CS4500
Operating Systems

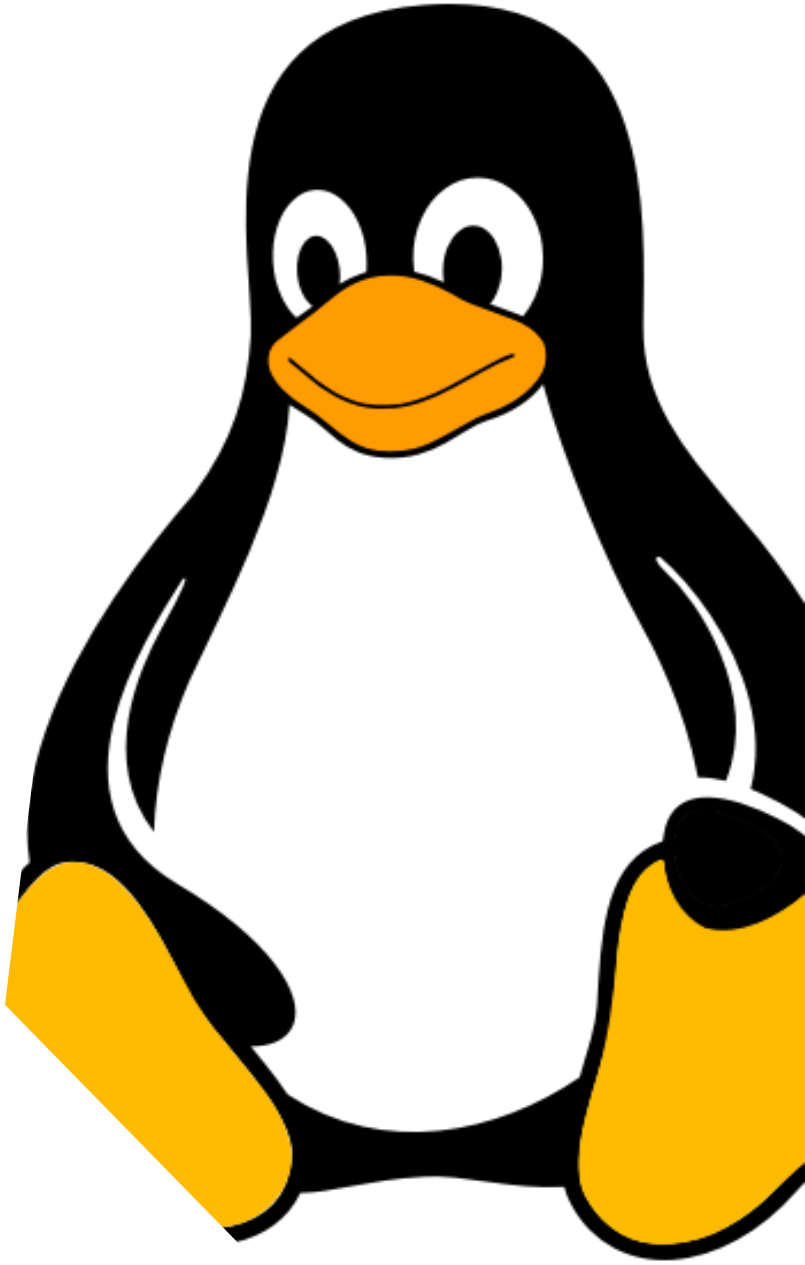
Week 4

Course Content

Unit	Module	Week	Topic
1 – Introduction	1 & 2	1	Welcome to OS & Introduction
	2	2	OS Overview
2 – Processes and Threads	3	3	Processes
	4	4	Threads
	5	5	CPU Scheduling
	6	6	Multiprocessor Scheduling
	7	7	Interprocess Communication
	8	8	Pthread
3 – Deadlocks & Memory	9	9	Deadlock
	10	10	Memory Management
	--	11	Midterm Exam
4 – Additional Topics	11	11	Page Replacement
	12	13	File Systems
	13	14	IO Devices
	14	15	Security
	--	16	Final Exam

CS 4500: Operating Systems I

Threads



What to expect today:

An overview of threads

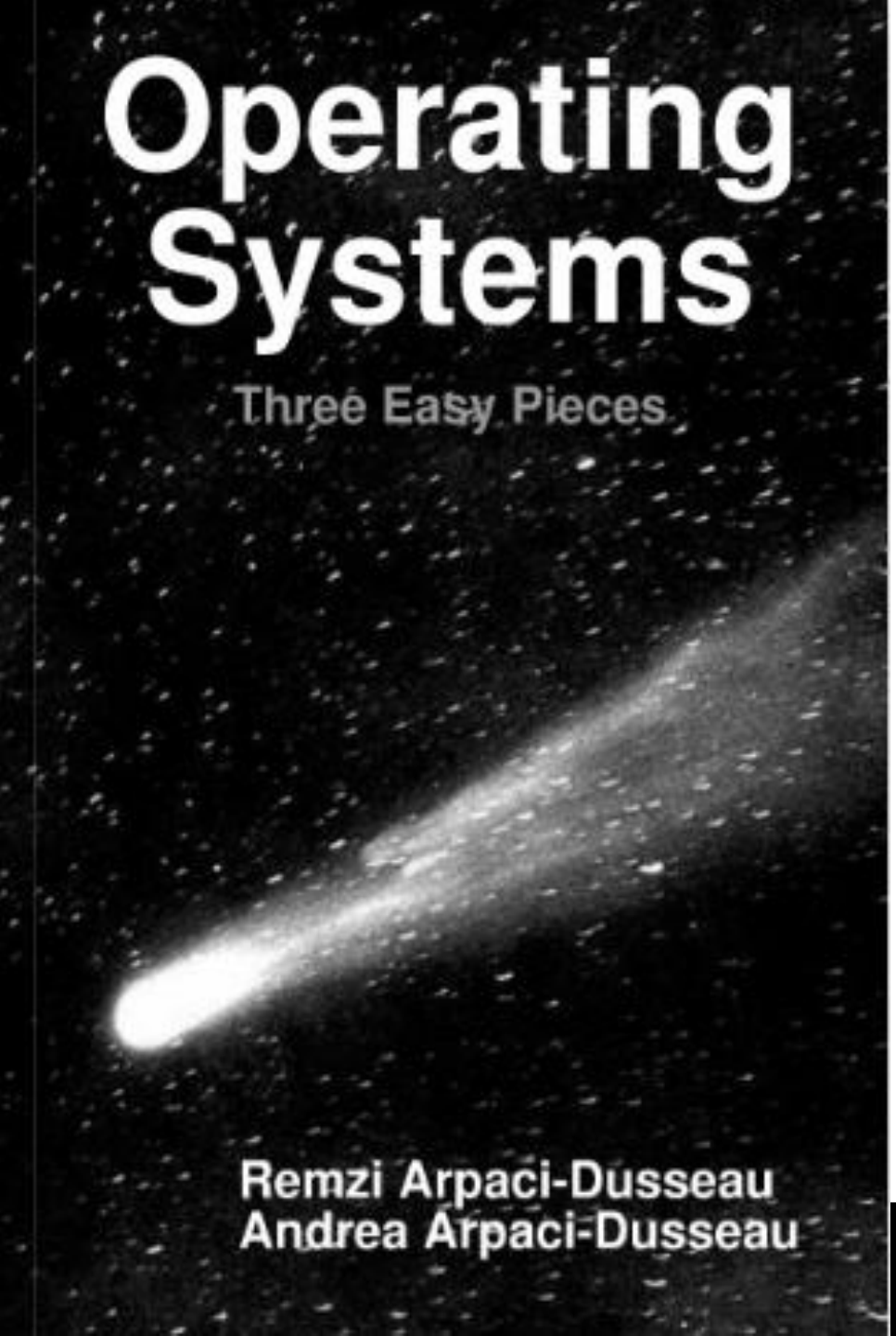
Assigned reading:

Andrew S. Tanenbaum, Modern Operating Systems, 4thd edition, 2014, Prentice Hall Chapter 2

Lecture Resources

Andrew S. Tanenbaum, Modern Operating Systems, 5th edition, 2022, Prentice Hall

Peter Jay Salzman, Michael Burian, Ori Pomerantz, Bob Mottram, Jim Huang. The Linux Kernel Module Programming Guide, 2022. https://sysprog21.github.io/lkmpg/



Additional Resource

Remzi H. Arpaci-Dusseau, R. H., Arpaci-Dusseau, A.C. *Operating systems: Three easy pieces: Chapter 10 and 28.* (2018). Arpaci-Dusseau Books.

https://pages.cs.wisc.edu/~remzi/OSTEP/cpu-sched-multi.pdf

Copy of .pdf available in Canvas

Processes

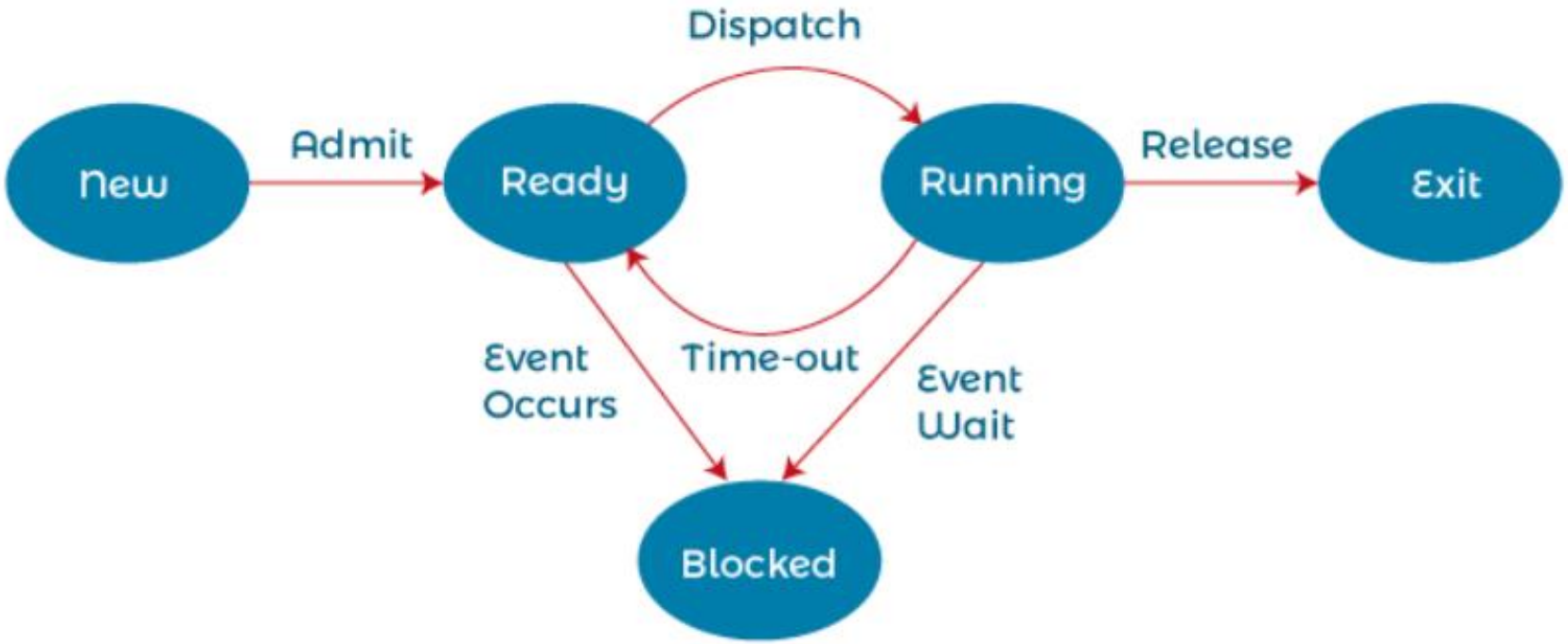
- Definition: A program in execution
- 5-state process model
- Process control block
- Process image in memory

