CS330: Operating Systems

Threads

What is a thread?

- Threads are (almost!) independent execution entities of a single process

What is a thread?

- Threads are (almost!) independent execution entities of a single process
- Threads of a single process can be scheduled different CPUs in a concurrent manner.

What is a thread?

- Threads are (almost!) independent execution entities of a single process
- Threads of a single process can be scheduled different CPUs in a concurrent manner. Therefore,
    - Each thread has a different register state and stack
    - At a given point of time, PC of different threads can be different

What is a thread?

- Threads are (almost!) independent execution entities of a single process
- Threads of a single process can be scheduled different CPUs in a concurrent manner. Therefore,
    - Each thread has a different register state and stack
    - At a given point of time, PC of different threads can be different
- How threads are different from processes?

What is a thread?

- Threads are (almost!) independent execution entities of a single process
- Threads of a single process can be scheduled different CPUs in a concurrent manner. Therefore,
    - Each thread has a different register state and stack
    - At a given point of time, PC of different threads can be different
- How threads are different from processes?
    - Threads of a single process share the address space
    - Context switch between two threads of a process does not require switching the address space
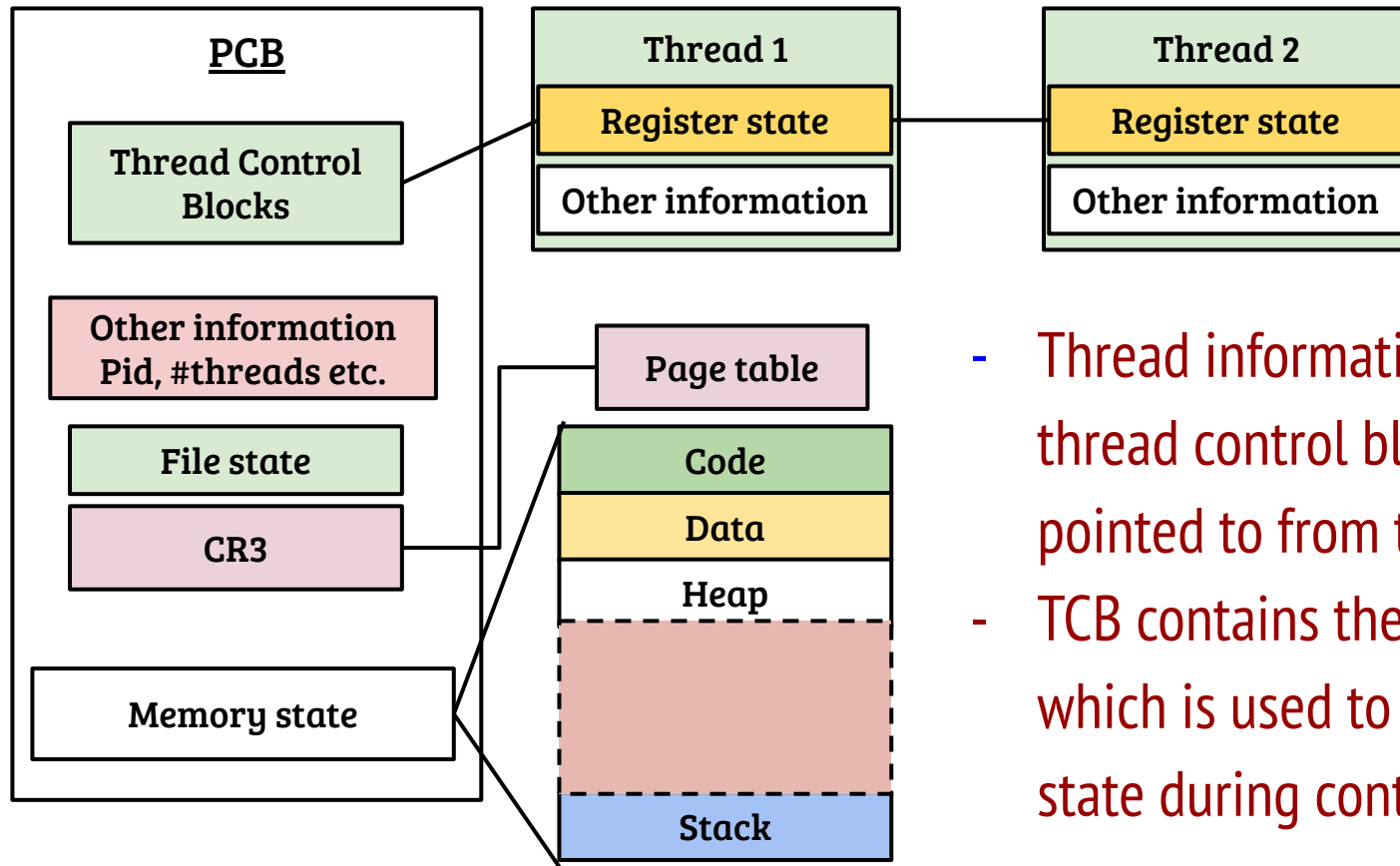
Multi-threaded processes

- Threads are (almost!) independent execution entities of a single process

- Why multithreading is useful?
- How does OS maintain thread related information?
- How stacks for multiple threads are managed?
- What is POSIX thread API? How is it used?

