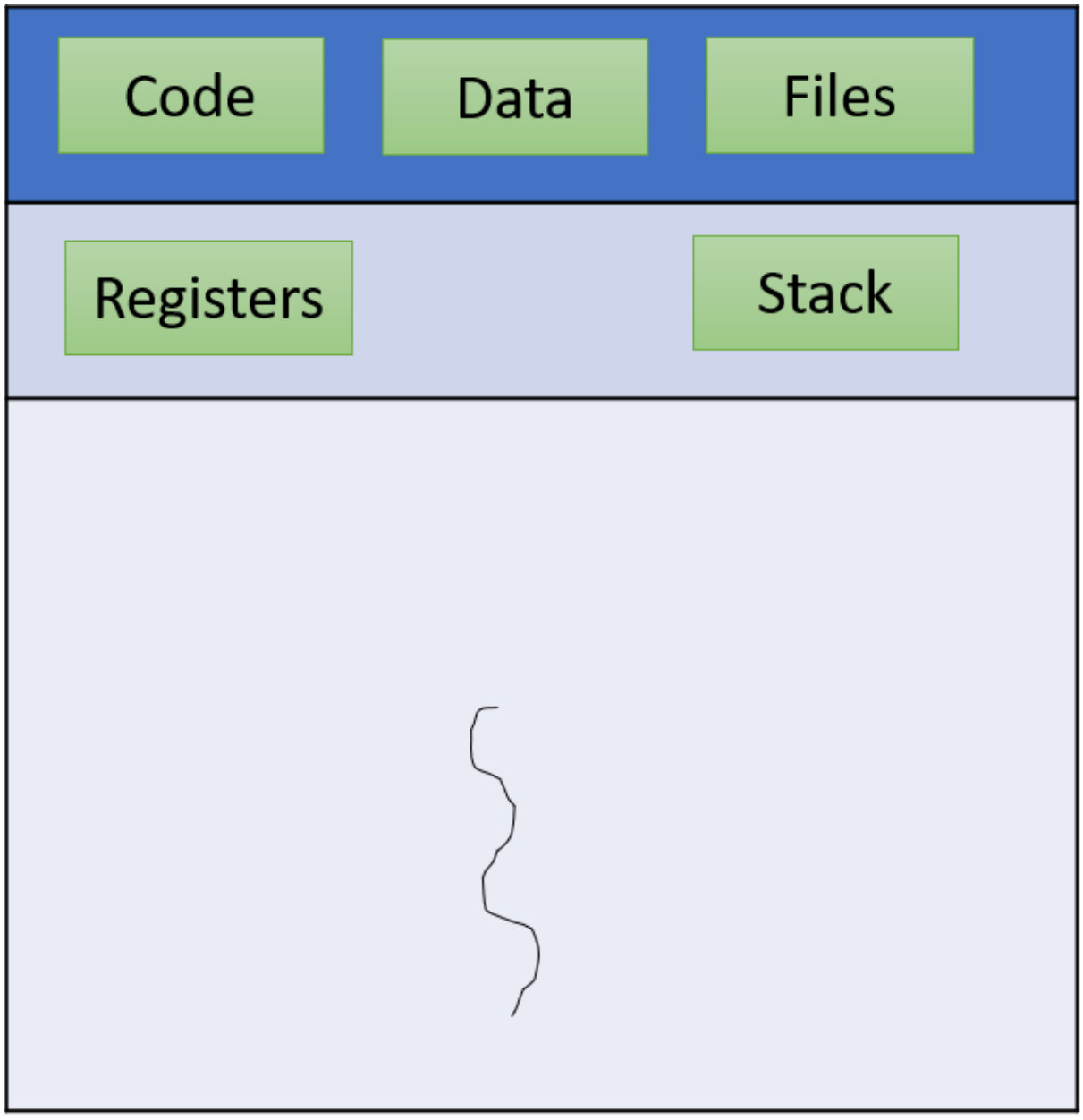
