CSE 3320
Operating Systems
Threads

Jia Rao

Department of Computer Science and Engineering

http://ranger.uta.edu/~jrao

Recap of the Last Class

- Processes
  - A program in execution
  - 5 (3)-state process model
  - Process control block
    - Execution flow
    - Resources
- Linux processes
  - The `task_struct` structure
  - Which field stores the program counter ?
    - The `thread` field
    - Saving hardware registers can not be expressed in C, but in assembly
    - It is a processor-specific context
      - SRC/arch/x86/include/asm/processor.h
      - Field `ip` in struct thread_struct

Thread and Multithreading

- Process
  - resource grouping and execution
- Thread
  - a finer-grained entity for execution and parallelism
  - Lightweight process
  - A program in execution without dedicated address space
- Multithreading
  - Running multiple threads within a single process

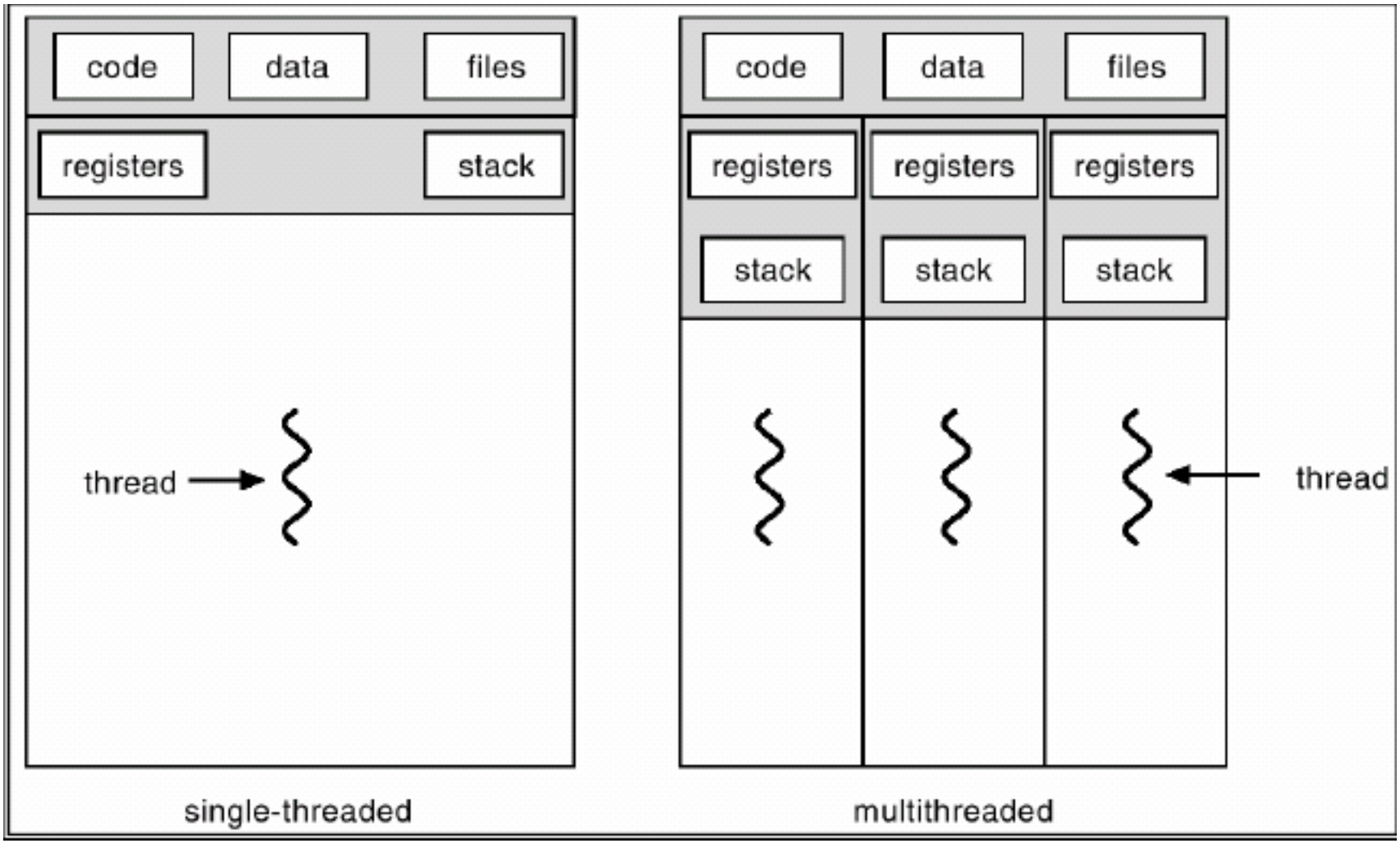
Processes v.s. Threads

- Process
  - Concurrency
    - Sequential execution stream of instructions
  - Protection
    - A dedicated address space
- Threads
  - Separate concurrency from protection
  - Maintain sequential execution stream of instructions
  - Share address space with other threads

