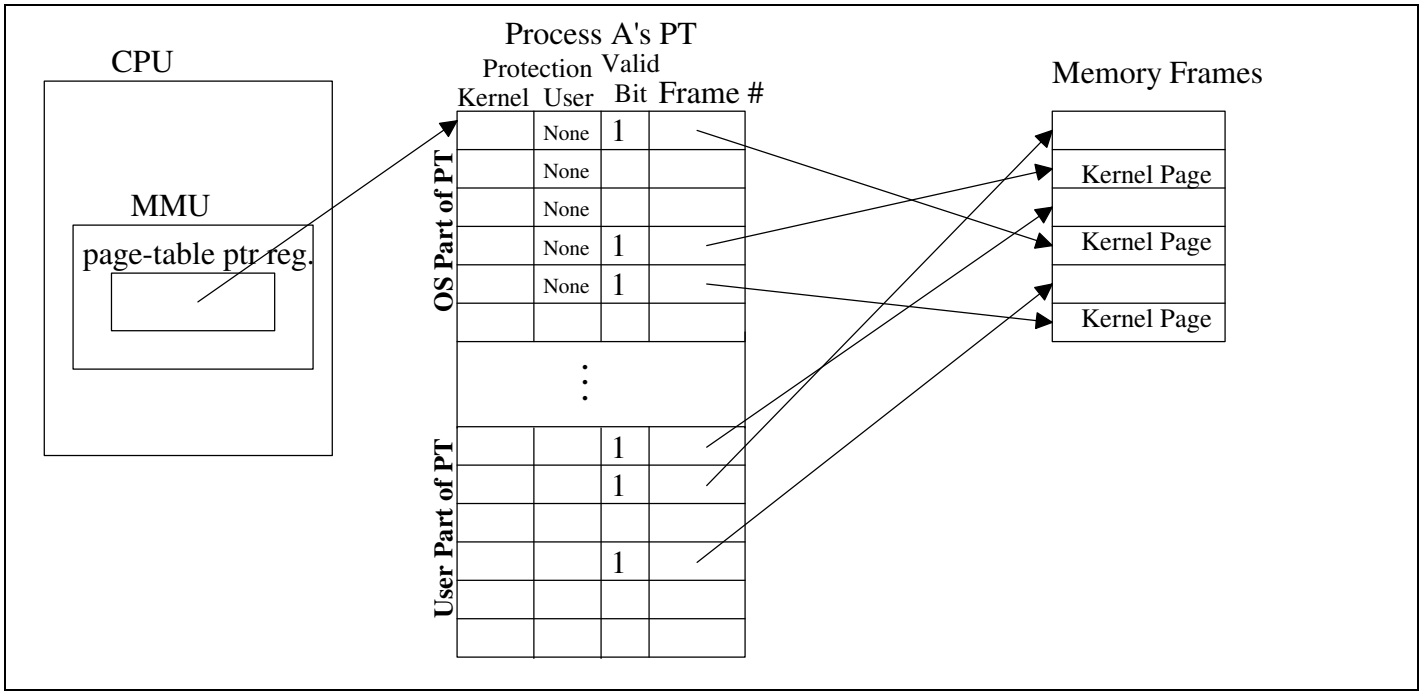
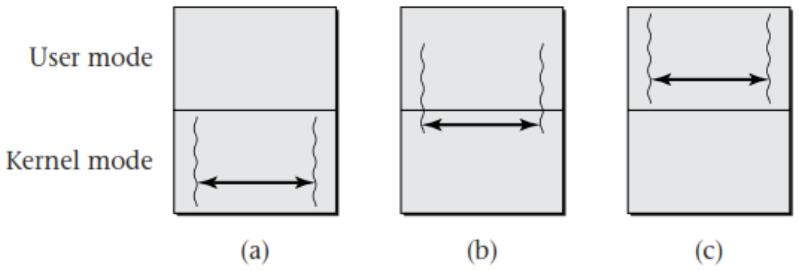
1. Linux on the IA-32 architecture has the OS kernel use the same page table as user-mode process. The user-mode process can access up to 3 GB of the virtual address space. virtual address space is split



If the scheduler runs as part of Process A, but decides to schedule a thread of process B. What will the MMU's page-table pointer register need to point at?

2. The following three relationships are possible between threads, the scheduling and dispatching code that switches threads, and the operating modes. Which of the following would involve operating mode switching?



a) The OS scheduler dispatches a kernel thread to perform an OS task.

