1. In a system using modern virtual memory techniques, what is the relationship between a page and a page frame?

Page frame is a piece of physical memory, usually 1-4k (the size of a page), it is the place where pages will be put in, allocated by the OS. And page table consists of all page frames, with each of its page table entry corresponding to a page. a page in virtual memory (virtual page) is mapped to a page table entry in physical memory, thus a page frame in physical memory (physical page).

2. Why are operating system ABIs of importance for convenient application software distribution?

It is specific for a machine. Mainly an interface designed for users, including applications, so that the users (applications) can use the system's resource, such as library. OS kernel/services, memory without too much further configuration. Convenice in the sense that the application usually can directly run on different machines because of it

3. Why is information hiding a good property in an operating system interface?

By information hiding, users in user-mode won't be able to access or edit critical information in the OS kernel, which is important to the whole system. A single bug can cause the entire system to crash.

Also in this way, a process cannot see the information (address space) of other processes, which ensures the security of the system.

Also it suppores abstractions better since no need to care those petty details about thus increase the robustness of the OS.

4. When an operating system performs a context switch between processes, what information must the OS save?

It must save the previous state, register information, Library information, stack information, and many other things of previous process. it will also switch to the new process's address space and use its stack, and page table instead.

In this way, when we come back to run this previous process. We know the current state, and place it's at PC, registers, stacks. to continue running.

Also the working set information of previous process. So that when switched back, pre-load that into TLB, to avoid huge amount of page at the beginning

5. What is the purpose of a trap table?

when a trap is "fired", the OS knows where to go to handle that particular exception (e.g. TLB-miss will go for TLB-miss handler)

It's usually loaded at boot time, and will be there until next boot. It stores the location to go (for the OS) when a trap occurred, so that the OS can perform operations for this trap.

specific process will be: get information from vector table, then 1st-level trap handler - second level trap handler for correct place to deal with the trap.

6. What is a race condition?

When two (or more) threads, or two or more processes (on multi-core system) try to access and update some same shared area of code, If we don't use synchronization technique (e.g. mutual exclusion), they might do the update concurrently, which causes trouble, including race conditions:

For example: int store = 0;
store = store + 1;

If "store" is a global variable (shared data), and say we want two threads to add 1 to it so that it will be added 2. But race condition happens when both threads do the update (concurrently) when "store" is 0, thus resulting in 1 instead of 2.

Update to a single piece of shared data will result in unexpected result if there's no protection.

7. Why is blocking a problem for user-mode threads? Why isn't it a problem for kernel-mode threads?

In user-mode, if a thread is blocked, as a user, we don't have right to know if other threads can still run. so all threads will be also blocked, waiting for it. waste of time, resources, and slow down the sy

However, in kernel-mode, the OS has information of the states of all threads. Thus if a thread is block and if another thread is still good to run, the OS will let it run.

8. Why does Shortest Time-To-Completion First (STCF) scheduling require preemption?

If not all processes arrive at the same time, a later-come process might have shorter "Time to Completion", so we need to preempt the current process and schedule the new one instead.

Also, For this scheduling, the processes with long time to complete may not be able to run at all. not fair and we might lose important information because of not running them. So we need preemption to run those process as well.

9. When a Unix-system follows a fork with an exec, what resources of the forked process are replaced?

An exec will create a brand new child process, a blank one

After the fork, the forked (child) process will copy all information of parent process except that it has its own address space, sets of registers and stacks, etc. and share all the other thing with the parent process

After the exec, the shared resources/data will be replaced. It will have its own data segment, code segment, libraries, etc.

[P.S Here "shared" does not mean sharing resources with the parent process. It means those things are the same between child and parent, "copied"]

10. What form of fragmentation do we still suffer if we use a paging memory management system? For a segmented paging system, how much fragmentation per segment do we see?

Internal Fragmentation.

[For a segmented paging system, we only acquire a new page if current segment is fully used, so internal fragmentation will be only on the last page of the segment, ranging from (1) to (page_size - 1) bytes]

- For "Fixed-sized segments", we have both internal and external fragmentation
- Internal fragmentation since fixed-sized segment allocated for a process might not be fully used.
  External fragmentation: since segments might not be contignous, there will be small memory chucks left over spread all over.
- At most ((segment_size) - 1 byte) fragmentation
- size of memory unused / total size of memory