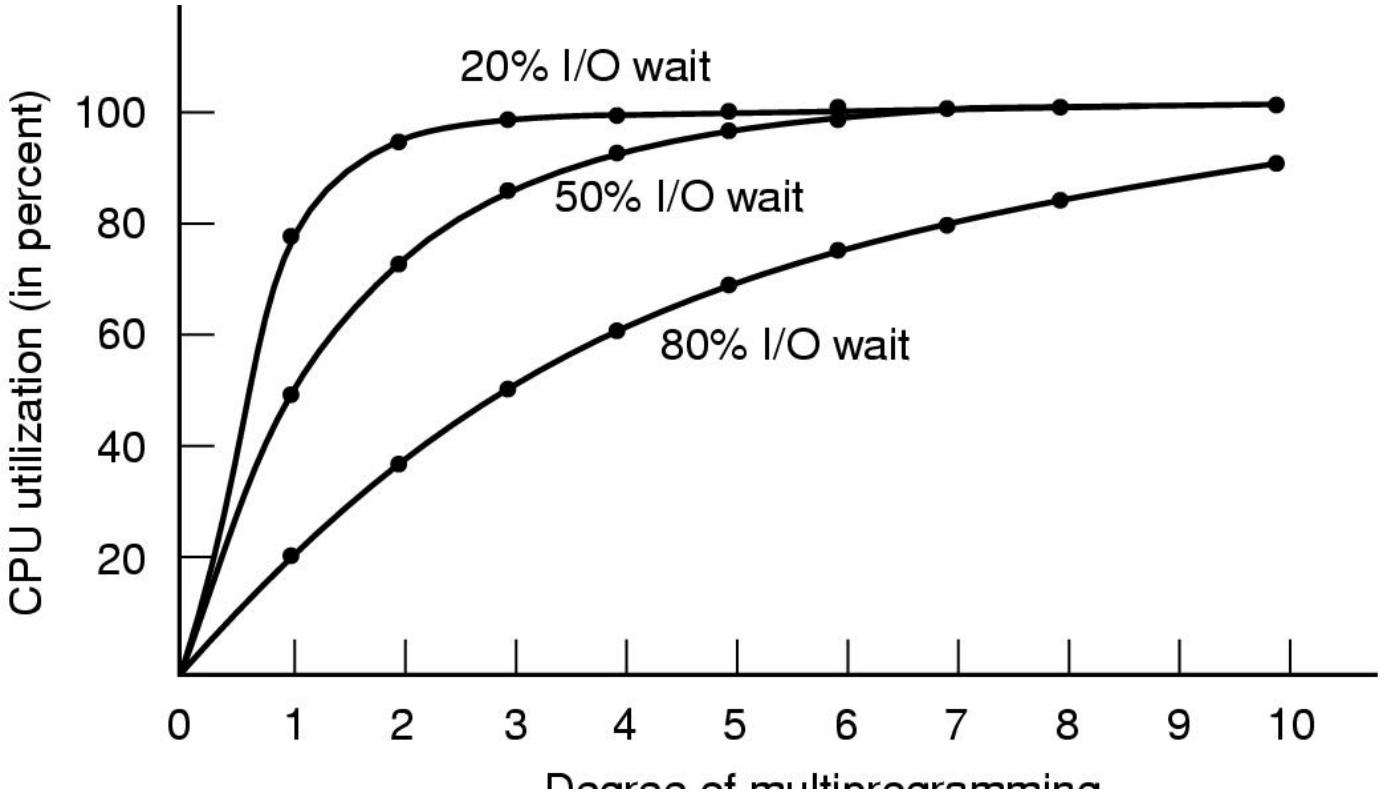
A Closer Look

- 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

An Illustration



Why Multiprogramming ?