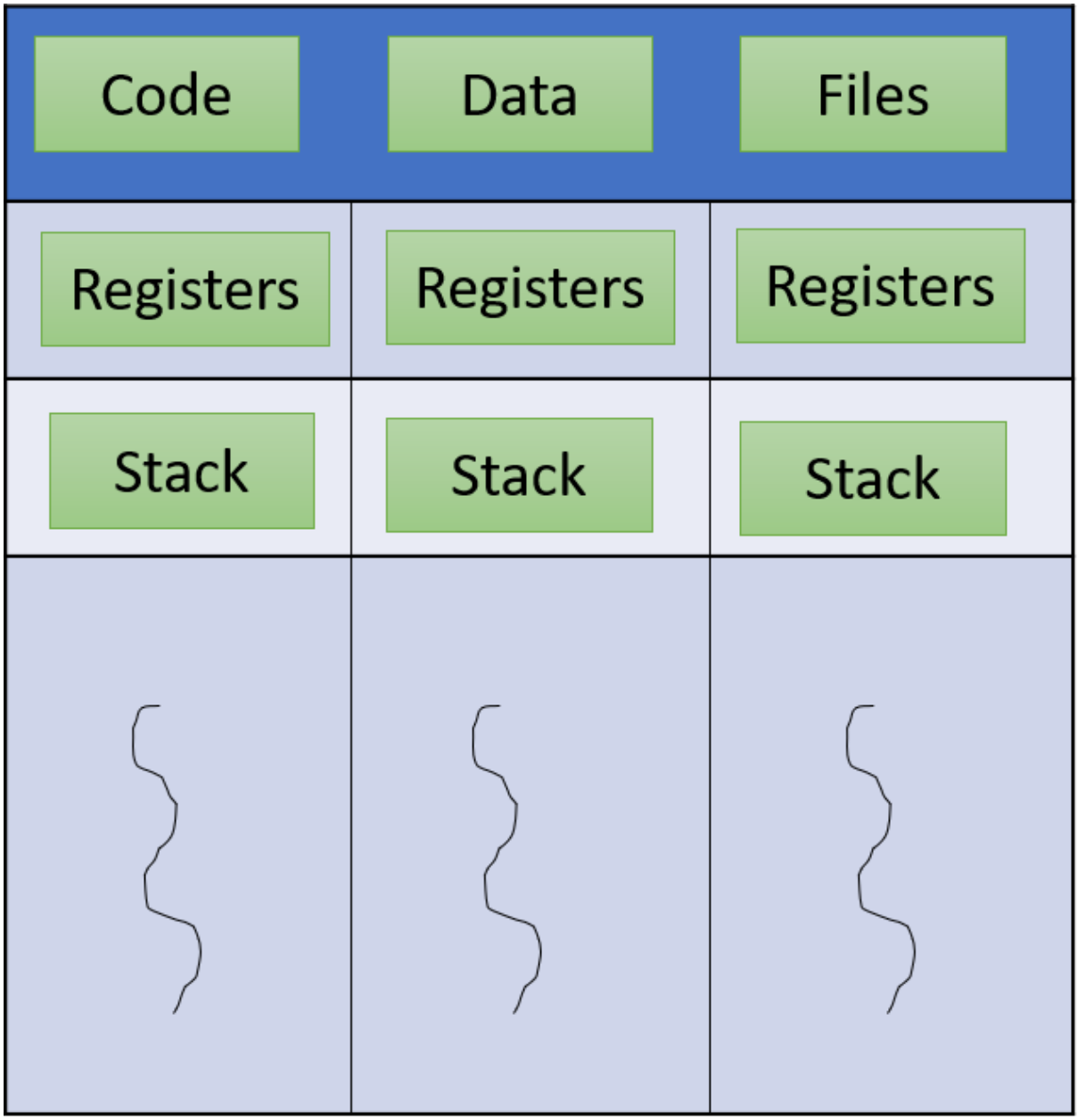
Threads in Operating System