Linux processes

- The `task_struct` structure

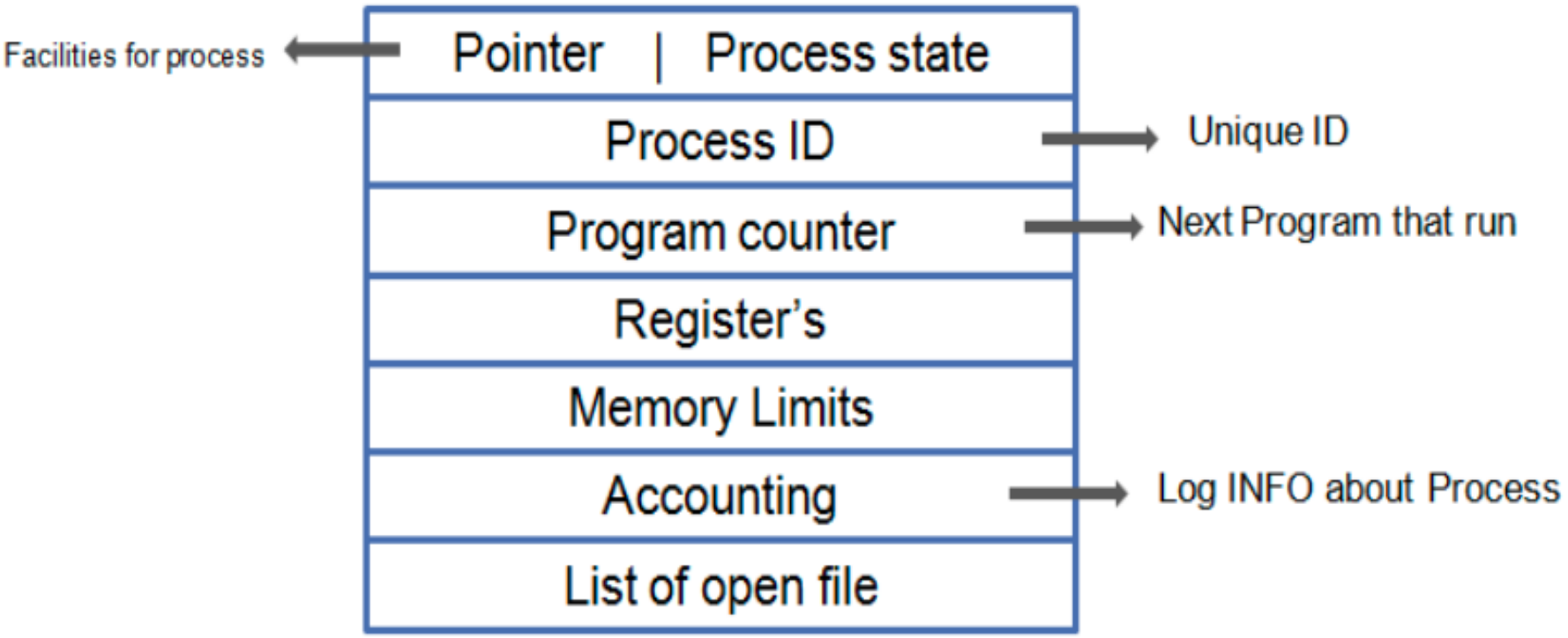
Review of the Last Class

Review Five State Process Model



Fire State Process Model

PROCESS CONTROL BLOCK (PCB)

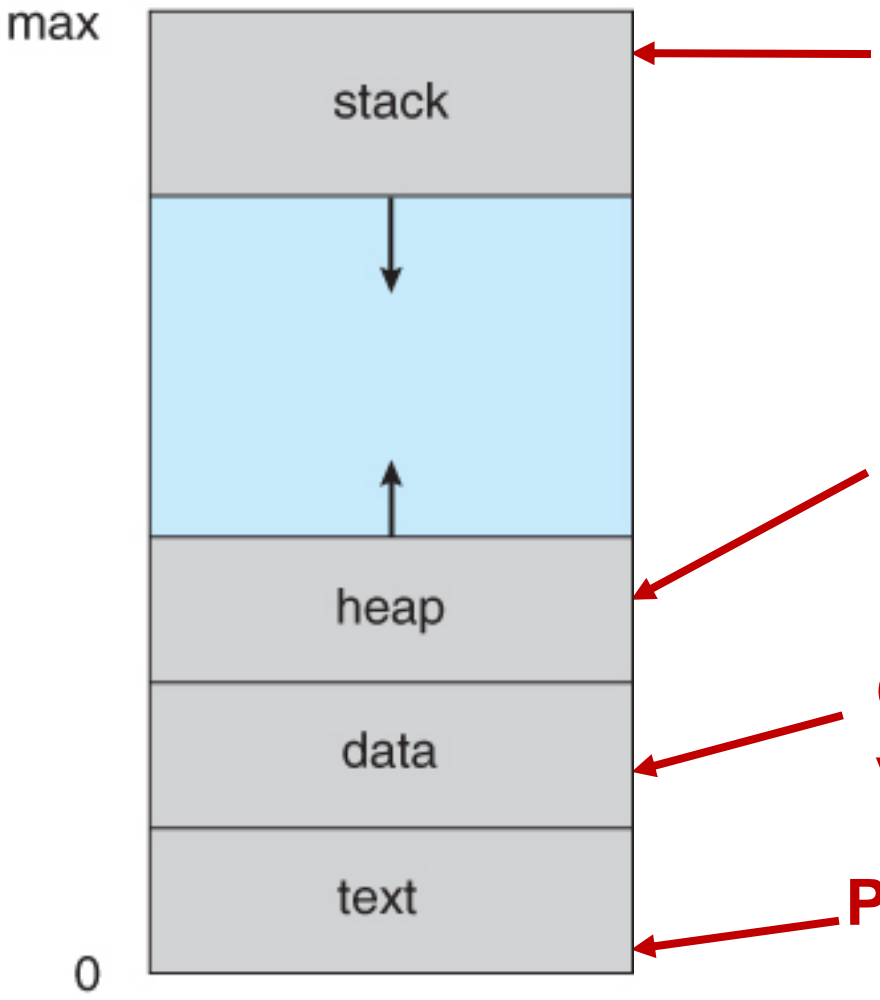


PCB Diagram

Image retrieved from: https://techaccess.in/2021/05/07/process-control-block/

REVIEW

Review: Process Image in Memory



Temporary data: return address, local var

(Optional) memory dynamically allocated during execution (malloc/free)