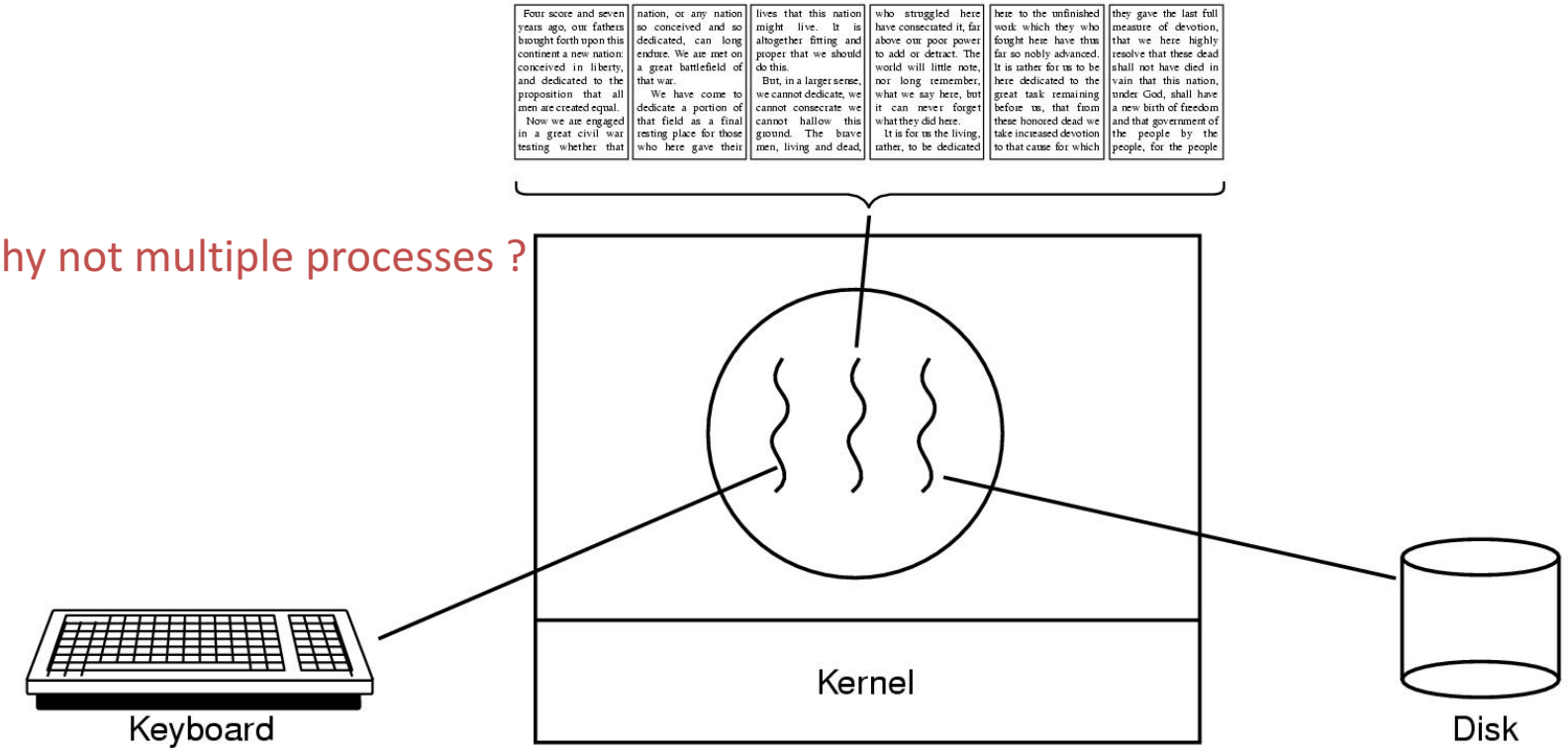
CPU utilization as a function of the number of processes in memory.

Why threads ?

