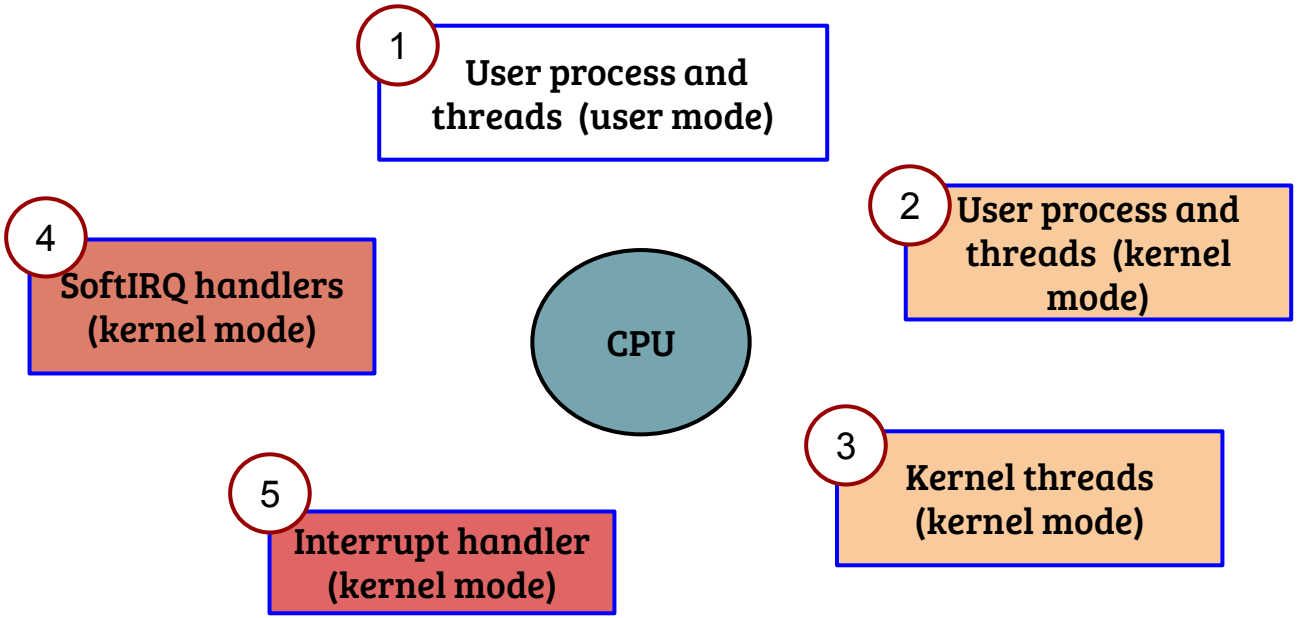
CS614: Linux Kernel Programming

Process, Thread, Kernel Threads

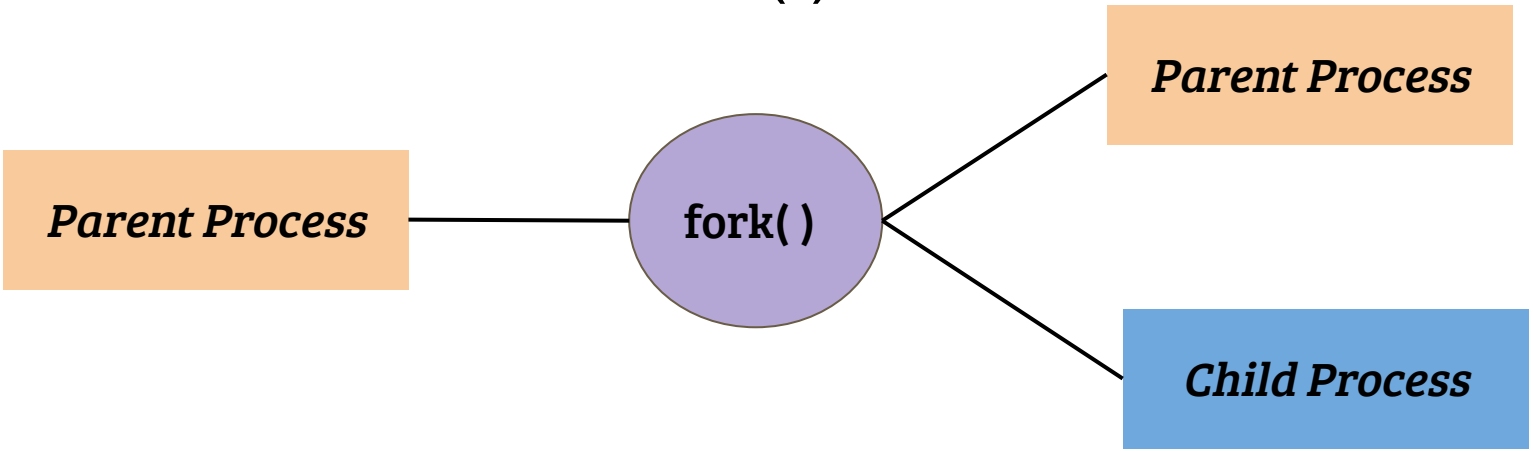
Debadatta Mishra, CSE, IIT Kanpur

Recap: Execution contexts in Linux



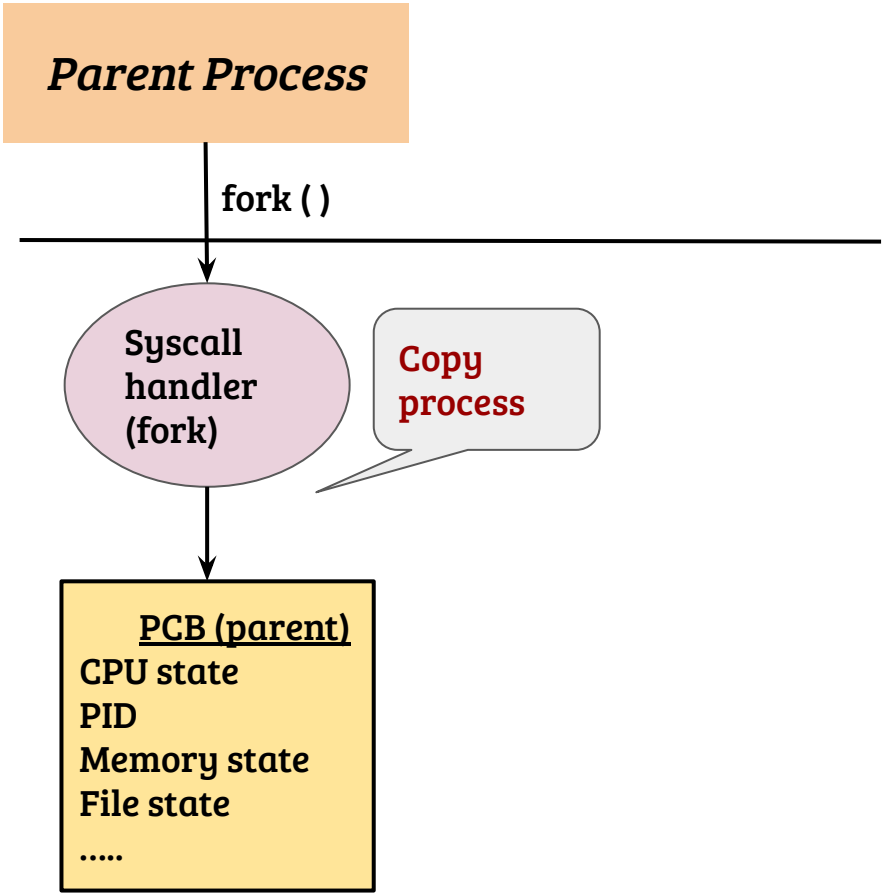
- In a linux system, the CPU can be executing in one of the above contexts
