103011227 周延儒 CS18

1. [20 points] 9.2 A simplified view of thread states is Ready, Running, and Blocked, where a thread is either ready and waiting to be scheduled, is running on the processor, or is blocked (for example, waiting for I/O). This is illustrated in Figure 9.30. Assuming a thread is in the Running state, answer the following questions, and explain your answer:

    1. Will the thread change state if it incurs a page fault? If so, to what state will it change?
       Yes, a tread changes from the **Running state to the Blocked** state.

       When a page fault occurs, the process begin to wait for I/O operation to finish. The OS does several things while the process is waiting. To be more specific, It checks whether the page is really invalid or just on disk, finds a free memory frame, schedules a disk access to load the page into the frame, updates the page table with the new logical-physical mapping, updates the valid bit of that entry, and eventually restarts the process by change its state **from Blocked to Ready.**

    2. Will the thread change state if it generates a TLB miss that is resolved in the page table? If so, to what state will it change?

       Not necessarily. If a page table entry is not found in the TLB (TLB miss), the page number will used to index and process the page table.
       If the page is already in main memory, TLB just need to be updated to include the new page entry, while the process execution **continues** since there is no I/O operation needed. And, the thread doesn't need to change the state

       If the page is not in the **main memory**, a page fault is generated.
       In this case, the process needs t**o change to the Blocked state** and wait for I/O to access the disk. It is as same as the q1

    3. Will the thread change state if an address reference is resolved in the page table? If so, to what state will it change?

       No, this is because **no I/O operation** is required when the address reference is resolved in the page table, which indicates the page needed is already loaded in the main memory already.

2. [10 points] **9.4** What is the copy-on-write feature, and under what circumstances is its use beneficial? What hardware support is required to implement this feature?

Well, Copy on Write allows processes to **share pages** rather than each having a separate copy of the pages. However, when one process tried to **write to a shared page,** then a trap is generated and the OS makes a separate copy of the page for each process. Consider fork() as a common example. The fork() operation is where the child is supposed to have a complete copy of the parent address space. Rather than creating a separate copy, the OS allows the parent and child to share the parent's pages. But, since each is supposed to have its own private copy of the pages, the pages are copied when one of them attempt a write.

Moreover, the hardware support required to implement are the followings:
On each memory access, the page table needs to be consulted to
check whether the page is write-protected. If it is indeed write-protected,
a trap would occur and the operating system could resolve the issue.

3. [10 points] **9.19** What is the cause of thrashing? How does the system detect thrashing? Once it detects thrashing, what can the system do to eliminate this problem?

Thrashing is caused by under **allocation of the minimum number of pages required by a process, forcing it to continuously page fault.**

Moreover, The system can **detect** thrashing **by evaluating the level of CPU utilization** as compared to the level of multiprogramming.
And, the system can be eliminate the thrashing by **reducing the level of multiprogramming.**