- Express concurrency
  - There are other ways to explore concurrency (e.g., non-blocking I/O), but they are difficult to program
- Efficient communication
  - Communication can be carried out via shared data objects within the shared address space
  - Inter-process communication usually requires other OS services: file system, network system
- Efficient creation
  - Only create the thread context

Thread Usage 1

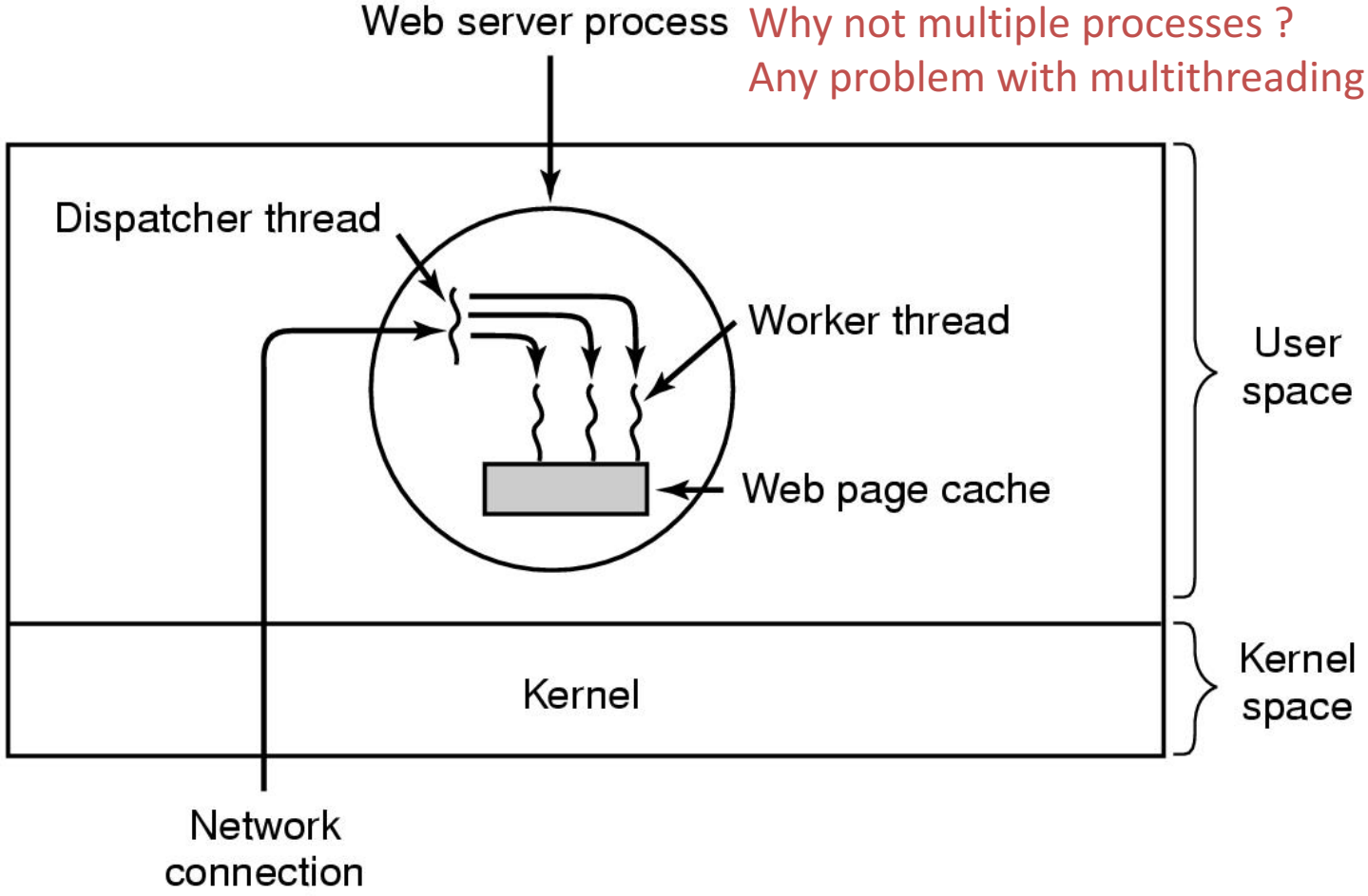
Why not multiple processes ?



A word processor with three threads.

Thread Usage 2

Why not multiple processes ?
Any problem with multithreading ?