- For (3), (4) and (5), the context is not associated with any user process

Process creation - fork( )

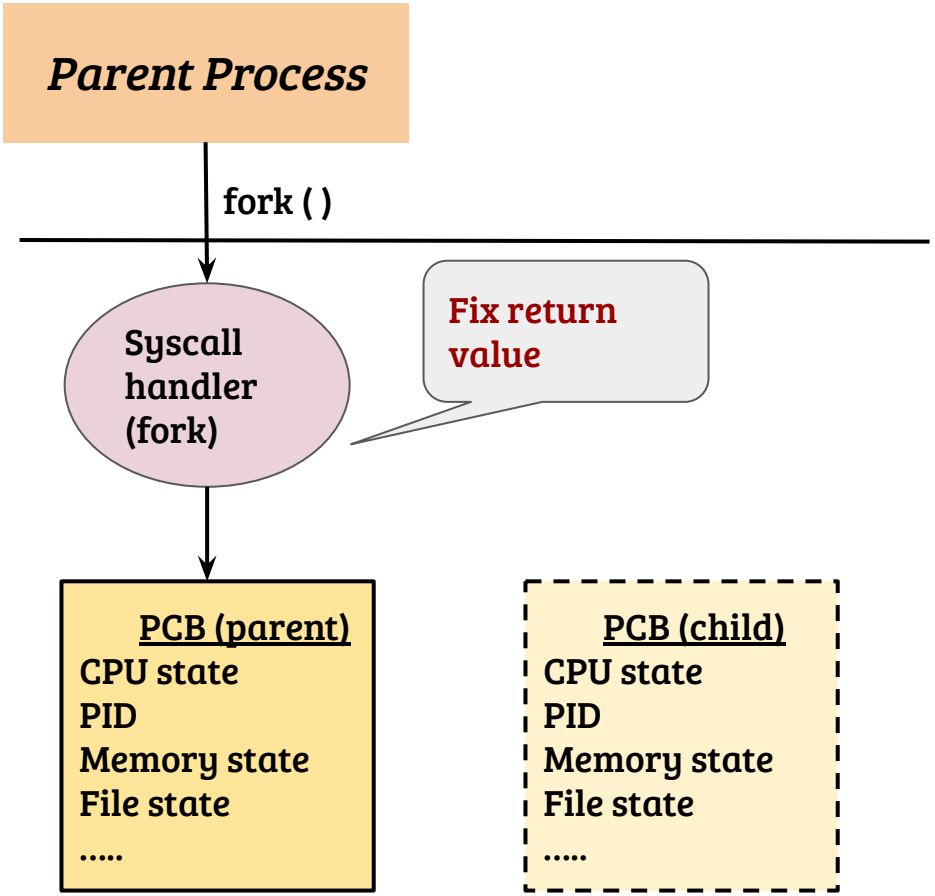


- fork( ) system call is weird; not a typical "privileged" function call
- fork( ) creates a new process; a *duplicate* of calling process
- On success, fork
  - Returns PID of child process to the caller (parent)
  - Returns 0 to the child