- How threads are different from processes?
    - Threads of a single process share the address space
    - Context switch between two threads of a process does not require switching the address space

Multi-threaded processes

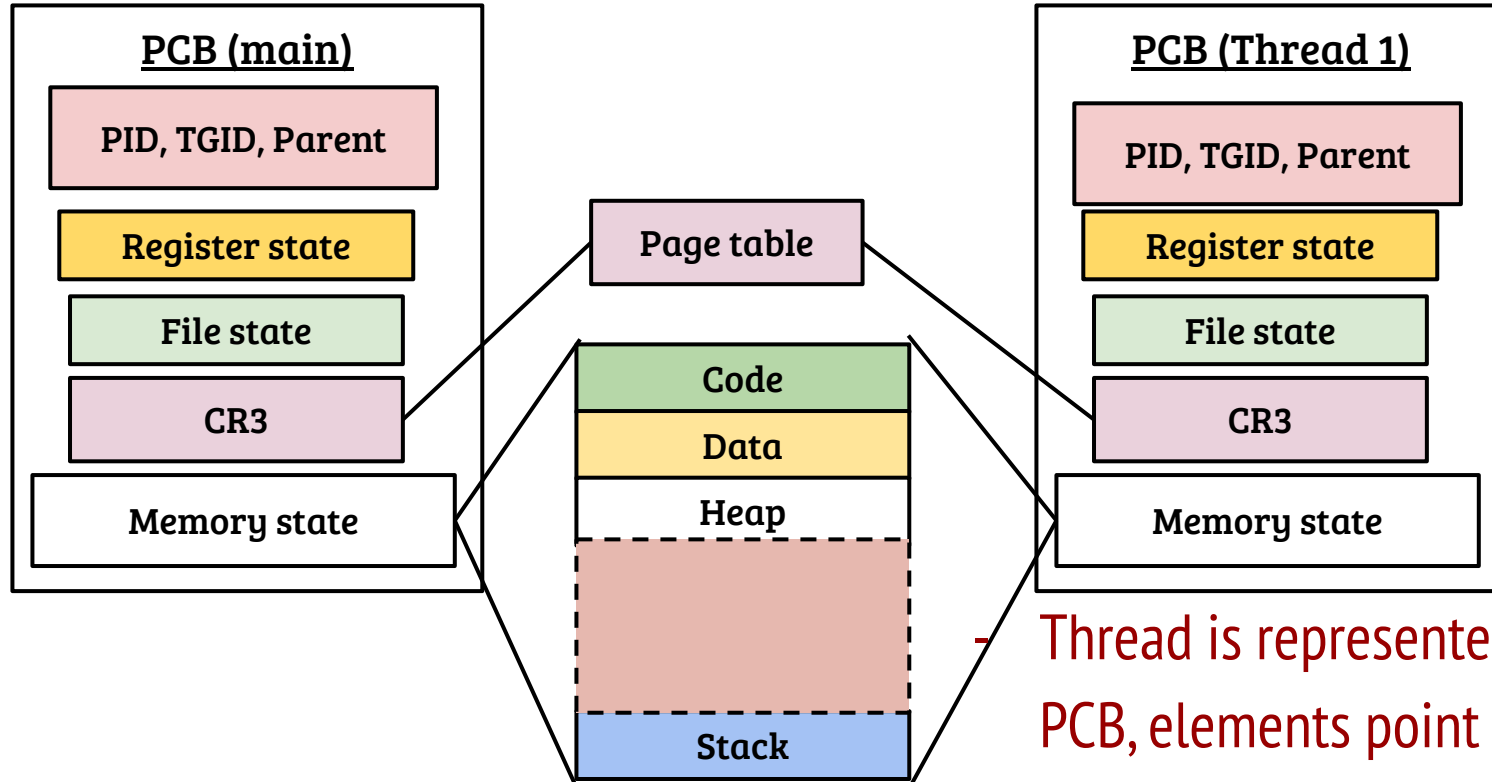Threads are (almost) independent execution entities of a single process

- Why multithreading is useful?
- Efficient execution on multicore systems, overlapping I/O and processing
- How does OS maintain thread related information?
- How stacks for multiple threads are managed?
- What is POSIX thread API? How is it used?

  - Threads of a single process share the address space
  - Context switch between two threads of a process does not require switching the address space

PCB of a multithreaded process

- Thread information is stored in thread control blocks (TCB) which is pointed to from the PCB
- TCB contains the register state which is used to save/restore CPU state during context switch

PCB of a multithreaded process (Linux)

| PCB (main)	Page table	PCB (Thread 1)
PID, TGID, Parent		PID, TGID, Parent
Register state	Code	Register state
File state	Data	File state
CR3	Heap	CR3
Memory state	Stack	Memory state

- Thread is represented by a separate PCB, elements point to the structure containing subsystem level info.