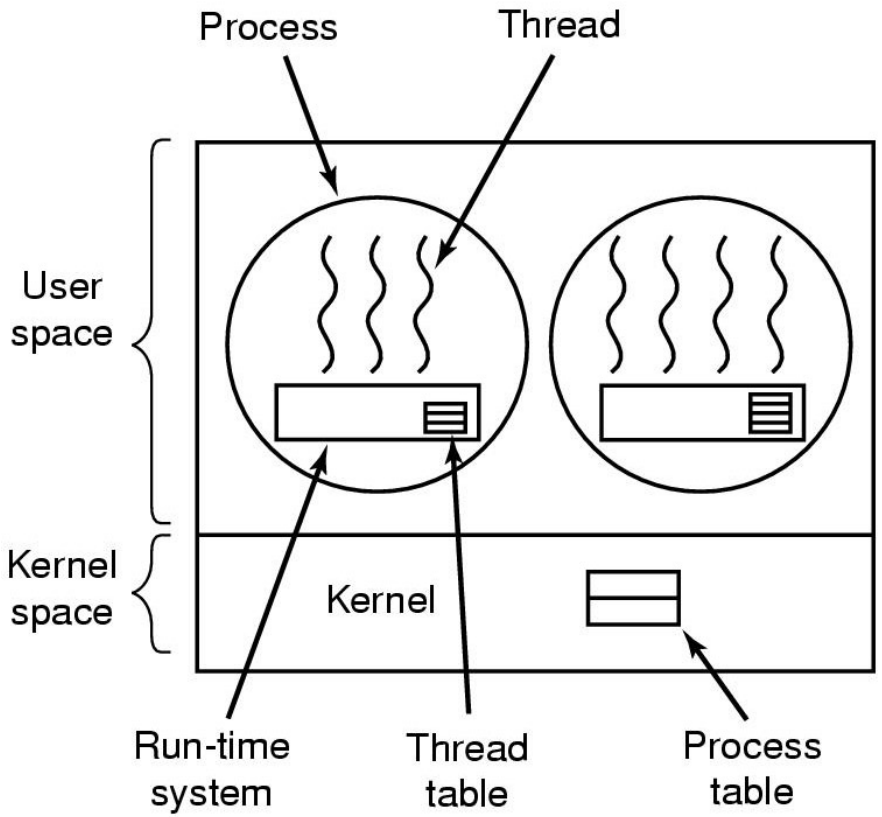
A multithreaded Web server.

A Simple Implementation

```
Void *worker(void *arg) // worker thread
{
          unsigned int socket;
          socket = *(unsigned in *)arg;
          process (socket);
          pthread_exit(0);
}
int main (void) // main thread, or dispatcher thread
{
           unsigned int server_s, client_s, i=0;
           pthread_t threads[200];
           server_s = socket(AF_INET, SOCK_STREAM, 0);
          ……
           listen(server_s, PEND_CONNECTIONS);
           while(1){
                     client_s = accept(server_s, …);
                     pthread_create(&threads[i++], &attr, worker, &client_s);
           }
}
```

Implementing Threads in User-Space

- User-level threads: the kernel knows nothing about them



A user-level threads package

User-level Thread - Discussions

- Advantages
  - No OS thread-support needed
  - Lightweight: thread switching vs. process switching
    - Local procedure vs. system call (trap to kernel)
    - When we say a thread *come-to-life*? SP & PC switched
  - Each process has its own customized scheduling algorithms
    - thread_yield()

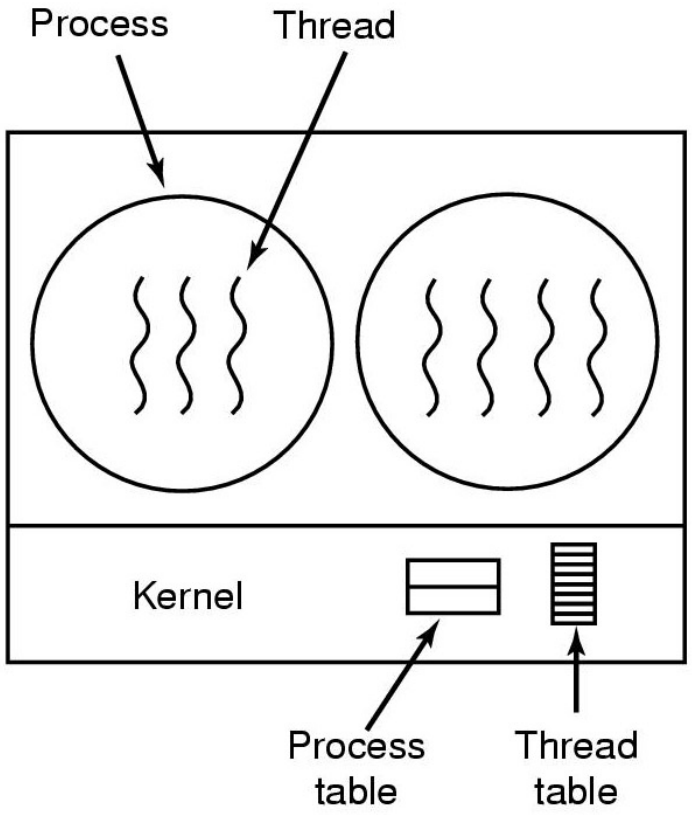
- Disadvantages
  - How blocking system calls implemented? Called by a thread?
    - Goal: to allow each thread to use blocking calls, but to prevent one blocked thread from affecting the others
  - How to change blocking system calls to non-blocking?
  - Jacket/wrapper: code to help check in advance if a call will block
  - How to deal with page faults?
  - How to stop a thread from running forever? No clock interrupts