Typical implementation of fork

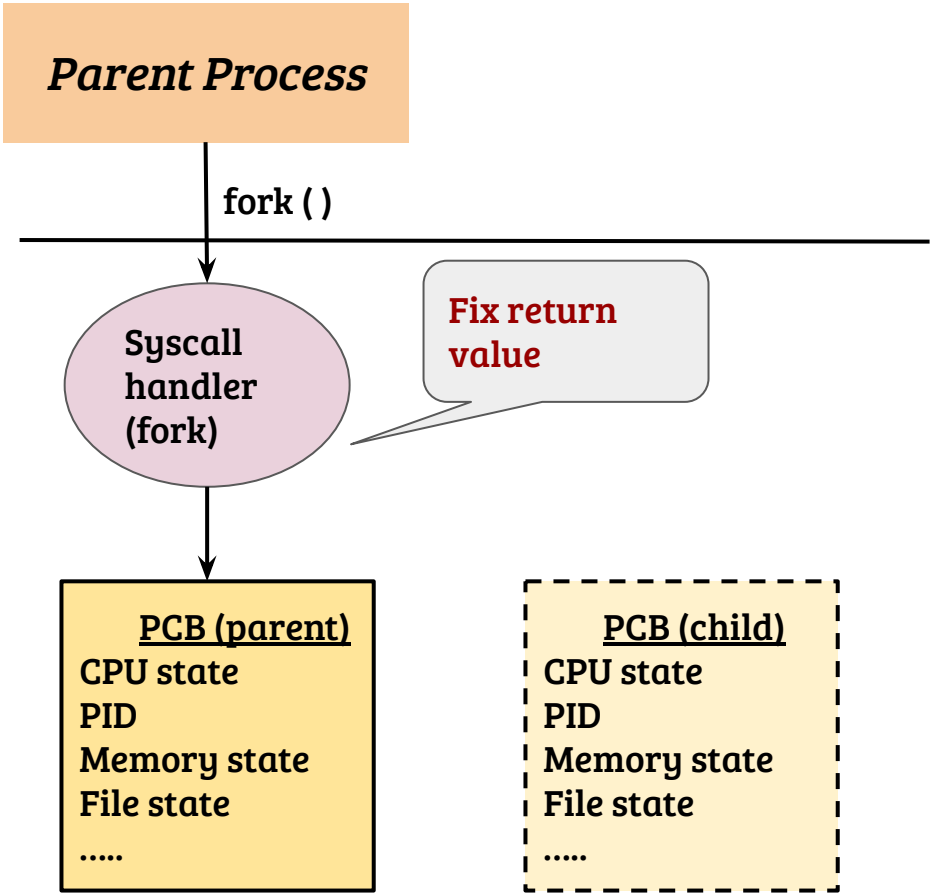


Typical implementation of fork



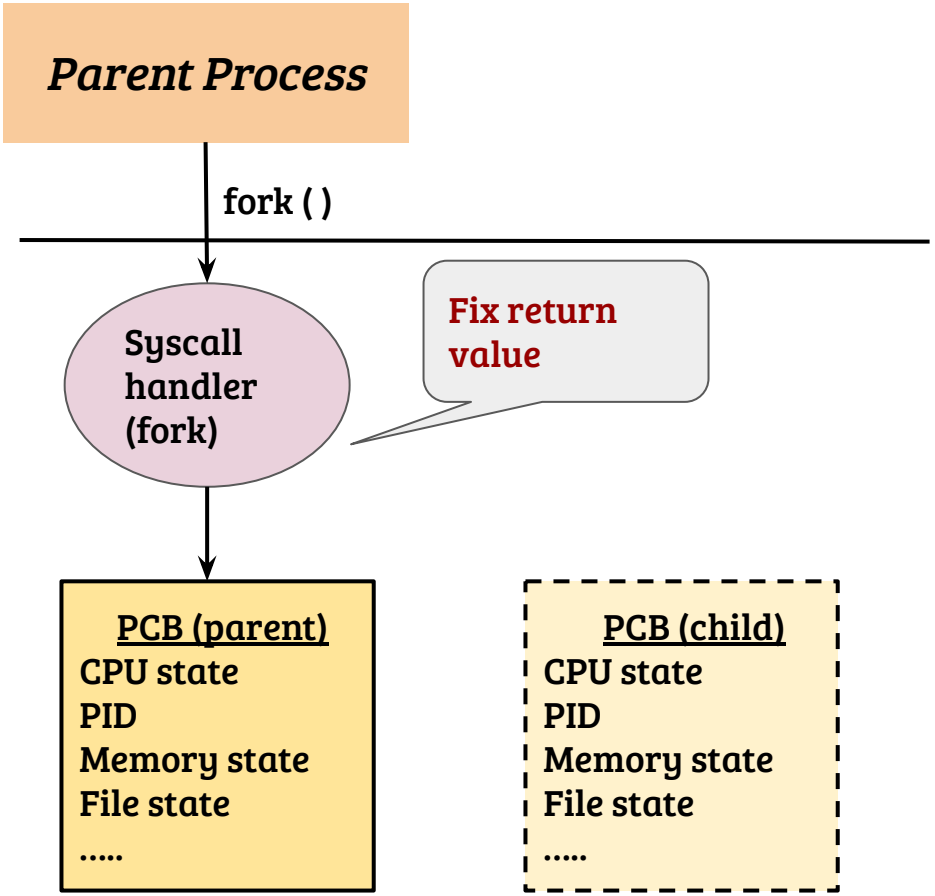
- Child should get '0' and parent gets PID of child as return value. How?

