Threads Introduction

Spring 2022

Reference

- Modern Operating System
- 2.2.1 The Thread Model
- 2.2.2 Thread Usage
- Andrew S. Tanenbaum
- 2nd edition

Thread: Introduction

- Each process has
  1. Own Address Space
  2. Single thread of control
- A process model has two concepts:
  1. Resource grouping
  2. Execution
- Sometimes it is useful to separate them

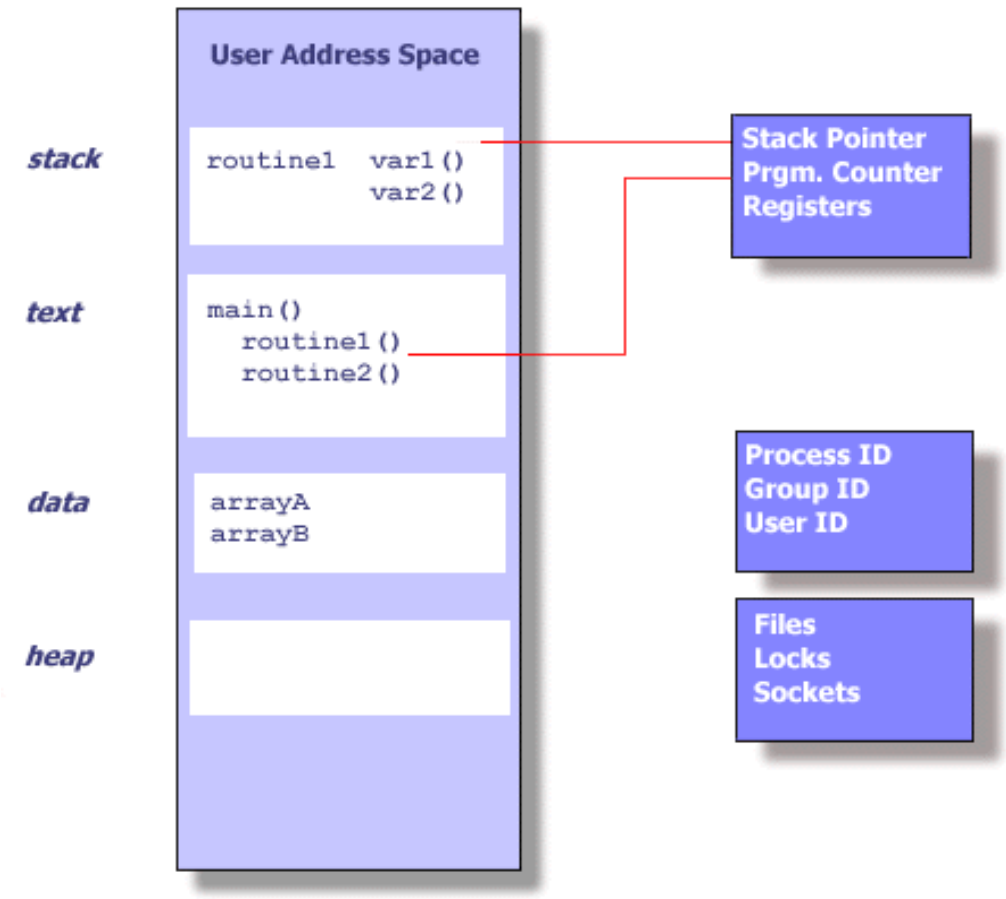
Unit of Resource Ownership

- A process has an
  - Address space
  - Open files
  - Child processes
  - Accounting information
  - Signal handlers
- If these are put together in a form of a process, can be managed more easily