Implementing Threads in the Kernel

- Kernel-level threads: when a thread blocks, kernel re-schedules another thread

  - 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 management info

  Any problem ?



A threads package managed by the kernel

Write a program that forks from a multithreaded process

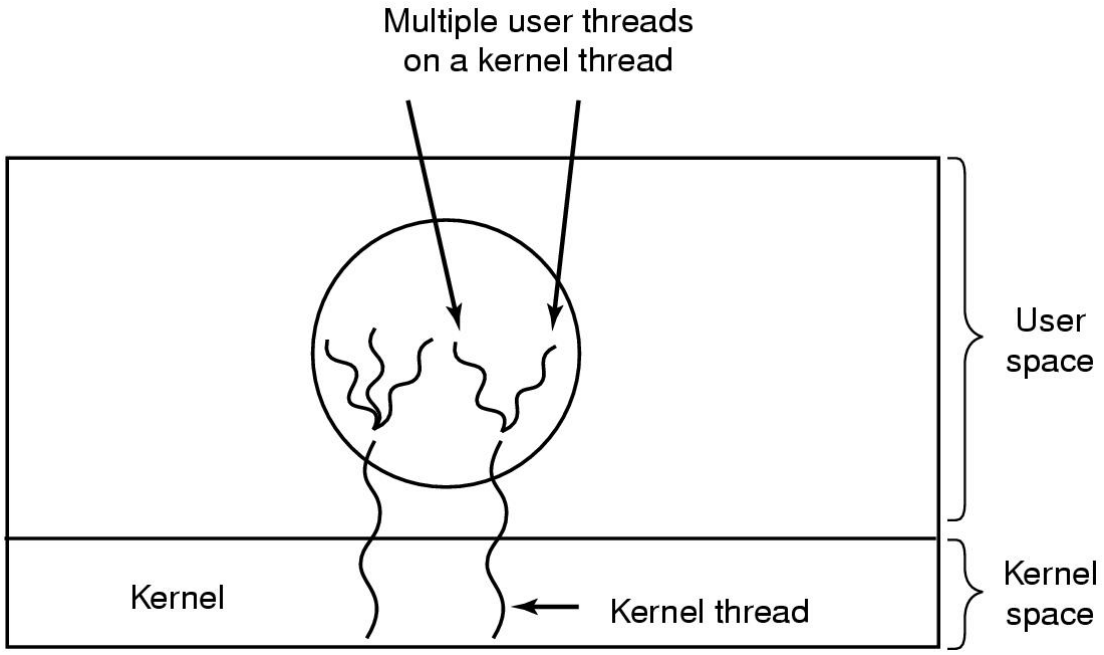
Hybrid Implementations

- 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 !
Any other problem ?

Priority inversion



Multiplexing user-level threads onto kernel-level threads

Threading Models

- N:1 (User-level threading)
  - GNU Portable Threads
- 1:1 (Kernel-level threading)
  - Native POSIX Thread Library (NPTL)
- M:N (Hybrid threading)
  - Solaris

An Example

```
void *my_thread(void *arg)
{
        int *tid = (int *)arg;
        printf("Hello from child thread: %d\n", *tid);
        return NULL;
}
int main(int argc, char *argv[]){
pthread_t threads[NR_THREADS];
        for (i = 1; i < NR_THREADS; i++){
                tid[i] = i;
                pthread_create(&threads[i], &a, my_thread, &tid[i]);
        }
        printf("Hello from the mother thread 0 !\n");
        for (i = 1; i < NR_THREADS; ++i)
        {
                pthread_join(threads[i], NULL);
        }
        return 0;
}
```

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, such as pthread.

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`
  - Download `glibc` at http://ftp.gnu.org/gnu/glibc/ and see the `nptl implementation`