Constants and global var

Program code

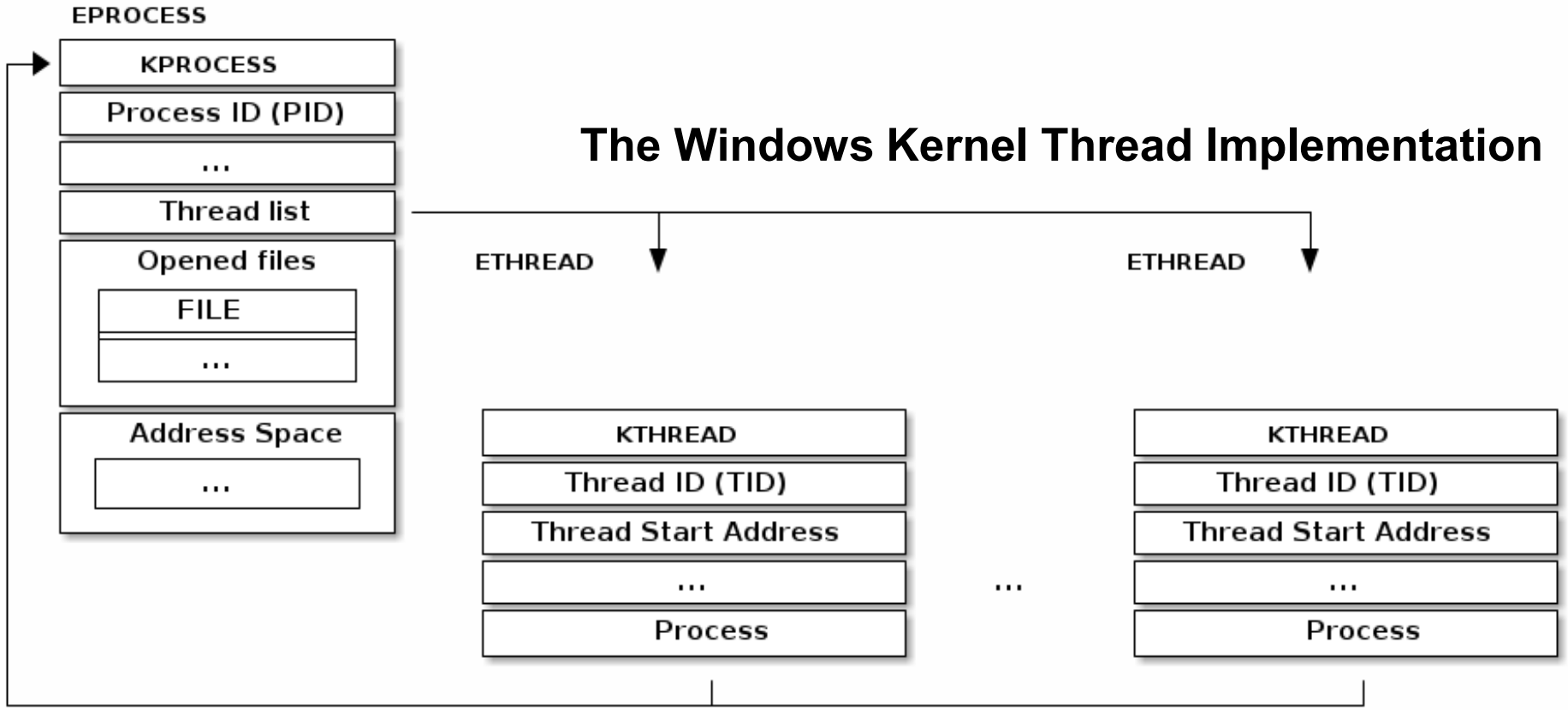
Threads

Threads

- The basic unit the kernel process scheduler uses to allow applications to run the CPU
- Each thread has its own stack
  - Combined with the register values it determines the thread execution state
- A thread runs in the context of a process and all threads in the same process share the resources
- The kernel schedules threads not processes and user-level threads (e.g. fibers, coroutines, etc.) are not visible at the kernel level

Threads

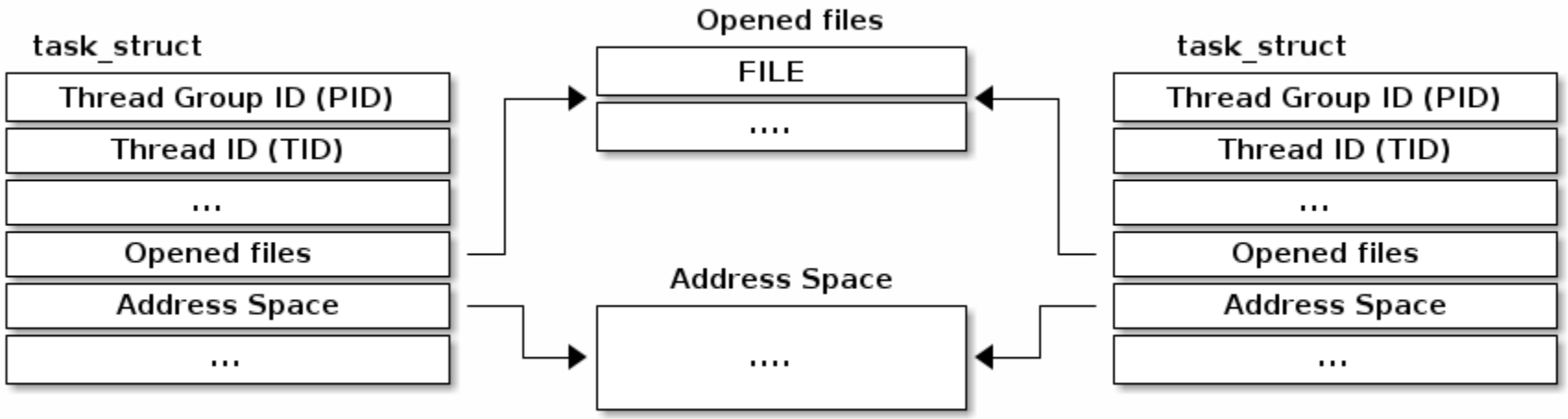
Common thread implementation as a separate data structure then linked to the process data structure



Data modified from information found here: https://linux-kernel-labs.github.io/refs/heads/master/lectures/processes.html

Linux Implementation for Threads

- The basic unit is called a ***task*** (hence the **struct task_struct**) and it is used for both tasks and processes.
- Uses pointers to resources in the task structure
- If two threads are the same process will point to the same resource structure instance.
- If two threads are in different processes they will point to different resource structure instances.