Typical implementation of fork



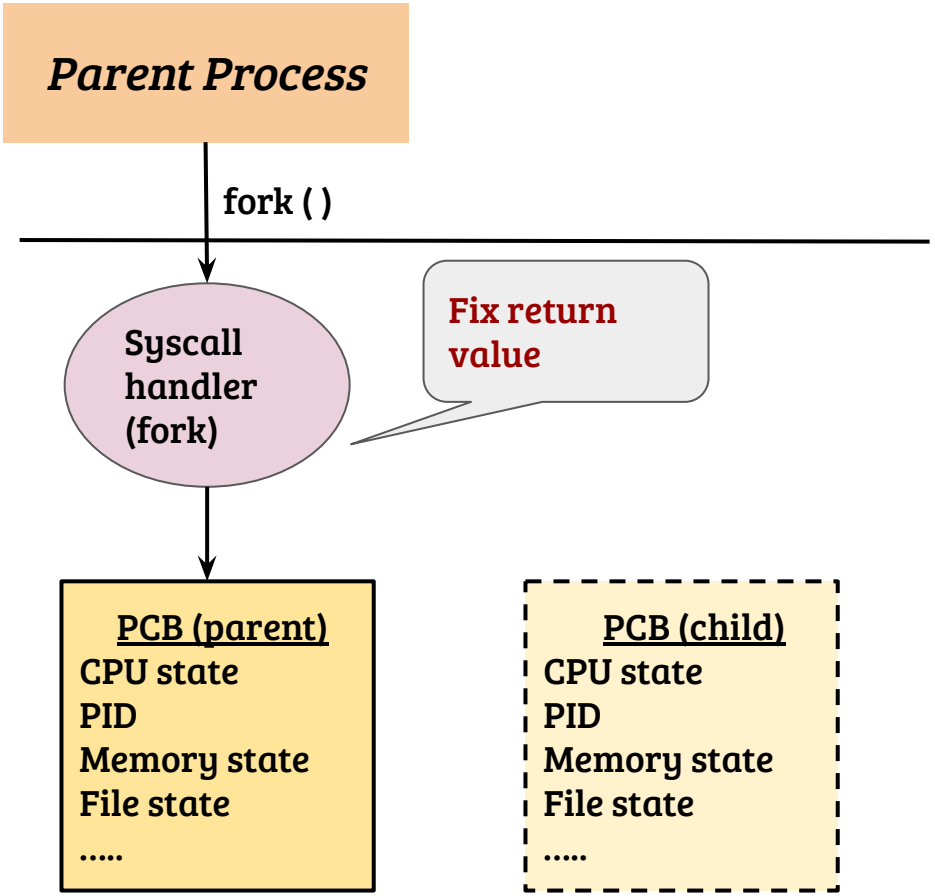
- Child should get '0' and parent gets PID of child as return value. How?
- OS returns different values for parent and child