1. What are Threads in Operating System

A thread is the smallest unit of execution which has its own thread ID, program counter, register set and stack. All the threads that belong to the same process share the code, data section and other resources like open files belonging to the process. So, remember thread is a part of the process. A thread can not exist without a process.

Single Threaded Process
Multi-threaded Process





1.1 Benefits of creating threads in Operating System

1. **Responsiveness** – multi-threading increase the responsiveness of the process. For example, in MSWord while one thread does the spelling check the other thread allows you to keep tying the input. Therefore, you feel that Word is always responding.
2. **Resource sharing** – All the threads share the code and data of the process. Therefore, this allows several threads to exist within the same address space
3. **Economy** – For the same reason as mentioned above it is convenient to create threads. Since they share resources they are less costly
4. **Scalability** – Having a multiprocessor system greatly increases the benefits of multithreading. As a result, each thread can run in a separate processor in parallel.

1.2 Challenges for Programmers while creating Threads

1. **Dividing activities** – It involves finding the functions within the job that can be run in parallel on separate processors.
2. **Balance** – The task assigned to each processor must also be equal. Now there can be different parameters for that. One parameter can be, assign equal tasks to each processor. But, tasks assigned to more processor may require higher execution time thus

overloading one processor. Thus, simply assigning equal tasks to each processor may not work.

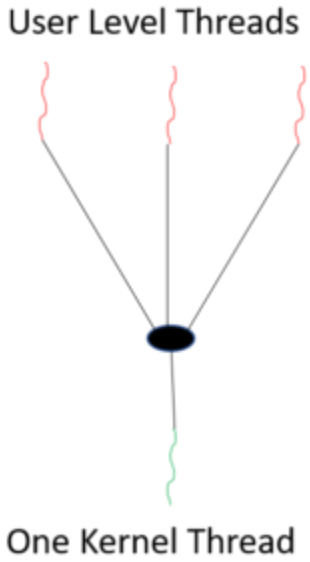
3. **Data splitting** – Another challenge is to split the data required for each task.
4. **Data dependency** – sometimes the data required by one thread (T1) might be produced by another (T2). Thus, T1 can not run before T2. Therefore, it becomes difficult for programmers to code.
5. **Testing and debugging** – Multiple threads running in parallel on multiple cores poses another challenge in the testing of applications.

2. Multi-threading Models

There are two kinds of threads in the system – *user threads* and *kernel threads.* **User threads** are supported above the kernel and managed without kernel support whereas **Kernel threads** are supported and managed directly by OS. Ultimately, a relationship must exist between user threads and kernel threads. In this section, we look at three common ways of establishing such a relationship.

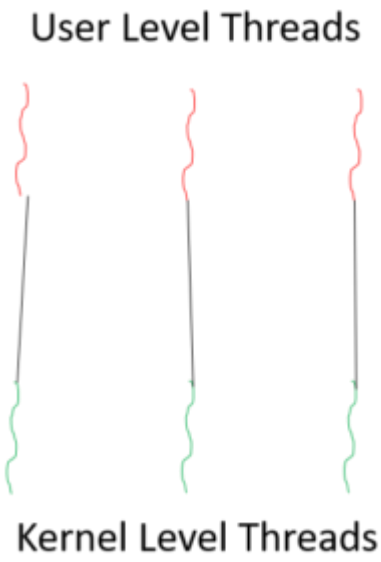
2.1 Many to One Model

1. Many user-level threads mapped to a single kernel thread
2. Thread library is in user space
3. Drawback: The Entire process will block if a thread makes a blocking system call
4. Examples:
    - **Solaris Green Threads**
    - **GNU Portable Threads**



2.2 One to One Model

1. Each user-level thread maps to kernel thread
2. Drawback: creating a user thread requires creating a corresponding kernel thread. Since the user can develop a malicious application to create unlimited threads. Therefore, the system too have to create those many kernel-level threads. As a result, the system might slow down.
3. Examples:
    - Windows NT/XP/2000
    - Linux Solaris 9 and later



2.3 Many to Many Model

1. Allows many user-level threads to be mapped to many kernel threads
2. Allows the operating system to create a sufficient number of kernel threads. The operating system creates a pool of kernel-level threads called the thread-pool. As long as a kernel-level thread is available in the thread-pool it is allocated to a user-level thread.
3. Example: Solaris prior to version 9