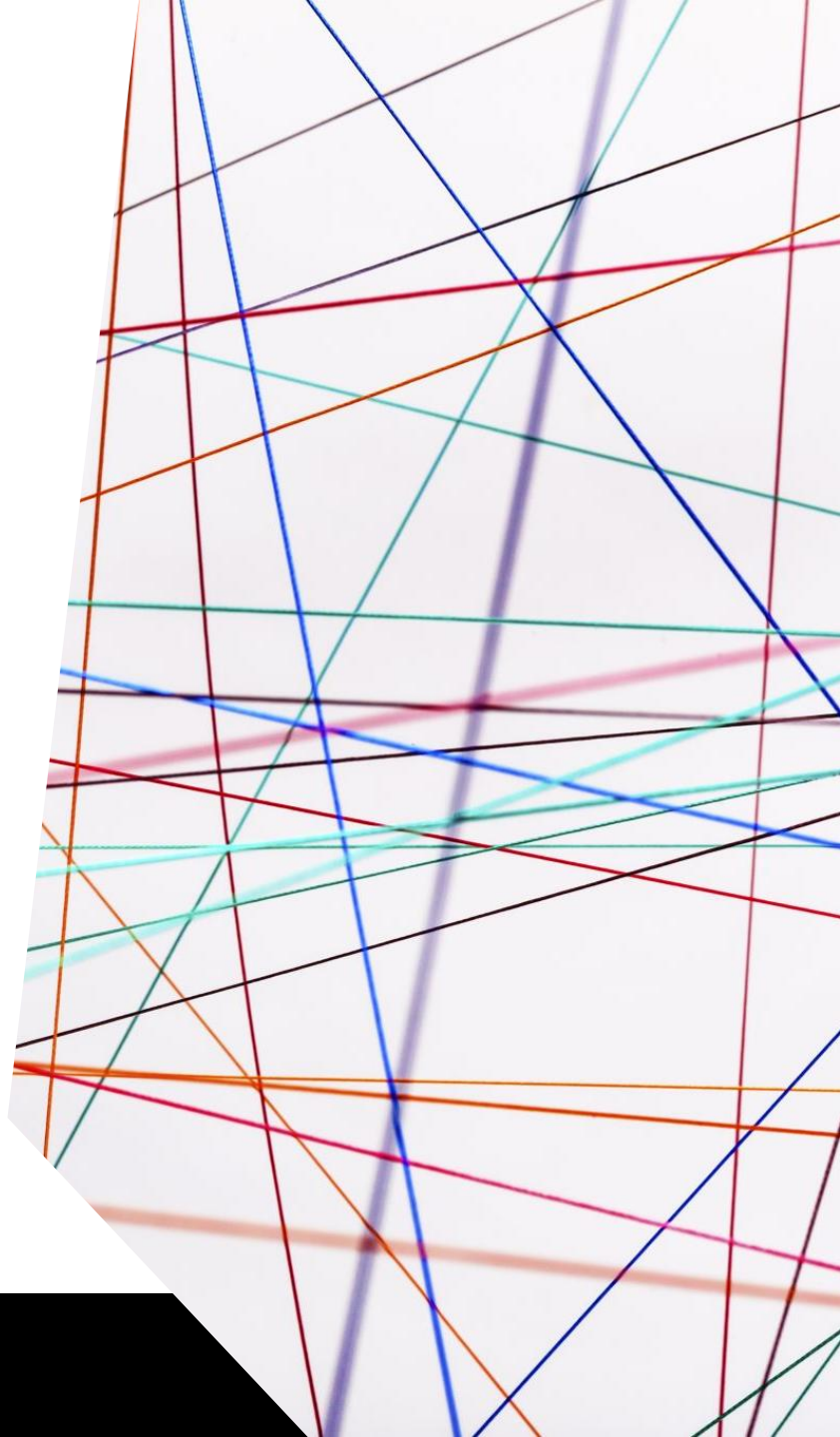


Data modified from information found here: https://linux-kernel-labs.github.io/refs/heads/master/lectures/processes.html

Threads

Each process has an address space and a single thread of control

Often, it is desirable to have multiple threads of control in the same address space running in quasi-parallel, as though they were (almost) separate processes (except for the shared address space)



Thread Usage

- Main reason for having threads is that in many applications, multiple activities are going on at once.
- Decomposing an application into multiple sequential threads that run in quasi-parallel, the programming model becomes simpler

Four Reasons for Using Threads

1. The ability for the parallel entities to share an address space and its data among themselves. This ability is essential for certain applications, which is why having multiple processes (with their separate address spaces) will not work.

2. Threads are lighter weight than processes, they are faster to create and destroy.

3. Performance: Threads yield no performance gain when all of them are CPU bound, but when there is substantial computing and substantial I/O, having threads allows these activities to overlap

4. Threads are useful on systems with multiple CPUs, where real parallelism is possible.

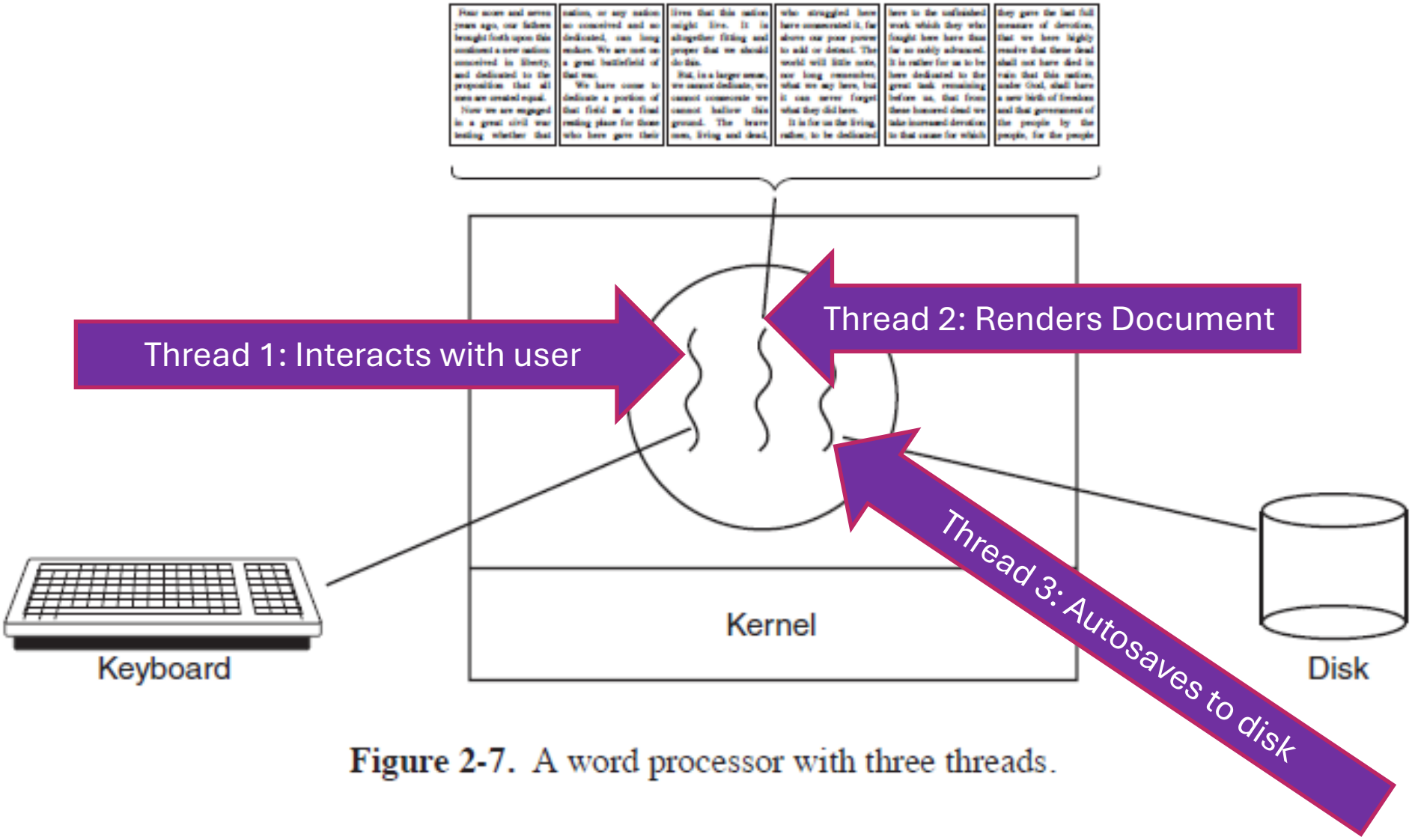


Figure 2-7. A word processor with three threads.