Typical implementation of fork



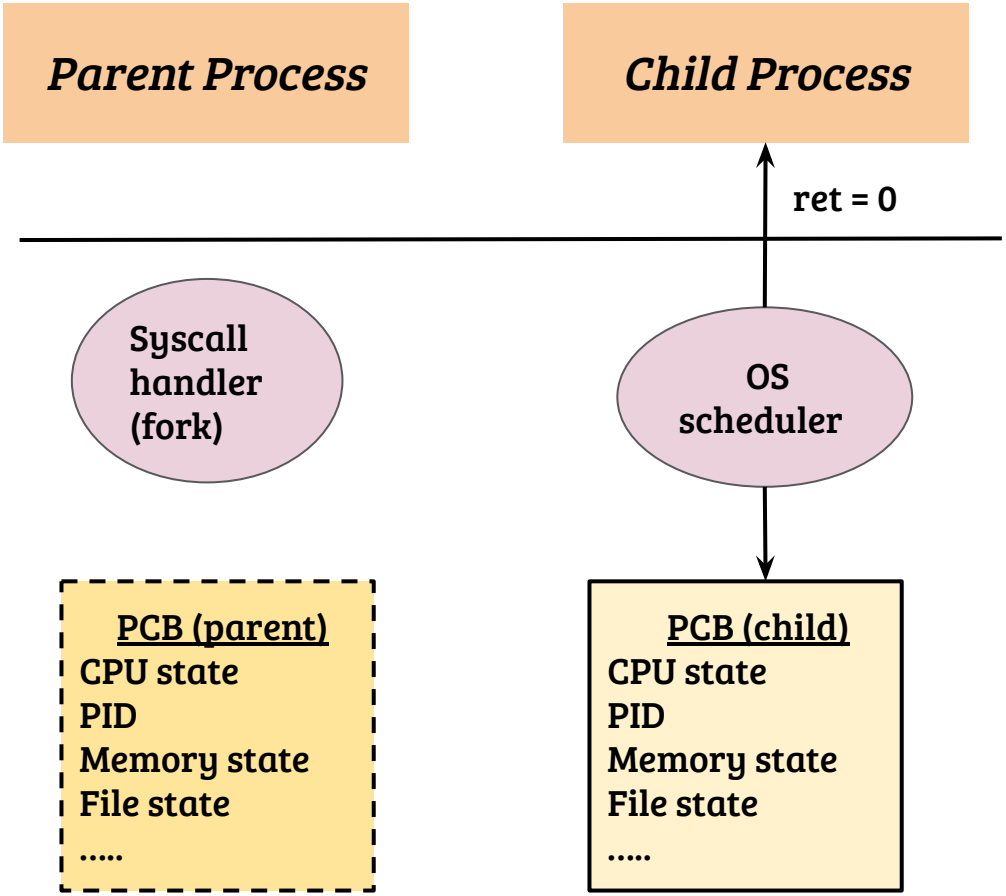
- Child should get '0' and parent gets PID of child as return value. How?
- OS returns different values for parent and child
- When does child execute?

Typical implementation of fork



- Child should get '0' and parent gets PID of child as return value. How?
- OS returns different values for parent and child
- When does child execute?
- When OS schedules the child process

Typical implementation of fork



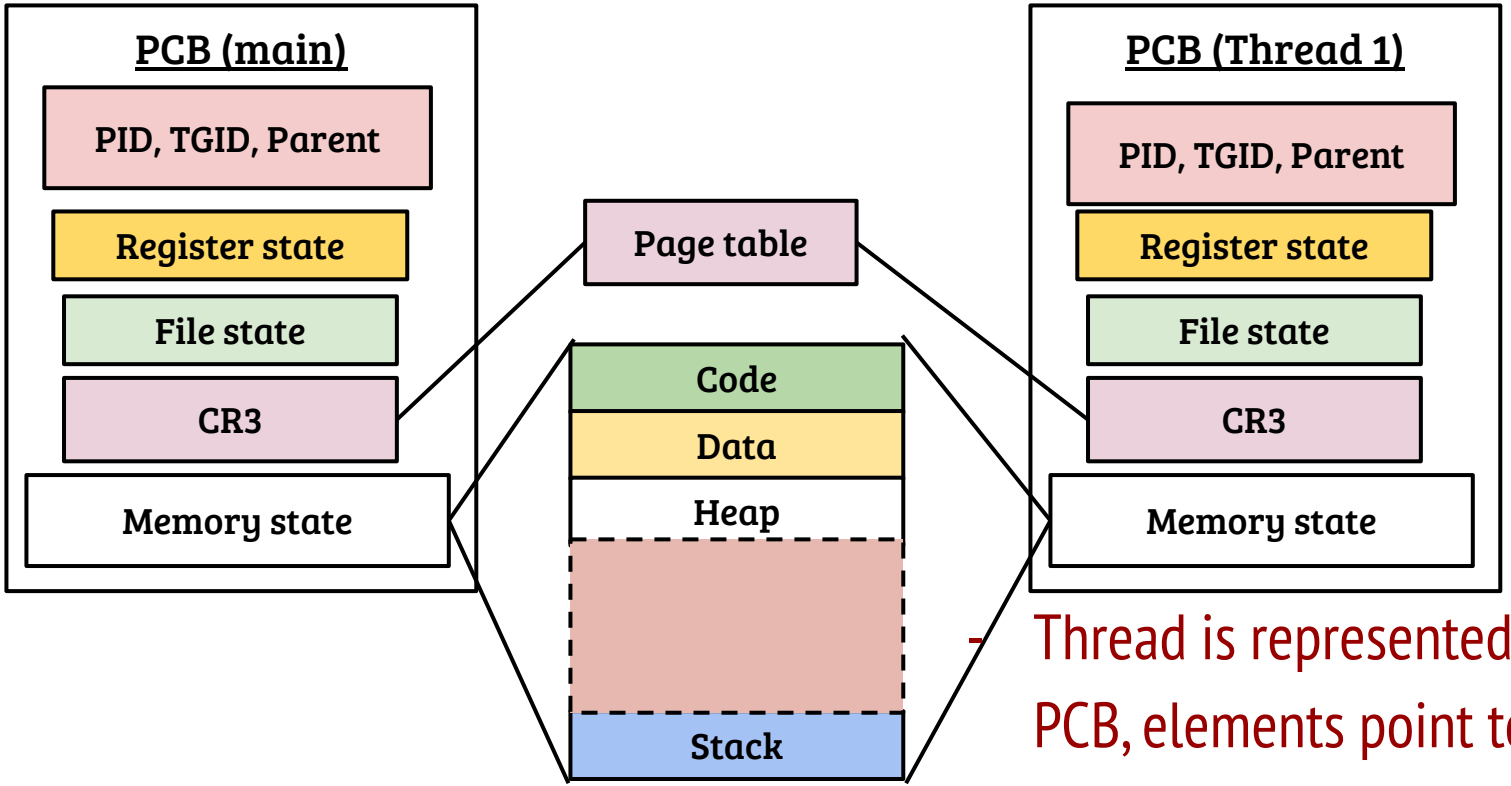
- PC is next instruction after fork( ) syscall, for both parent and child
- Child memory is an exact copy of parent
- Parent and child diverge from this point

