OS Assignment 3 Report

name: 刘乐奇

sid: 12011327

1. (1) The “three easy pieces” are: Virtualization, Concurrency, Persistence.

- Virtualization: operating system will provide plenty of virtual CPU, which are actually derived from a single CPU. Besides, operating system will map the virtual memory onto physical memory, making every process have its own private virtual address space.
- Concurrency: operating system has to manage many things so that makes numerous process running at the same time.
- Persistence: operating system needs some hardwares and softwares to persistently store the data.
  (2) In the “dinosaur book”, virtualization maps to part 4, concurrency maps from part 2 to part 3, persistence maps from part 4 to part 6.

2. During context switch, operating system will eexecute some assembly code to save some state of registers, such as general registers, program counter, and the kernel stack pointer of the running process. Then the registers and program counter are reset and switch kernel stack.
3. (1) When the user process invoke `fork()`,

- the user process is trapped so that the mode bit is set to be 0, i.e., it gets into kernel mode;
- then CPU will look up the trap table and create a new process;
- During the creation of the new process, the PCB will be created, which inherits some information from parent process such as file descriptor, context, program counter, and so on; but some are different, such as PID.
- The address space of the new process may be totally duplicated (the same as the address space of parent process but not in the same place) or the same but cpoy-on-write (the address space of the new process still points to the parent process's, but will be copied when modified).
- The PCB of the new process and parent process will be inserted into the scheduling queue, waiting for scheduled by CPU scheduler.
- If the next process scheduled is the new process, there will be a context switch, including loading the PCB of the new process into kernel and giving the control of CPU to the new process; else will not.
- The return value of `fork()` depends on which process is next scheduled: if new process, it will be 0; else it will be the PID of the new process.

(2) When the user process invoke `exit()`,

- The kernel will free all the allocated memory of this process, and close all the opened file in this process.
- Then, the kernel will free everything in the user space memory of this process.
- The status of this process will turn to ZOMBIE.
- The kernel will notify the parent process of this process by sending a SIGCHILD signal.
- If the parent process is in `wait()`, signal handler in the parent process will accept SIGCHILD signal and destroy it, and wipe out the ZOMBIE child process in kernel space. Then the signal handler will be deregistered and return the PID of the ZOMBIE child process as the return value of `wait()`.
- If the parent process is running, SIGCHILD signal will queue in parent process. As soon as the parent process gets into `wait()`, it will take action immediately to SIGCHILD signal.

4. The user process makes a system call, raises an exception, and an interrupt occurs.

- System call: when the user process makes a system call, it will be trapped into kernel mode. Then operating system will look up the trap table to find which system call to be invoked, i.e., other behaviours will not be taken.
- Exception: it is synchronous and reacts to an abnormal condition.
- Interrupt: it is asynchronous and preempt normal execution. When interrupt occurs, CPU will find handler in interrupt descripter table with interrupt number.

5. There are 5 states of a process: new, ready, running, waiting, terminated.
   When a program is going to execute, operating system or another existing process will create a new process (`new`). The new process then will be admitted to waiting queue, ready for CPU scheduling (`ready`). When the process is scheduled, it will be dispatched to CPU and start running (`running`). If the running process is descheduled, it will transit to `ready` state; if the running process is blocked, such as I/O invocation, it will transit to `waiting` state, and after the I/O invocation, it will transit to `ready` state, waiting for being scheduled. While running, if the process halts, it transit to `teminated` state.