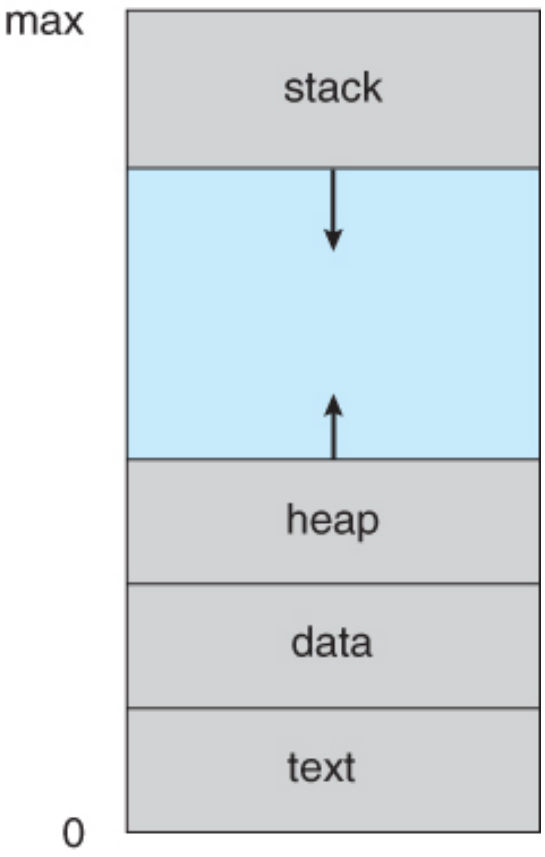
Multithreading

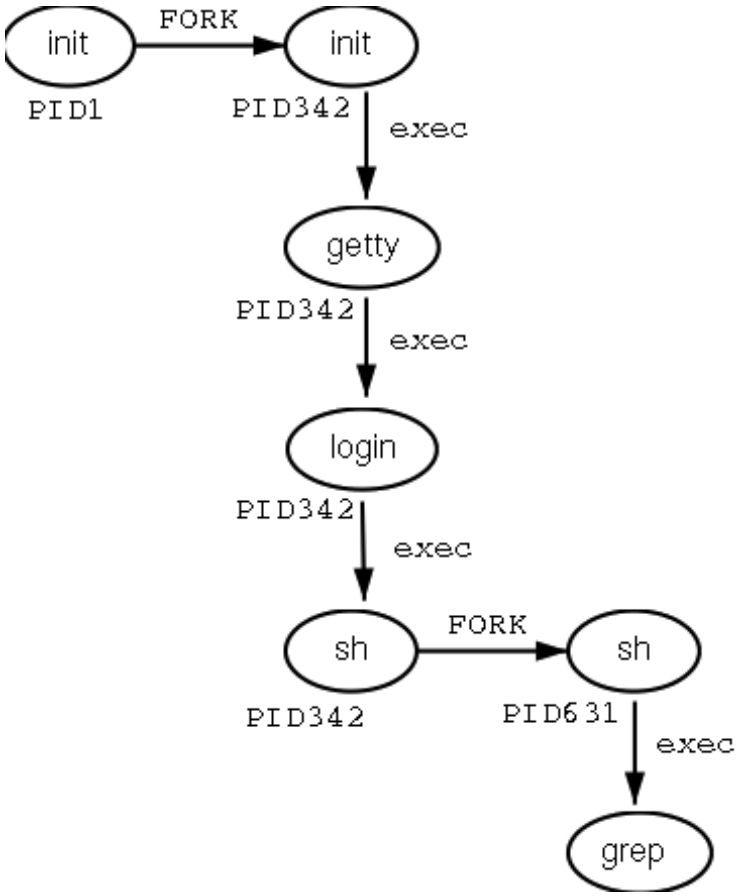
- With three threads, the programming model is much simpler.
- The first thread interacts with the user. The second thread reformats the document when told to.
- The third thread writes the contents of RAM to disk periodically.

Thread and Multithreading

Process

- Resource grouping and execution





Single Threaded

- Only one task at a time could be run.

Multithreading

- Multithreading is the ability of a program or an operating system to enable more than one user at a time
- No need for multiple copies of the program running on the compute
- Can also handle multiple requests from the same user

Thread and Multithreading

- Each one is doing something
- They **share** the same data but look at different parts
- They have **private** state but can communicate easily
- They must coordinate!

Thread and Multithreading

- Process
  - Resource grouping and execution
- Thread
  - A program in execution without dedicated address space: *threads of the same process share address space*
- Efficient communication
  - Inter-**thread** communication can be carried out via shared data objects within the shared address space
  - Inter-**process** communication usually requires other OS services
- Efficient creation
  - Only create thread context

Interprocess Communication

- Processes often need to communicate with other processes or services in the Operating System
- Example: In a shell pipeline, the output of the first process must be passed to the second process, and so on down the line
- Communicate in a well-structured way not using interrupts

Three Issues with Interprocess Communication

1. One process can pass information to another
2. Ensure two or more processes do not get in each other's way
3. Proper sequencing when dependencies are present: if process *A* produces data and process *B* prints them, *B* has to wait until *A* has produced some data before starting to print.

Three Issues with Interthread Communication

Two of the same issues as interprocess communication

Passing information is easy for threads since they share a common address space

- Threads in different address spaces that need to communicate fall under the heading of communicating processes

However, the other two—keeping out of each other's hair and proper sequencing also apply to threads