User threads using posix thread API

```
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

PCB of a multithreaded process (Linux)



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

The clone system call

int clone(int (*fn)(void *), void *child_stack, int flags, void *arg, ...)

- Parent can control the execution of new process (execution and stack)
- Provides flexibility to the parent to share parts of its execution context in a selective manner
- Examples flags:
  - CLONE_FILES: Share files between parent and new process
  - CLONE_VM: Share the address space
  - CLONE_VFORK: Execution of parent process is suspended

Clone: Implementation in Linux kernel

- Syscall handler for clone should provide flexible sharing. Implementation?

Clone: Implementation in Linux kernel

- Syscall handler for clone should provide flexible sharing. Implementation?
  - Syscall Handler → Kernel clone → Copy process
  - Depending on flags, different subsystems are copied or shared
- Depending on the usage, the saved user state is required to be changed. Why? How implemented?

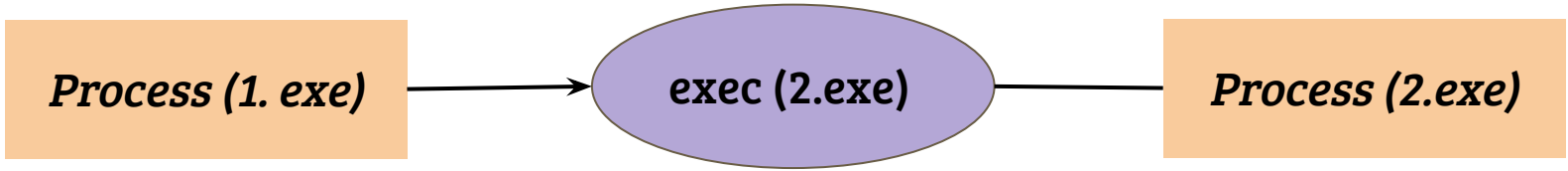
Clone: Implementation in Linux kernel

- Syscall handler for clone should provide flexible sharing. Implementation?
  - Syscall Handler → Kernel clone → Copy process
  - Depending on flags, different subsystems are copied or shared
- Depending on the usage, the saved user state is required to be changed. Why? How implemented?
  - For pthreads, the SP and RIP need to be changed
  - Change the register states during CPU thread copy
- Changes to the kernel space of newly created execution context required. Why? How implemented?

Clone: Implementation in Linux kernel

