CS302: Assignment3 Report

Name：陆荻芸 SID：12011537

1 Answers of Question 1

1.1.

Three easy pieces are: virtualization, concurrency and persistence.

- **Virtualization**: It is the action that OS generates a virtual form which is easier, more general and powerful to use using physical resources. Virtualizing a single CPU allows many programs to seemingly run at once using seemingly many CPUs. It is also about virtualizing physical memory that OS somehow maps virtual address space of each process to.
- **Concurrency**: To enable OS to handle multiple objects at once, concurrency is here to aid. This is achieved by dividing the available resources of the system among the different processes.
- **Persistence**: Persistence is required in data storage, we don't want to lose data when system crashed or shutdown. For OS, a file system is used to manage the disk.

1.2.

Virtualization is mentioned in the introduction chapter 1.7 and chapter 18 introducing virtual machines.

Concurrency is in chapter 4, mentioning threads and concurrency.

Persistence is in chapter 13 and chapter 14, where discuss about file system.

Q1(2) Virtualization: Chapter 3, 5, 9, 10 of "dinosaur book" Concurrency: Chapter 4, 6, 7, 8 Persistence: Chapter 11-15

2 Answer of Question 2

Context switch begins when the scheduler decided to switch to another process when the OS regained the control from CPU. First of all, OS will execute some low-level assembly codes to **save** some values of general purpose registers, PC, and the kernel stack pointer of the currently-running process. Then to **restore** the information of the soon-to-be-executing process, and switch to the kernel stack for it. By switching stacks, kernel calls the code to switch to another process. When the OS finally returns, the soon-to-be-executing process will **resume** its execution from where the previous process was interrupted and become the current process. Then the context switch is finished.

3 Answers of Question 3

3.1.

When the kernel handles the `fork()` system call:

- First when the user program invokes the system call, interrupt will be caused and transfer the OS into kernel mode.
- Then for PCB, most of things will be cloned including CPU registers which enables the child process to execute from the line after the `fork()` returns, and files, memory and list of opened files also. However, the PID will be changed. Information including the parent process, the process state of parent process and the running time will change too.
- Action on address space depends on specific cases. If it is the case of copy on write, then the address space will not be copied unless updates have been made. If it is not, then the address space will be duplicated.
- CPU scheduler will schedule the process according to the using policy. Both parent and child process have the chance to be executed first after `fork()` returns
- If the CPU scheduler decides to let the child process to run which is different from the previous one, context switch will happen.
- For the parent process, the return value of the `fork()` system call is the PID of the child process. For the child process, the return value will be 0 if the process has bee successfully created.

3.2. Q3(2)应该提及PID 是否在 Kernel's Process Table中

When the kernel handles the `exit()` system call:

First the kernel frees all the allocated kernel-space memory and closes all the opened files. Then kernel frees everything on the user-space memory. After, the kernel notifies its parent process with SIGCHLD that this process has been terminated and needed to be cleaned up.

- If the parent process didn't call the `wait()` system call, then the parent process will ignore the SIGCHLD. Then the child process now will become a "zombie".
- If the parent process called `wait()` system call, a signal handling routine will be registered to handle the SIGCHLD. If the `wait()` system call is called before the child process terminated, then the parent process will wait until it receives the SIGCHLD then return after handling the terminated child process. If it is called after the child's termination, then the SIGCHLD will be pended till the call to `wait()` happened and handle the termination. So between the `exit()` and `wait()` calls, the child process is in an zombie state waiting to be totally removed.

4 Answer of Question 4

Three methods: system calls, interrupt, trap or exception. When these methods trigger transitions, they will all result in context switch.

1. **System Calls**: They are programming interfaces provided by the OS. Calling system calls can trigger interrupts to convert the OS into kernel mode then let the kernel mode to finish the jobs. Caller only needs to obey the calling convention but nothing about the kernel mode. And we can pass parameters through system calls.

2. **Interrupt**: They are external asynchronous events which trigger context switch. When CPU receives notification from device or from another CPU, interrupts will take place and preempt current execution. Then processor will find the entry of the handler of current interrupt in IDT. By executing the handler program, the interrupts are handled and OS will then return.

3. **Trap or Exception**: They are internal synchronous reactions to an abnormal conditions that are related to programs like the error of divided by zero. They are handled in the same way as OS handles interrupts.

5 Answer of Question 5

Life cycle of a process: There are 5 status in the life cycle of a process and they transit from one to another under different situations.

New: When a process is newly created.

Ready: Waiting to be assigned to CPU resource.

1. When a process is **New** and it is initialized, it will then enter the ready state.

2. When a process is **Running** and interrupt takes place, then it will enter the ready state.

3. When a process is **Waiting** and the waiting event has been done, then it will enter the ready state.

Running: Instructions are executing on the CPU.

1. When a process is **Ready** and the CPU scheduler has assigned CPU resource to it according some policy, then it will enter the running stage.

Waiting: Waiting for some operations in the kernel mode to be done such as I/O events.

1. When a **Running** process need to handle some events, it will turn into **Waiting** and wait for the completion of the events.

Terminated: Execution has been completed and the process is either terminated by the OS or by its parent process.

1. When a **Running** process finishes, it will exit and being terminated.