Processes vs Threads: A Closer Look

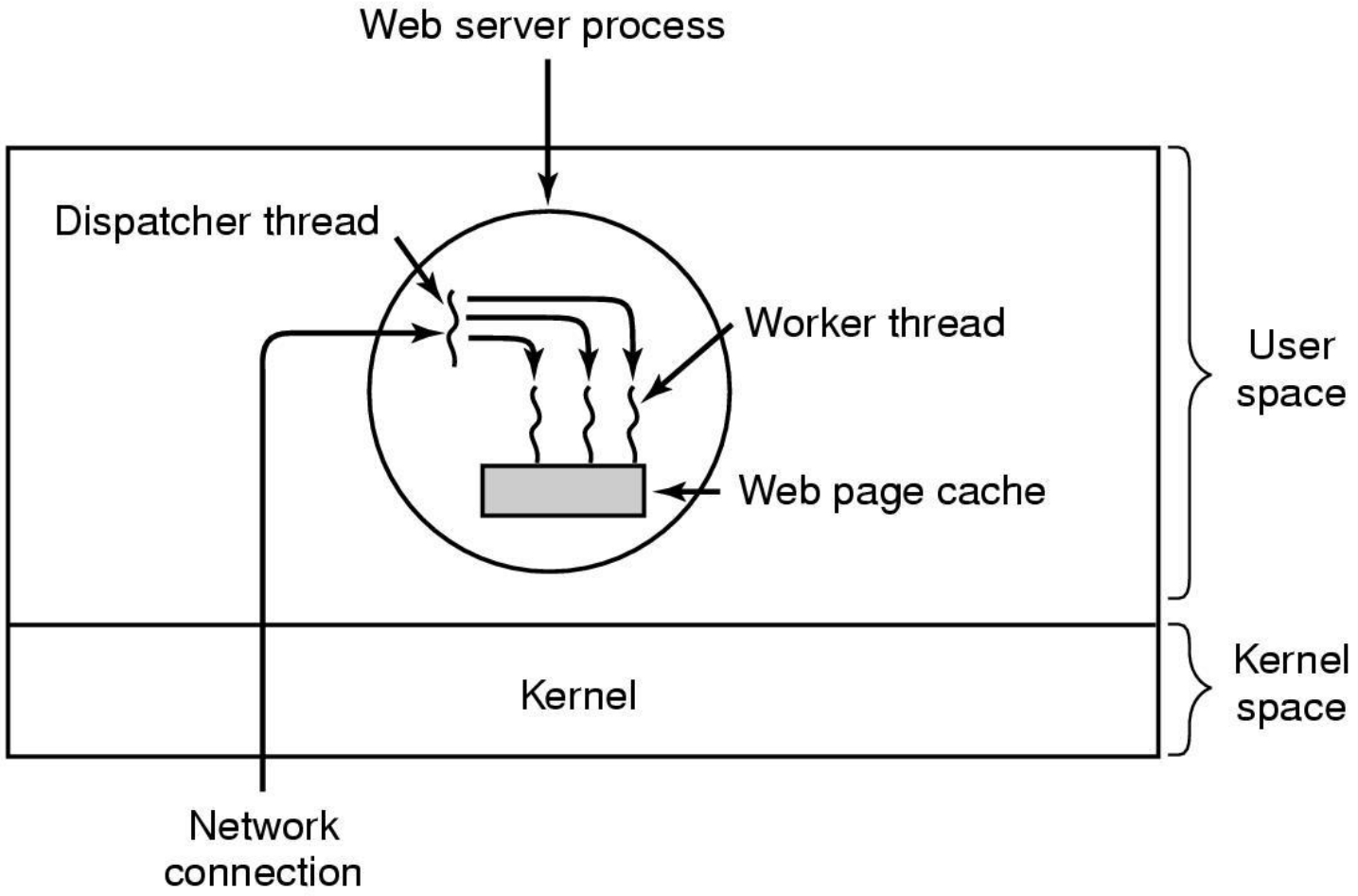
Threads

- No data segment or heap
- Multiple can coexist in a process
- Share code, data, heap, and I/O
- Have own stack and registers
- Inexpensive to create
- Inexpensive context switching
- Efficient communication

Processes

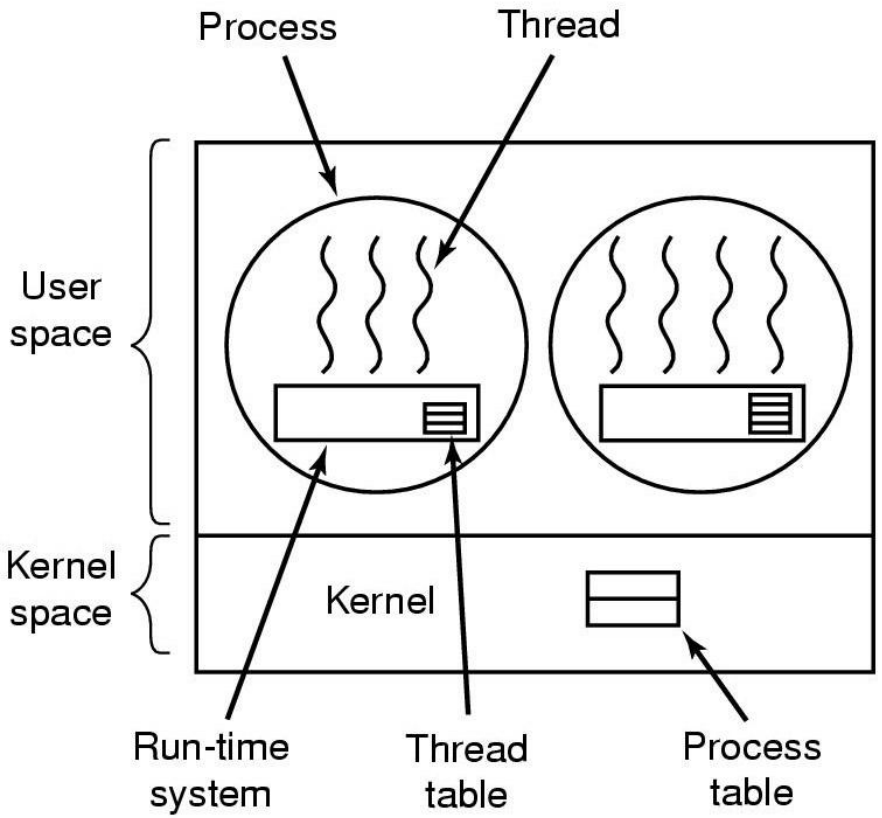
- Have data/code/heap
- Include at lease one thread
- Have own address space, isolated from other processes
- Expensive to create
- Expensive context switching
- IPC can be expensive

Thread Usage



A multithreaded Web server

Implementing Threads in User-Space



A user-level threads package

- OS thinks there's only a single-threaded process
- Threads in same process don't involve multiplexing between processes so no kernel privilege required
- User-level threads: the kernel knows nothing about them

User-level Thread - Discussions

- Advantages
  - No OS thread-support needed
  - Lightweight: thread switching vs. process switching
    - Local procedure vs. system call (trap to kernel)
  - Each has its own customized scheduling algorithms
    - thread_yield()

User-level Thread - Discussions

- Advantages
  - No OS thread-support needed
  - Lightweight: thread switching vs. process switching
    - Local procedure vs. system call (trap to kernel)
  - Each has its own customized scheduling algorithms
    - thread_yield()
- Disadvantages
  - How blocking system calls implemented? Called by a thread?
    - Goal: to allow each thread to use blocking calls, but to prevent one blocked thread from affecting the others
  - How to deal with page faults?
  - How to stop a thread from running forever?
    - No clock interrupts in a single process

Implementing Threads in the Kernel

- Threads known to OS
  - Scheduled by the scheduler
- Slow
  - Trap into the kernel mode
- Expensive to create and switch
  - Less expensive if in the same process
  - Registers, PC, stack pointer need to be created/changed
  - Not the memory info
- Kernel-level threads: when a thread blocks, kernel re-schedules another thread

Any problems?