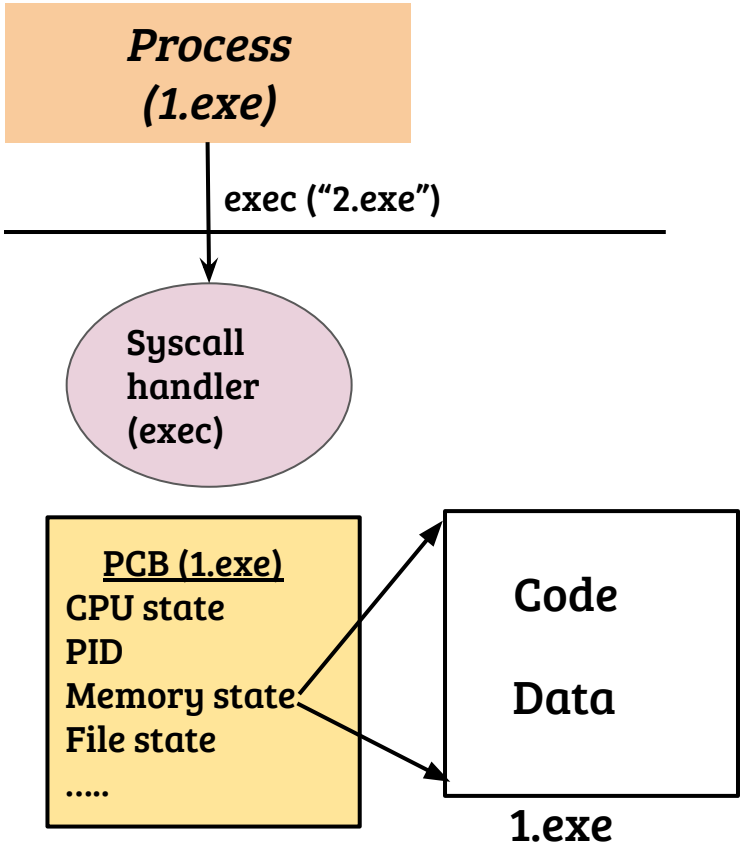
- Syscall handler for clone should provide flexible sharing. Implementation?
  - Syscall Handler → Kernel clone → Copy process
  - Depending on flags, different subsystems are copied or shared
- Depending on the usage, the saved user state is required to be changed. Why? How implemented?
  - For pthreads, the SP and RIP need to be changed
  - Change the register states during CPU thread copy
- Changes to the kernel space of newly created execution context required. Why? How implemented?
  - Child can not return in the same path, returns through a special stub

Load a new binary - exec( )



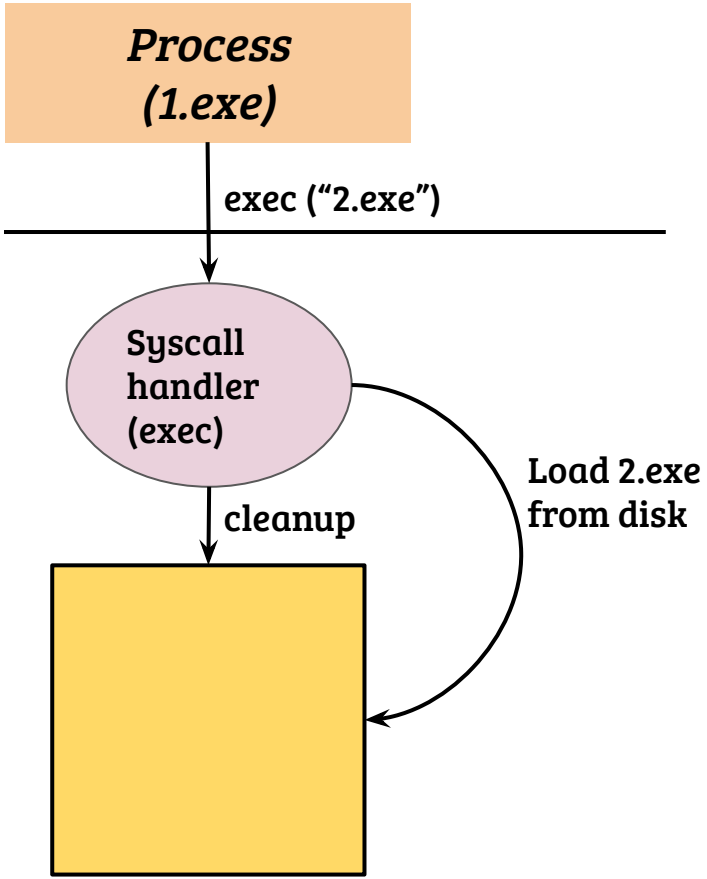
- Replace the calling process by a new executable
  - Code, data etc. are replaced by the new process
  - Usually, open files remain open

Typical implementation of exec



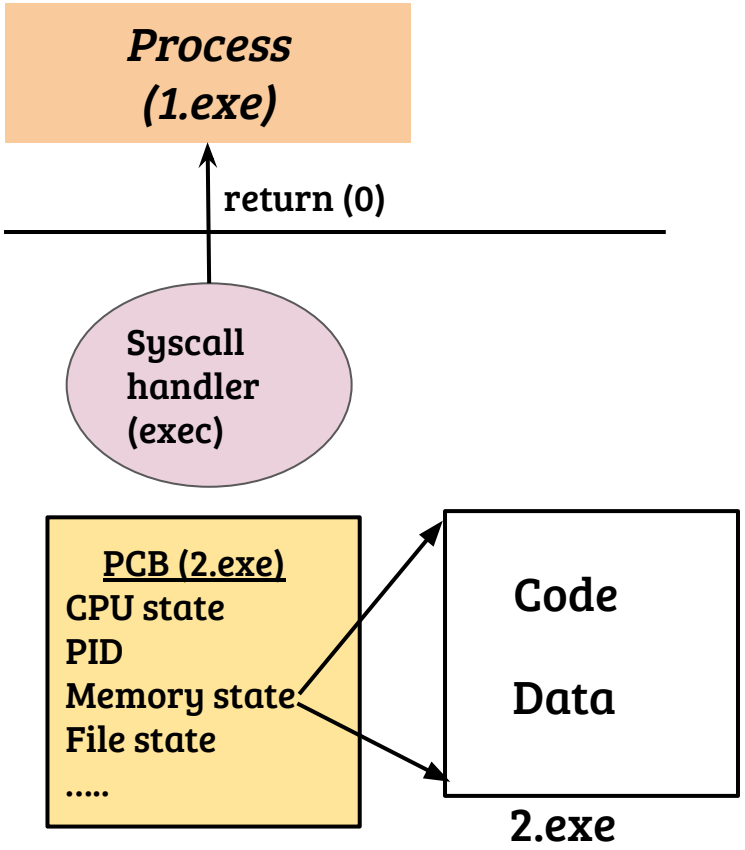
- The calling process commits self destruction! (almost)

Typical implementation of exec



- The calling process commits self destruction! (almost)
- The calling process is cleaned up and replaced by the new executable
- PID remains the same

Typical implementation of exec



- The calling process commits self destruction! (almost)
- The calling process is cleaned up and replaced by the new executable
- PID remains the same
- On return, new executable starts execution
- PC is loaded with the starting address of the newly loaded binary

Exec: Implementation in Linux kernel

- When should the self destruction of address space take place? What are the design choices?

Exec: Implementation in Linux kernel

- When should the self destruction of address space take place? What are the design choices?