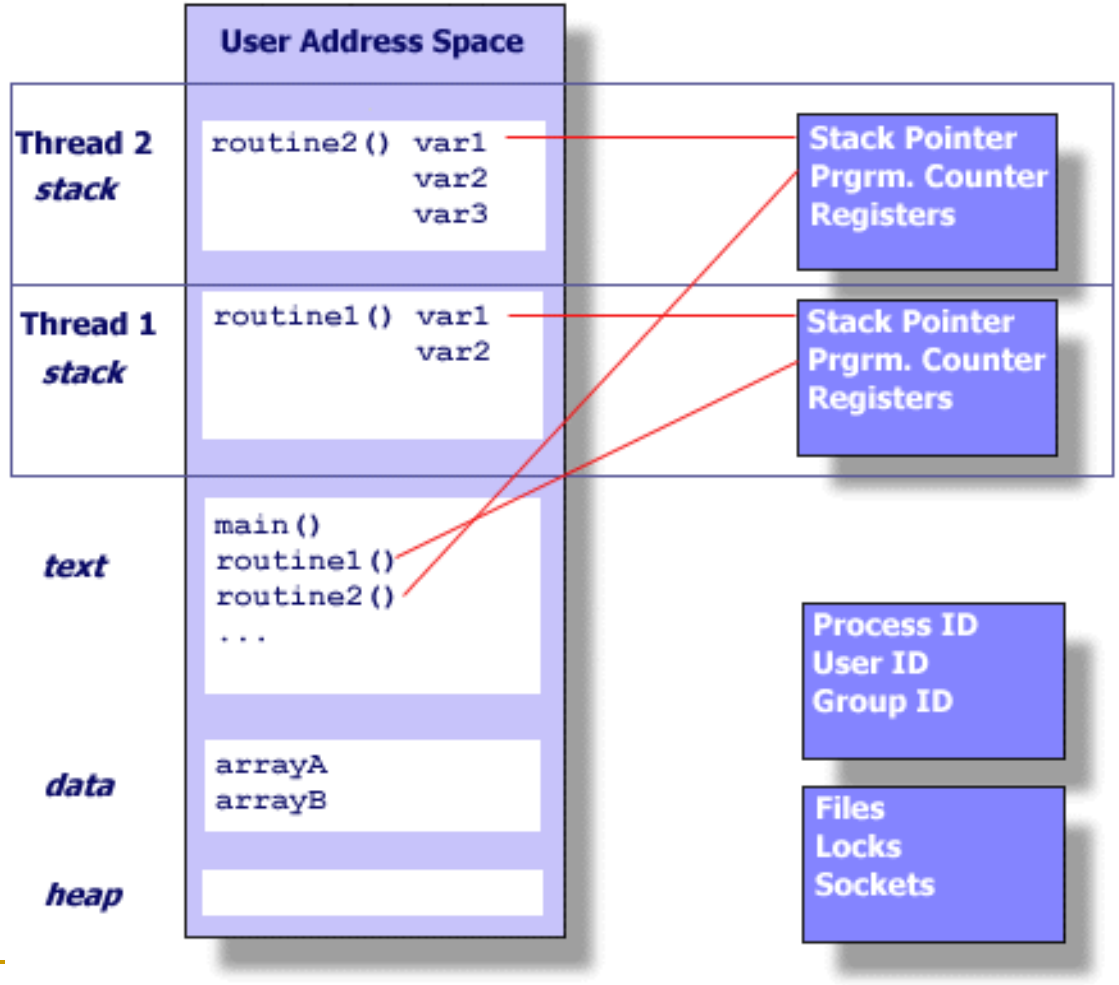
Unit of Dispatching

- Path of execution
  - Program counter: which instruction is running
  - Registers:
    - holds current working variables
  - Stack:
    - Contains the execution history, with one entry for each procedure called but not yet returned
  - State
- Processes are used to group resources together
- Threads are the entities scheduled for execution on the CPU
- Threads are also called ***lightweight*** process