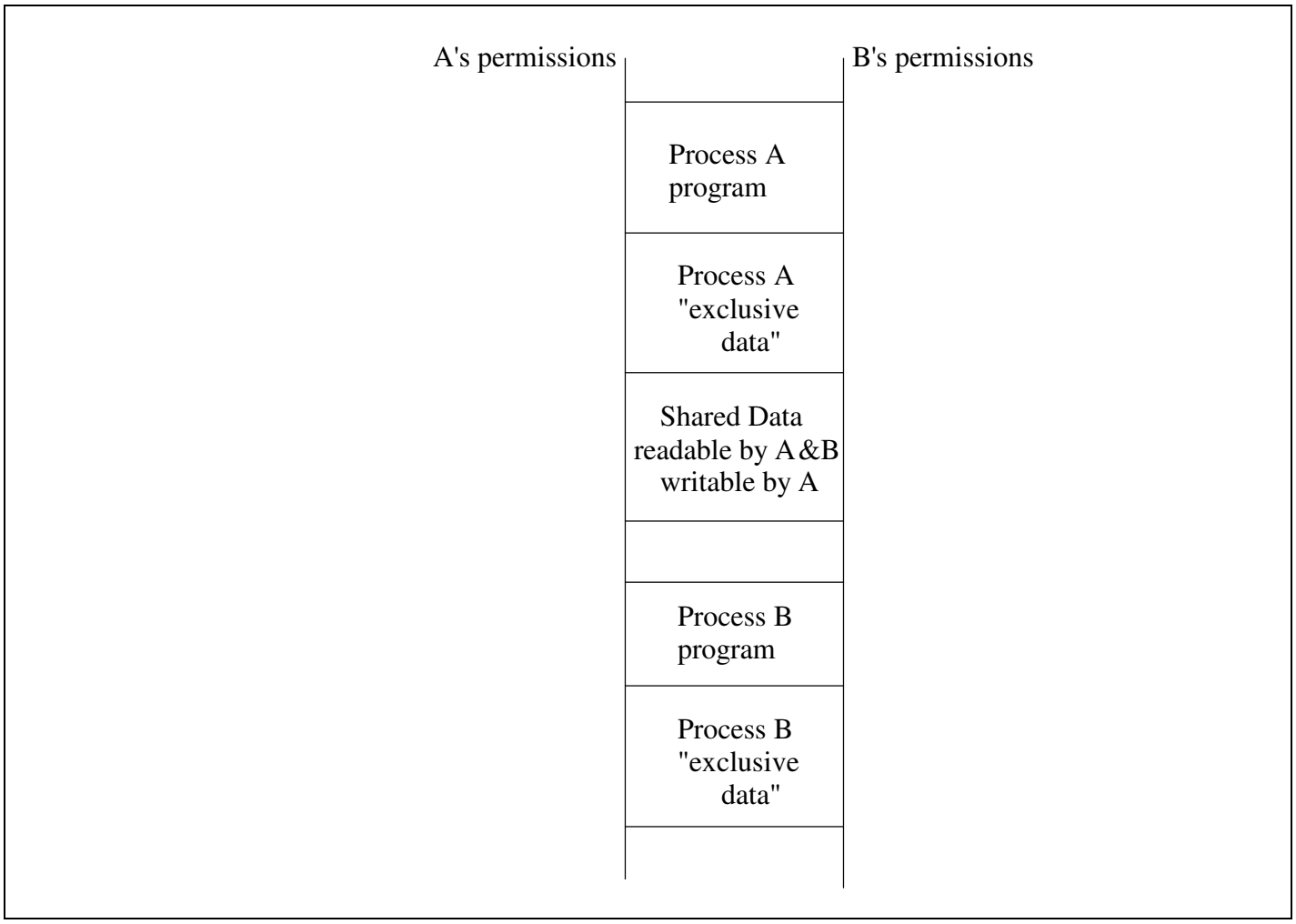
b) The OS scheduler dispatches switches between two threads that are part of a user-mode process.

c) The middleware system provides a scheduling and dispatch mechanism that runs in user mode and schedules between two user-level threads within the same process (e.g., *Microsoft fibers*)

3. What advantage does middleware systems running user-level threads (e.g., *Microsoft fibers*) have over (b)?

4. "Mainstream" operating systems/architectures protect one process's physical memory from another processes by having each process run in its own virtual address space. Each process has its own page table containing references to a disjoin set of main memory frames. Thus, processes are protected from one another through *addressability*, a process has no ability of to generate a physical address used by another process.

Alternatively (rarely done), all processes share a single virtual address space and page table, so they are **not** protected from each other by addressability. Processes need to be protected from one another via *accessibility*, processes need to differ with regard to their read, write, execute permission to regions of virtual memory. What read, write, execute permissions should process A and B have in the following memory regions?



5. Intel Itanium architecture has the following characteristics:

- single address space shared by all processes
- each page-table entry contains protection key (a #)
- all pages protected in the same way have the same key
- each process prossess a set of key
- for every memory access by a process, the process must hold the key for that page of memory

a) What if a process access a memory reference without holding its key?

b) How can this scheme be made efficient?