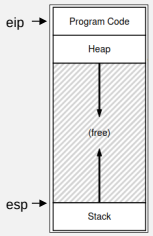
xv6 Processes and Threads

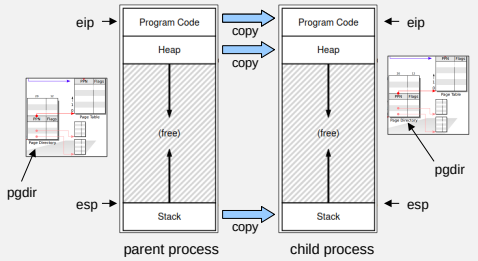
- processes and threads are the same from the perspective of the scheduler
  - both are included in the process table
- key registers
  - `eip` points to the next instruction to execute
  - `esp` points to the top of the stack
- when a trap occurs, registers are stored in the trap frame
  - part of the process table entry (`tf` field of `struct proc`)



xv6 Processes

`fork()` creates a new process with its own address space (which is a copy of the parent's)

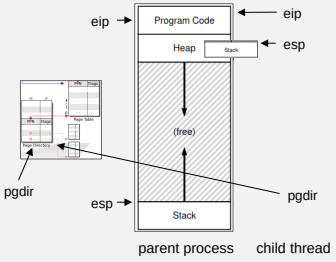
– parent's pages are copied
– child gets a new page directory where the same virtual addresses map to the copied pages
– `eip`, `esp` registers are copied – virtual address is the same even though the underlying physical pages are in different places
– parent's address space is inaccessible to the child



xv6 Threads

`clone()` creates a new thread which shares the address space of the original process

– only the pointer to the page directory is copied to the child (page directory is shared)
– child's stack is allocated on the heap
– `eip` is set to the entry point for the child thread (the subroutine the child thread should start running, and different from the parent's `eip`)
– `esp` is set to the top of the child's stack
– parent's address space (including the parent's stack) is accessible to the child, but the child does not access the parent's stack
– the child's stack is accessible to the parent, but the parent does not access it



Setting Up the Stack

the new thread starts running as if a function has been called so the contents of the stack must be set up as expected by the C calling convention

– `clone()` expects (a pointer to) a function taking two parameters, along with pointers to the values for those parameters and a pointer to the stack
– the thread never returns from this function call (`exit` must be called instead) so a fake return address of `0xffffffff` is used – this is why xv6 requires `exit` rather than return for `main`