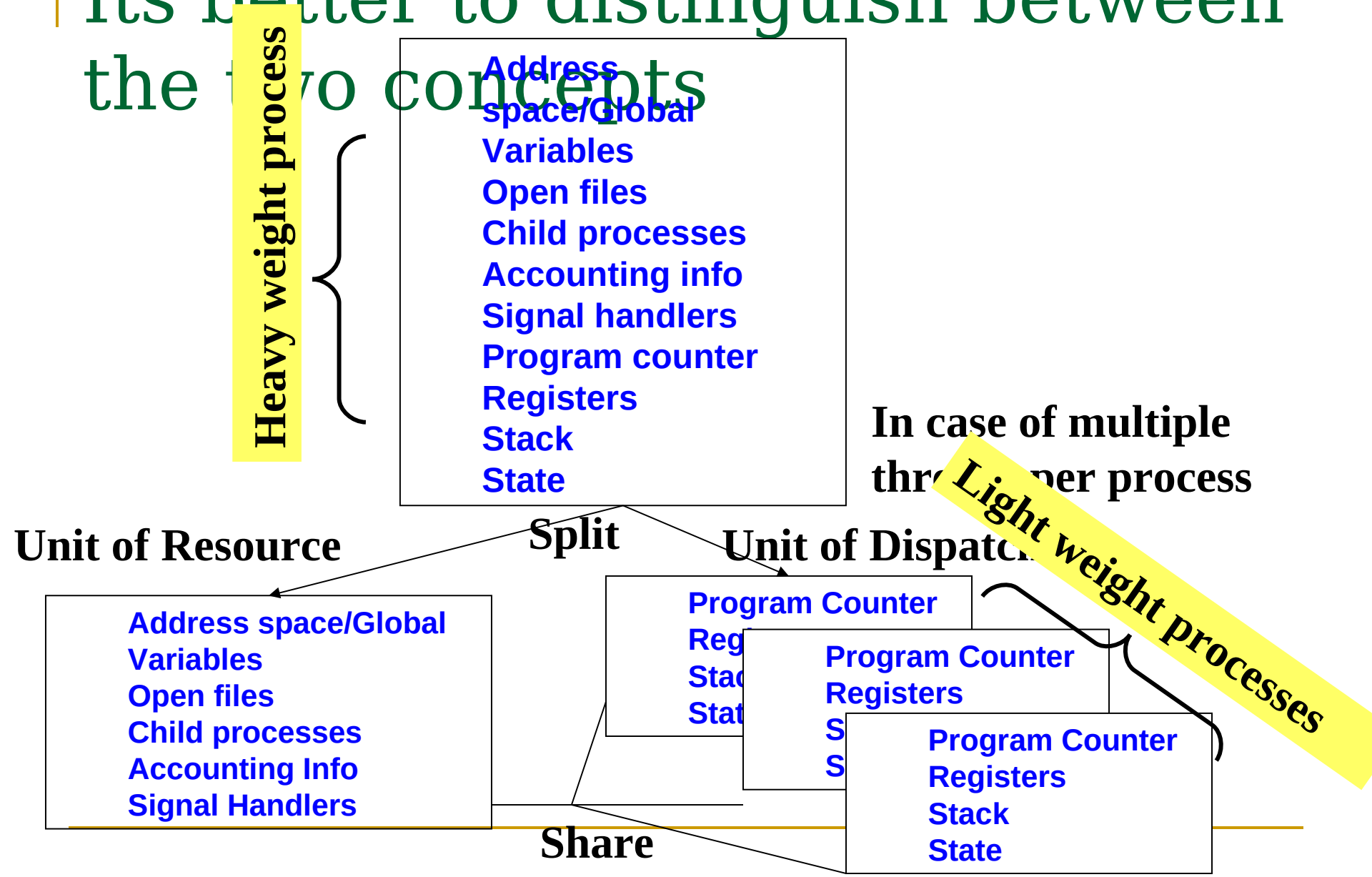
UNIX Process



Threads within a UNIX Process



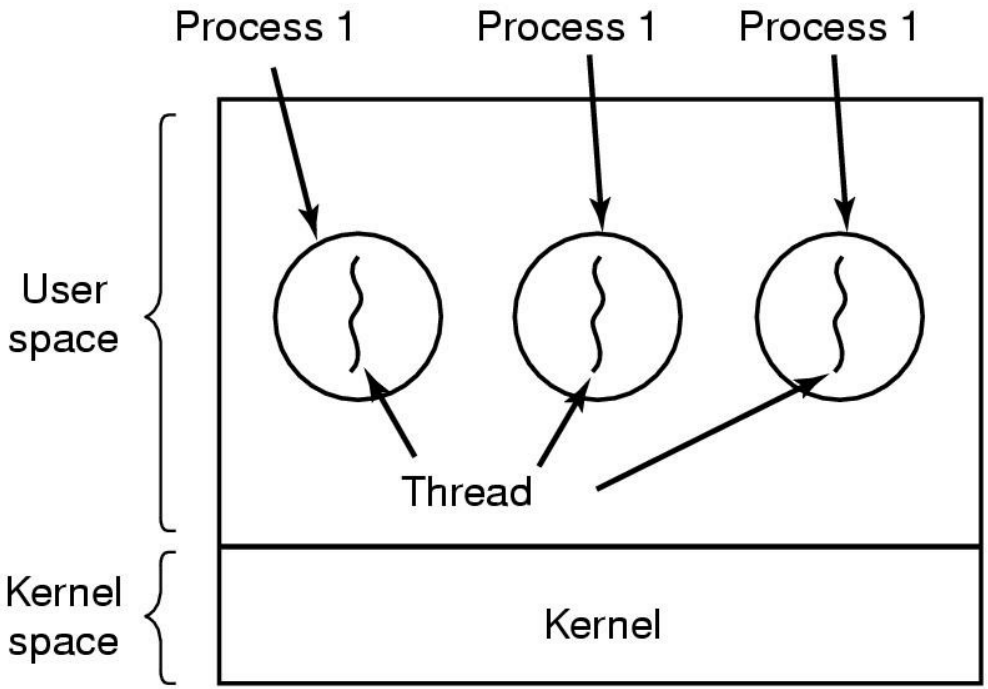
Its better to distinguish between the two concepts



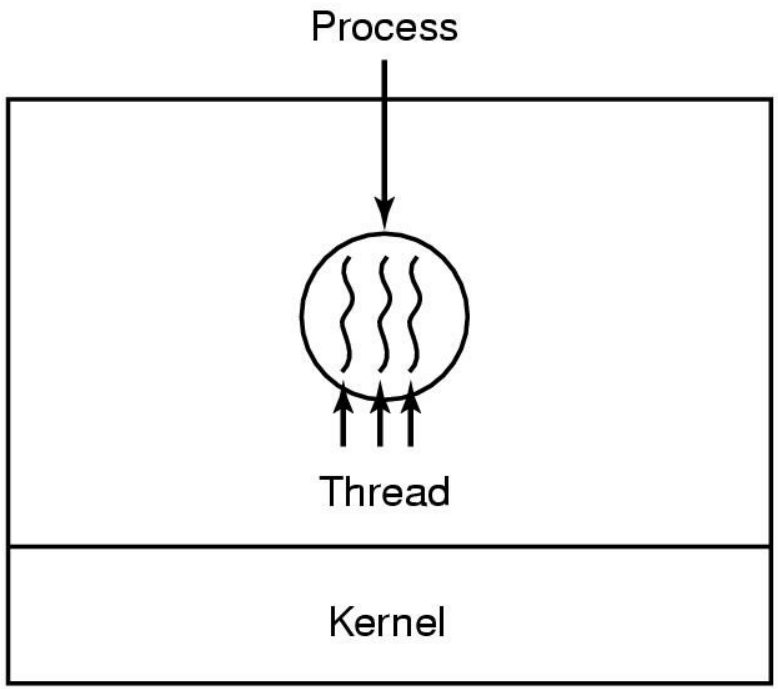
Threads

- Allow multiple execution paths in the same process environment
- Threads share address space, open files etc
- But have own set of Program counter, Stack etc
- The first thread starts execution with
  - `int main(int argc, char *argv[])`
- The threads appear to the Scheduling part of an OS just like any other process