Multi-threaded processes

Threads are (almost) independent execution entities of a single process

- Why multithreading is useful?
- Efficient execution on multicore systems, overlapping I/O and processing
- How does OS maintain thread related information?
- Maintain thread information using separate PCB or using TCB
- How stacks for multiple threads are managed?
- What is POSIX thread API? How is it used?
  - Context switch between two threads of a process does not require switching the address space

Stack for multi-threaded processes
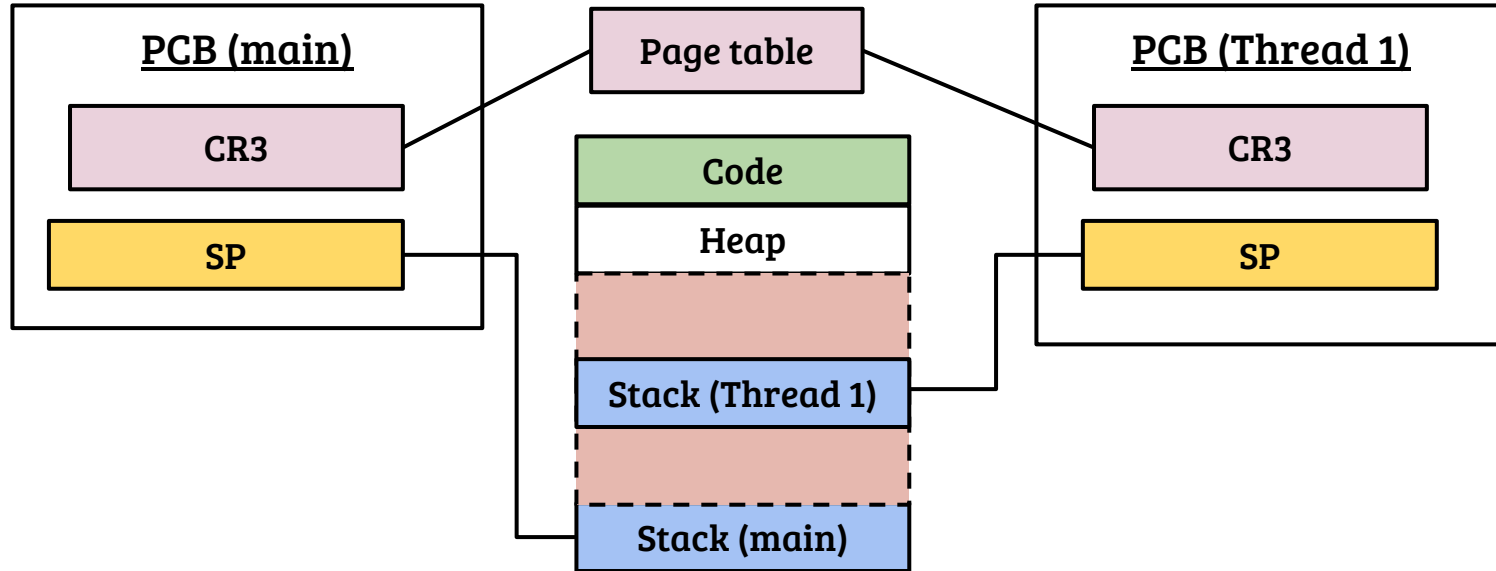
- Stack for threads dynamically allocated from the address space using mmap() system call and passed to the OS during thread creation

Multi-threaded processes

Threads are (almost) independent execution entities of a single process

- Why multithreading is useful?
- Efficient execution on multicore systems, overlapping I/O and processing
- How does OS maintain thread related information?
- Maintain thread information using separate PCB or using TCB
- How stacks for multiple threads are managed?
- Stacks for threads are allocated using memory allocation APIs
- What is POSIX thread API? How is it used?

switching the address space

Posix thread API (pthread_create)

int pthread_create(pthead_t *tid, pthread_attr_t *attr,
                    void * (*thfunc) (void*), void *arg);

- Creates a thread with "tid" as its handle and the thread starts executing the function pointed to by the "thfunc" argument
- A single argument (of type void *) can be passed to the thread
- Thread attribute can be used to control the thread behavior e.g., stack size, stack address etc. Passing NULL sets the defaults
- Returns 0 on success.
- Thread termination: return from thfunc, pthread_exit( ) or pthread_cancel( )
- In Linux, pthread_create and fork implemented using clone( ) system call

Posix thread API (pthread_join)

int pthread_join(pthead_t tid, void **retval)

- This call waits for the thread with handle "tid" to finish
- The return value of the thread is captured using the "retval" argument
  - The thread must allocate the return value which is freed after the process joins
- Invoking pthread_join for an already finished thread returns immediately

Multi-threaded processes

- Why multithreading is useful?
- Efficient execution on multicore systems, overlapping I/O and processing
- How does OS maintain thread related information?
- Maintain thread information using separate PCB or using TCB
- How stacks for multiple threads are managed?
- Stacks for threads are allocated using memory allocation APIs
- What is POSIX thread API? How is it used?
- Easy to use thread library with OS support. Important APIs: pthread_create, pthread_join