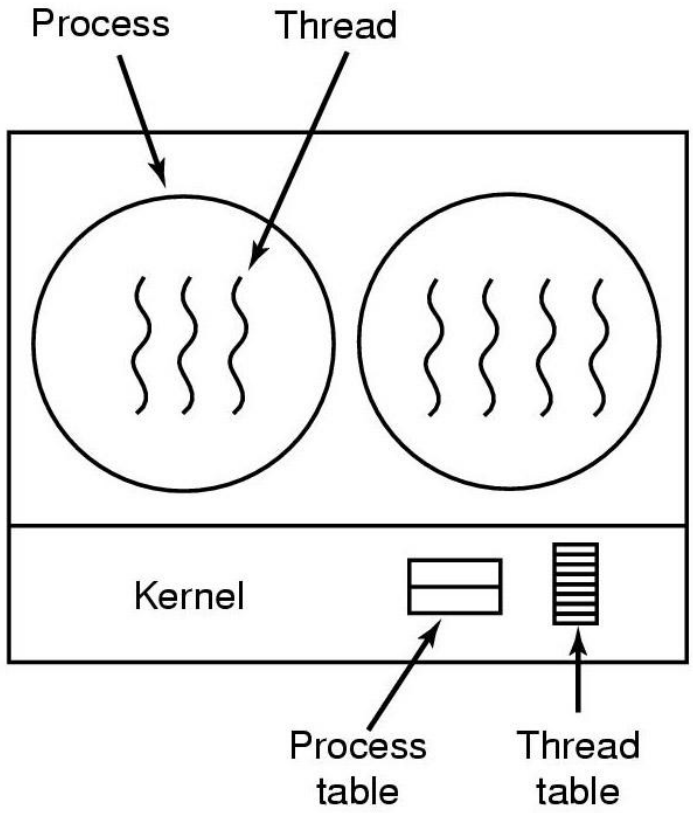
A threads package managed by the kernel

What happens when forking a multithreaded process

Implementing Threads in the Kernel

Any problems?

What happens when forking a multithreaded process?



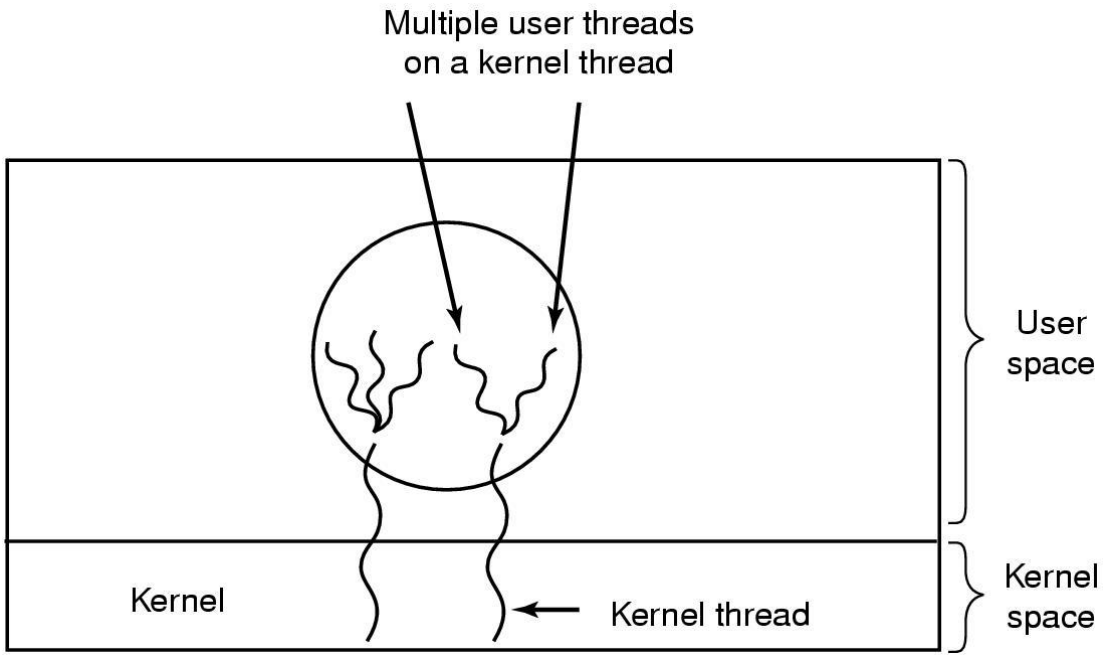
A threads package managed by the kernel

Hybrid Implementations

Use kernel-level threads and then *multiplex* user-level threads onto some or all of the kernel-level threads

- Multiplexing user-level threads onto kernel-level threads
- Enjoy the benefits of user and kernel level threads
- Too complex!

Hybrid Implementations



Multiplexing user-level threads onto kernel-level threads

Threading Models

N:1 (User-level threading)

- N user threads that look like 1 to the OS kernel

1:1 (Kernel-level threading)

- Each user-thread is paired with a kernel-thread (aka native thread)

M:N (Hybrid threading)

- Solaris

Threads in Linux

Thread control block (TCB)

- The `thread_stuct` structure
- Includes registers and processor-specific context

Linux treats threads like processes

- Use `clone()` to create threads instead of using fork()
- `clone()` is usually not called directly, but from some threading libraries

Summary

Processes v.s. threads?

Why threads?

- Concurrency + lightweight

Threading models

- N:1, 1:1, M:N

Additional practice

- Find out what threading model does Java belong to
- Write (download) a simple multithreaded java program
- When it is running, issue `ps -eLf | grep YOUR_PROG_NAME`

For Next Time

Read: Andrew S. Tanenbaum, Modern Operating Systems, 4thd edition, 2014, Prentice Hall Chapter 2

Start Project 2

Course Content

Unit	Module	Week	Topic
1 – Introduction	1 & 2	1	Welcome to OS & Introduction
	2	2	OS Overview
2 – Processes and Threads	3	3	Processes
	4	4	Threads
	5	5	CPU Scheduling
	6	6	Multiprocessor Scheduling
	7	7	Interprocess Communication
	8	8	Pthread
3 – Deadlocks & Memory	9	9	Deadlock
	10	10	Memory Management
	--	11	Midterm Exam
4 – Additional Topics	11	11	Page Replacement
	12	13	File Systems
	13	14	IO Devices
	14	15	Security
	--	16	Final Exam