Process Vs. Threads

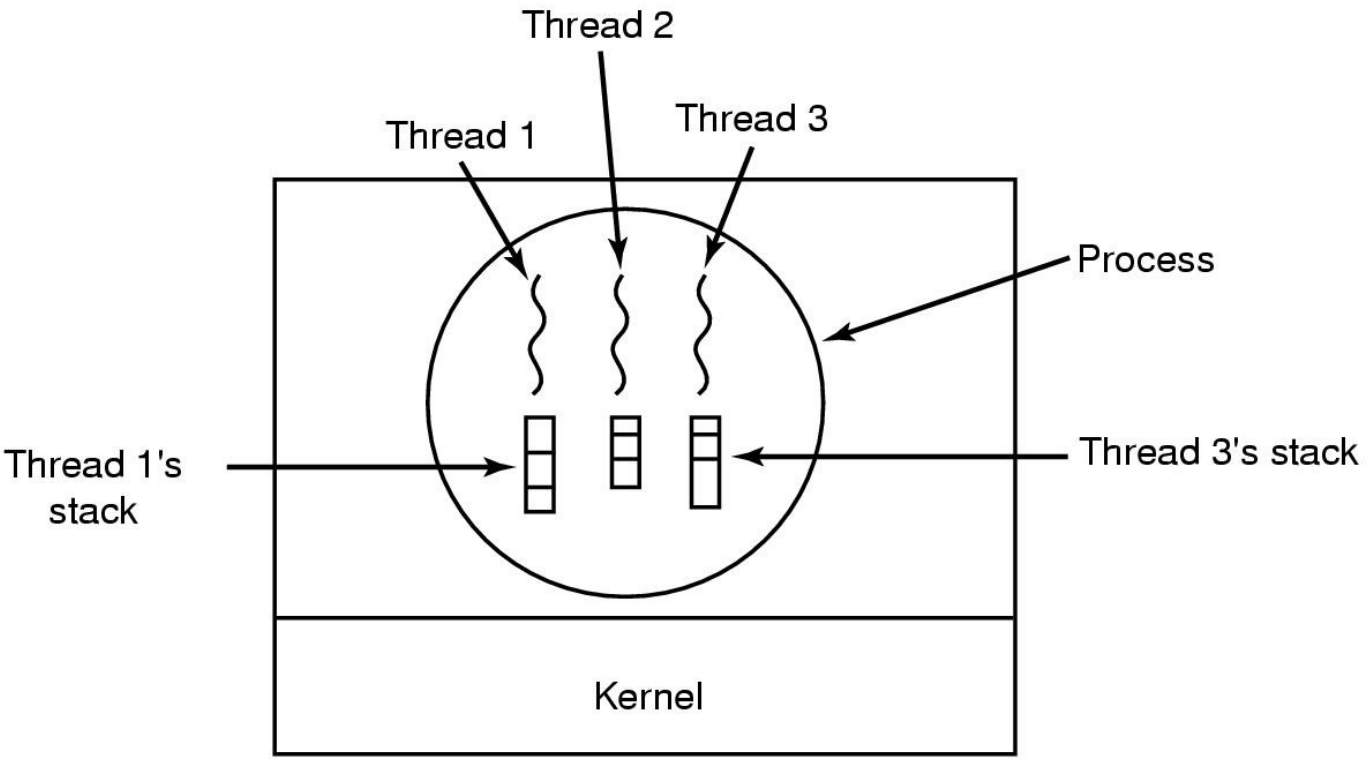


(a) Three threads, each running in a separate address space



(b) Three threads, sharing the same address space

The Thread Model

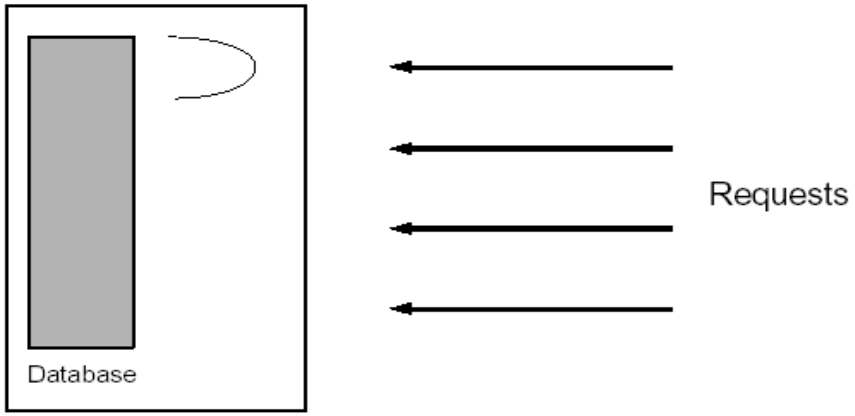


Each thread has its own stack

Thread Usage

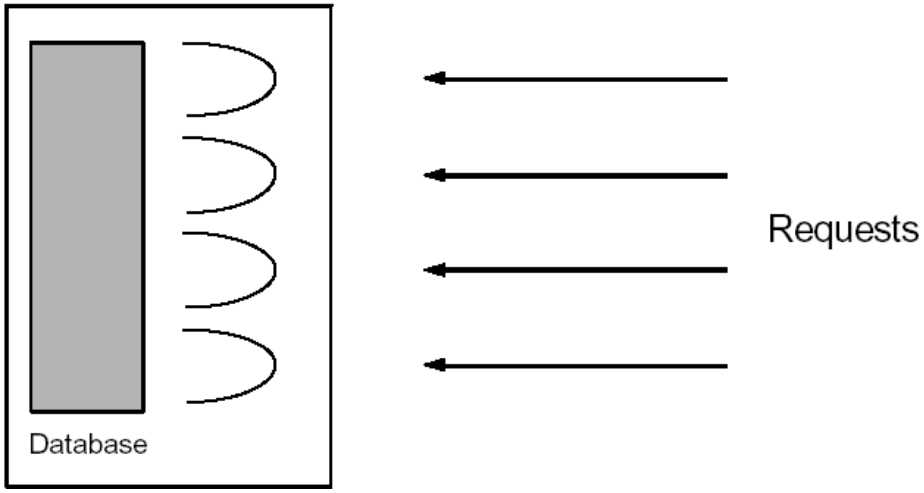
- Less time to create a new thread than a process
  - the newly created thread uses the current process address space
  - no resources attached to them
- Less time to terminate a thread than a process.
- Less time to switch between two threads within the same process, because the newly created thread uses the current process address space.
- Less communication overheads
  - threads share everything: address space, in particular. So, data produced by one thread is immediately available to all the other threads
- Performance gain
  - Substantial Computing and Substantial Input/Output
- Useful on systems with multiple processors