  - Can not destroy until validity is checked; validity check not complete until the binary/arguments are examined
  - Duplicated processing vs. working with a fresh (discardable) space
  - There would be a point of no return, delayed is better
- How does the kernel parse the binary (and deduce entry address)? What about command line arguments?

Exec: Implementation in Linux kernel

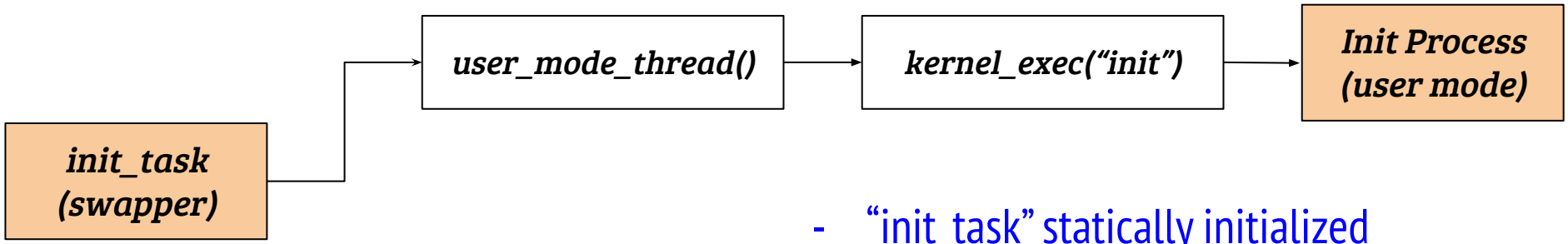
- When should the self destruction of address space take place? What are the design choices?
  - Can not destroy until validity is checked; validity check not complete until the binary/arguments are examined
  - Duplicated processing vs. working with a fresh (discardable) space
  - There would be a point of no return, delayed is better
- How does the kernel parse the binary (and deduce entry address)? What about command line arguments?
  - Basic binary parsing for ELF (and other types) e.g., load_elf_binary ( )
  - Command line arguments are placed in the stack

The first process

- What is the first execution entity in Linux?

The first process

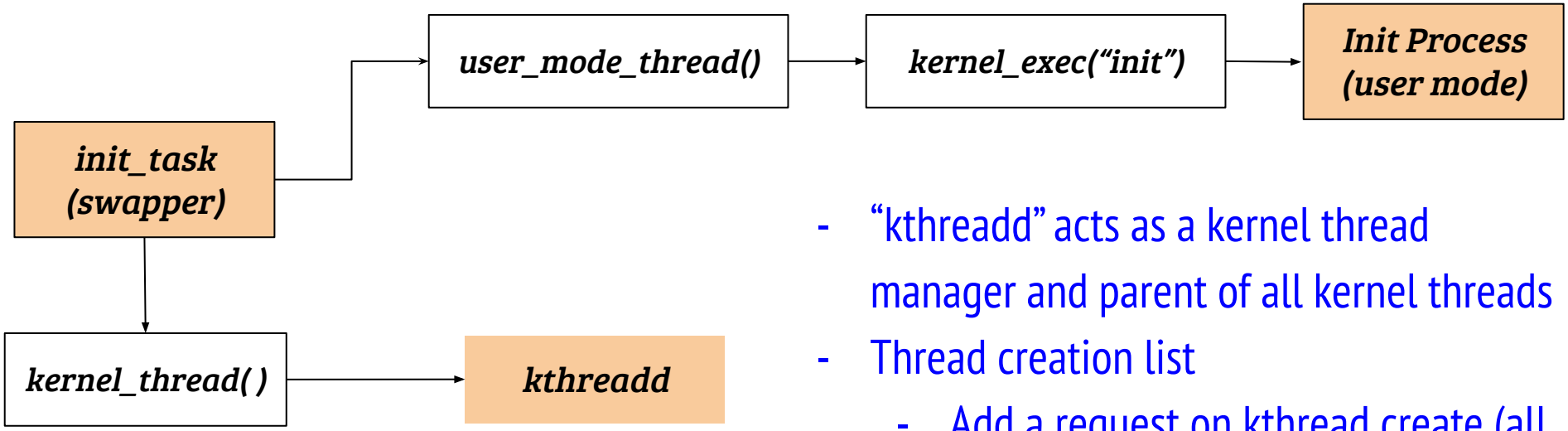
- What is the first execution entity in Linux?



- "init_task" statically initialized
- A special "clone" call from the kernel mode to create a thread of execution in kernel till actual init is executed
- Executes user space init based on configuration and default paths

The first process

- What is the first execution entity in Linux?



- "kthreadd" acts as a kernel thread manager and parent of all kernel threads
- Thread creation list
  - Add a request on kthread create (all types of kernel threads)
  - Wakeup kthreadd
  - Kthreadd ⟶ kernel_thread( )