Why does a Page Fault take so long?

Page Fault Handling

We are finally in a position to describe what happens on a page fault in some detail. The sequence of events is as follows:

1. The hardware traps to the kernel, saving the program counter on the stack. On most machines, some information about the state of the current instruction is saved in special CPU registers.
2. An assembly code routine is started to save the general registers and other volatile information, to keep the operating system from destroying it. This routine calls the operating system as a procedure.
3. The operating system discovers that a page fault has occurred, and tries to discover which virtual page is needed. Often one of the hardware registers contains this information. If not, the operating system must retrieve the program counter, fetch the instruction, and parse it in software to figure out what it was doing when the fault hit.
4. Once the virtual address that caused the fault is known, the operating system checks to see if this address is valid and the protection consistent with the access. If not, the process is sent a signal or killed. If the address is valid and no protection fault has occurred, the system attempts to acquire a page frame from the list of free frames. If no frames are free, the page replace algorithm is run to select a victim.
5. If the page frame selected is dirty, the page is scheduled for transfer to the disk, and a context switch takes place, suspending the faulting process and letting another one run until the disk transfer has completed. In any event, the frame is marked as busy to prevent it from being used for another purpose.
6. As soon as the page frame is clean (either immediately or after it is written to disk), the operating system looks up the disk address where the needed page is, and schedules a disk operation to bring it in. While the page is being loaded, the faulting process is still suspended and another user process is run, if one is available.
7. When the disk interrupt indicates that the page has arrived, the page tables are updated to reflect its position, and the frame is marked as being in normal state.
8. The faulting instruction is backed up to the state it had when it began and the program counter is reset to point to that instruction.
9. The faulting process is scheduled, and the operating system returns to the assembly language routine that called it.
10. This routine restores the registers and other volatile information, and returns to user space to continue execution, as if no fault had occurred.

taken from Andrew Tanenbaum's Modern Operating Systems
Prentice Hall, 1992. p. 127-128.