1. Single threaded database server



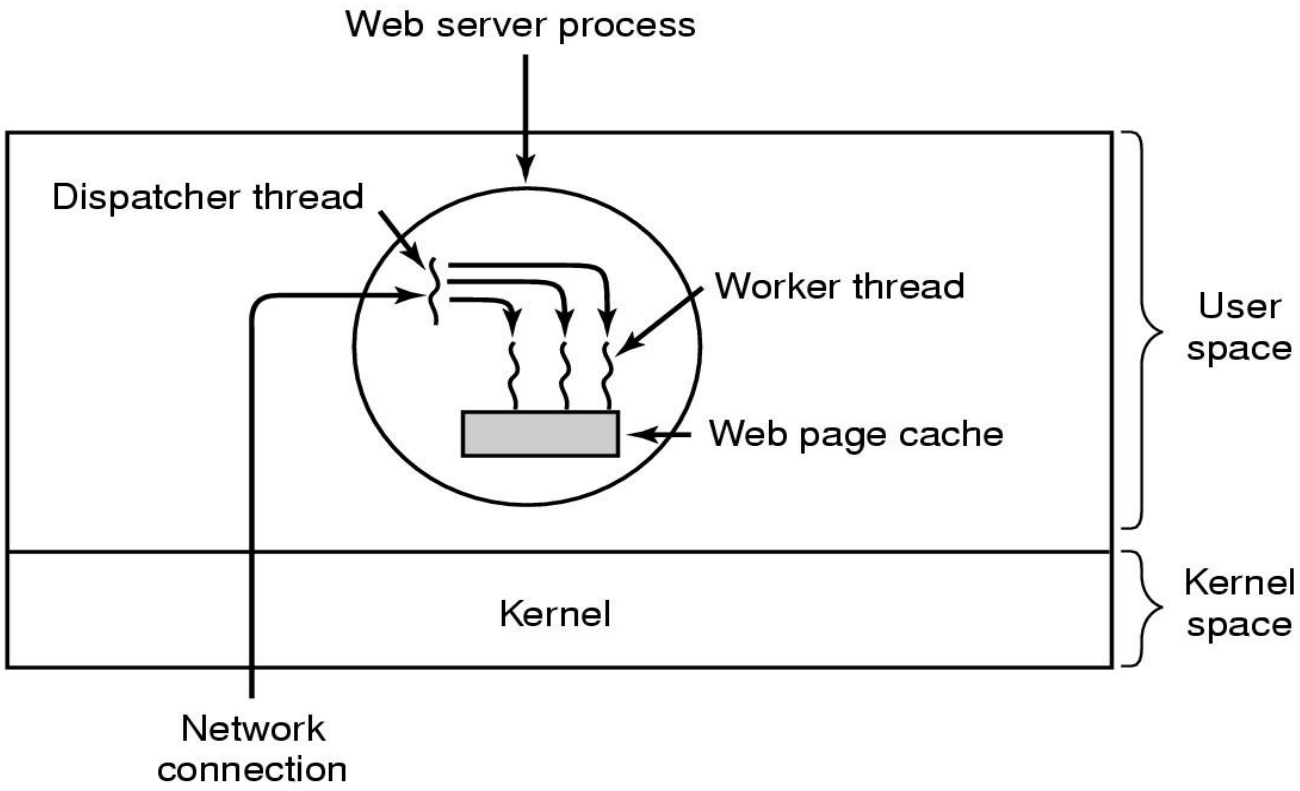
- Handles multiple clients
  - Either handle the requests sequentially
  - Or multiplex explicitly by creating multiple processes
- Problems
  - Unfair for quick requests, occurring behind lengthy request
  - Complex and error prone
  - Heavy IPC required

1. Multithreaded database server



- Assign a separate thread to each request
- As fair as in the multiplexed approach.
- The code is as simple as in the sequential approach, since the address space is shared - all variables are available
- Some synchronization of access to the database is required, this is not terribly complicated.

e.g. A multithreaded web server

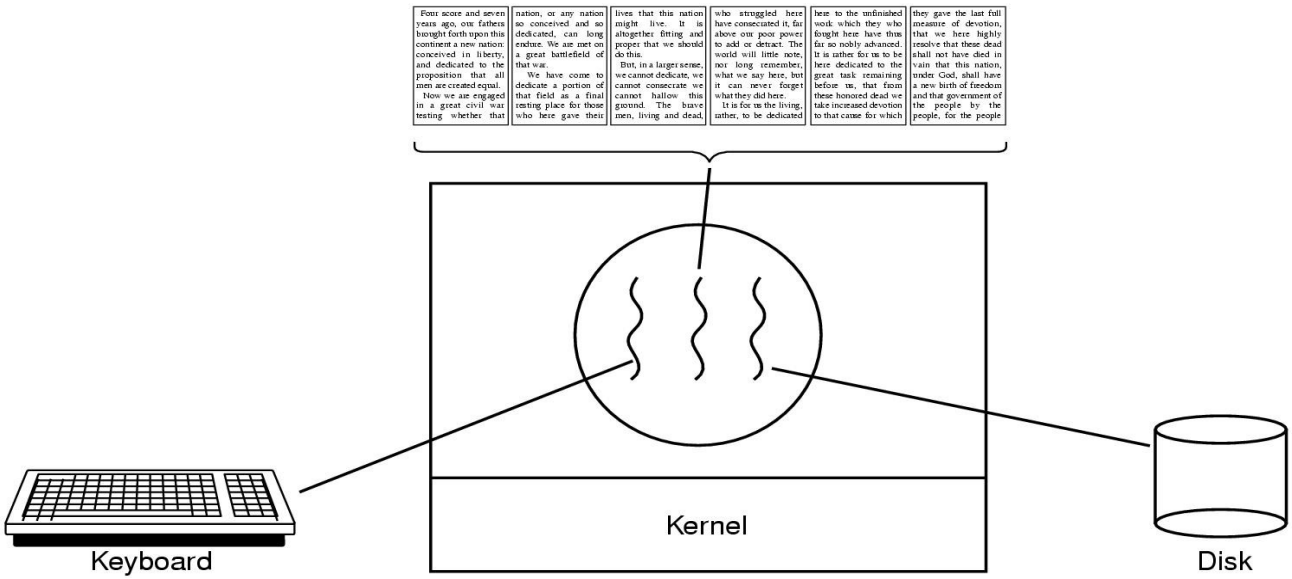


2. Background Processing

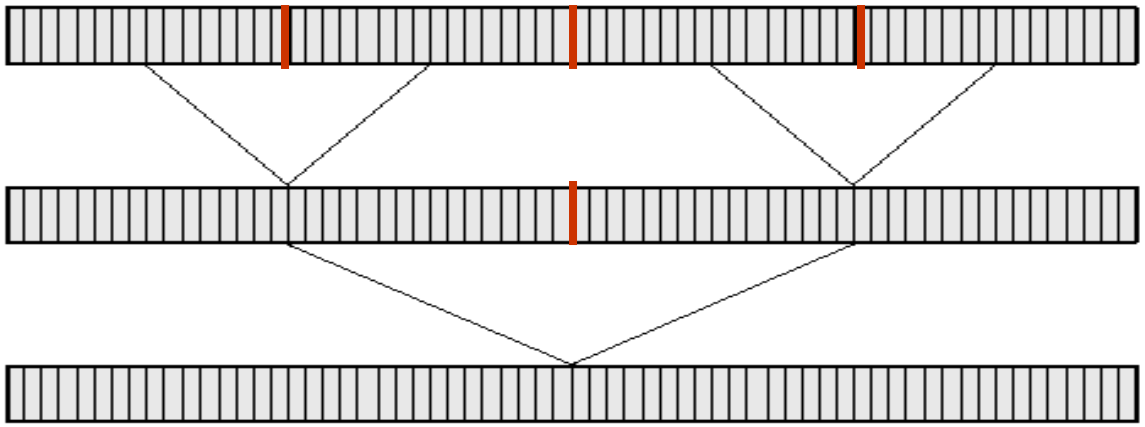
- Consider writing a GUI-based application that uses:
  - Mouse
  - Keyboard input
  - Handles various clock-based events

- In a single threaded application, if the application is busy with one activity, it cannot respond (quickly enough) to other events, such as mouse or keyboard input.
- Handling such concurrency is difficult and complex
- But simple in a multithreaded process

e.g. A word processor with 3 threads



3. Parallel Algorithms e.g. Mer



- Sort some data items on a shared-memory parallel computer.
- Our task is merely to implement a multithreaded sorting algorithm.
  - Divide the data into four pieces
  - Have each processor sort a different piece.
  - Two processors each merge two pieces
  - One processor merges the final two combined pieces.