Assignment 2

Please complete a report.

The documents to be submitted for this assignment are:

- report.pdf

---

1. **[20 pts]** Read Chapter 2 of “Three Easy Pieces” (https://pages.cs.wisc.edu/~remzi/OSTEP/intro.pdf). Answer the following questions:

   (1) What are the “three easy pieces” of operating systems? Explain each of them with your own words.

   Virtualization, concurrency, persistence.

   - **Virtualizarion:** The operating system virtualizes physical resources (such as CPU, memory, and hard disk) and provides different APIs for direct use of these physical resources. This allows different programs to have their own suitable running resources. From this perspective, the OS can be seen as a resource manager, providing the necessary running environment for programs.
   - **Concurrency:** Handling different processes and make concurrency. Actually, only one instruction/program is running at a time. Concurrency involves how to correctly schedule these threads, handle their memory access and usage correctly, so that they can appear to be running "simultaneously" and produce the expected results.
   - **Persistence:** For data that is easily lost when power off (DRAM, SRAM), people should found effective ways to persistently store it. This part involves various directions such as I/O programs, hardware devices and drivers, RAID, file systems, etc。

   (2) How do these “three easy pieces” map to the chapters in the “dinosaur book”?

   - Virtualizarion: Chapter 3, 5, 8, 9
   - Concurrency: Chapter 4, 6, 7
   - Persistence: Chapter 10-13

2. **[20 pts]** Read Chapter 6 of “Three Easy Pieces” (https://pages.cs.wisc.edu/~remzi/OSTEP/cpu-mechanisms.pdf) and explain what happens during **context switch** in detail?

   For example, when a context switch is about to occur, the register data of process A is saved to the kernel stack. At this point, the OS enters kernel mode and decides to switch from process A to process B. The register data of process A is saved to its structure entry, and the required register data for process B is restored from the structure entry of process B. Then, the context switch occurs, stack pointer to use B’s kernel stack. Finally, the context switch is complete, and process B is running.

3. **[20 pts]** Read slides “L03 Processes” and answer the following questions:

(1) Explain what happens when the kernel handles the fork() system call (hint: your answer should include the system call mechanism, PCB, address space, CPU scheduler, context switch, return values of the system call).

- **System call mechanism:** It creates the child process by cloning from the parent process, including all user-space data, e.g. program counter, program code, memory, opened files。
- **PCB:** Child process will clone necessary information from parent process, e.g. state, program counter, register data. But some data will be updated, e.g. such as PID.
- **Address space:** The child process will have a new address space, they are independent (it an be either duplicate address space or copy on write actually).
- **CPU scheduler & context switch:** Kernel decide the order of program execution and perform context switch if necessary.
- **Return values:** In the parent process, the `fork()` returns the process ID of the child process, while in the child process, the system call returns 0.

(2) Explain what happens when the kernel handles the exit() system call (hint: your answer should include discussion on the zombie state and how it is related to the wait() system call).

In `exit()`, the kernel frees all the allocated memory, and the list of opened files are all closed.

Then, the kernel frees everything on the user-space memory about the concerned process.

But in this case, child process ID remains in the kernel's process table, the status of the child is now called zombie. This means the program execution is complete but the kernel still reserve data for it.

The kernel sends a `SIGCHLD` signal to the parent to notify the termination of its child.

When the parent process execute the `wait()`, it will suspend the parent process to wait the child process `SIGCHLD` signal (when child call `exit()`). When one of its child processes changes from running to terminated, it will return and recycle the child process resource. Thus the child process will be not zombie but "dead".

If the parent process doesn't call the `wait()` and the child process call the `exit()`, child process will become zombie process.

4. **[20 pts]** What are the three methods of transferring the control of the CPU from a user process to OS kernel? Compare them in detail.

   System call, interrupt, trap or exception.

   - System call: Processes requests a system service, and the OS give the response. Like a function call, but "outside" the process. It can considered as APIs that provided by OS.
   - Interrupt: Preempt normal execution, usually some notification from device or preemptive scheduling.
   - Trap or exception: React to an abnormal condition, usually some illegal instructions.

   Compare to others, system call is more limited and complete.

   For exception and interrupt, they both involve stop the current program and start execution of a handler, and determine execution code and priority from IDT. But exceptions are synchronous, interrupts are asynchronous. And exceptions come from "inside" code or OS wrong, and interrupts come from "outside" device.

5. **[20 pts]** Describe the life cycle of a process (hint: explain the reasons for process state transitions).

There are some states of a process life cycle, I will mention their transitions in the description.

- New: When the process is created, it is in new state. It need to be initialize. If it is admitted by OS, it will become ready state.
- Ready: In this state, it get memory and is ready for execution, but it needs scheduler to get CPU resources to become running state.
- Running: When process get CPU resources and is executing instructions, it is in running state. In this state, if a interrupt occurs, the process will back to ready state. If it have I/O or event to wait, it will become waiting state.
- Waiting: In this state, process wait for some I/O or event. Once the I/O or event is complete, the process will back to ready state.
- Terminated: If a process finish its execution or the OS need to terminate the process, it will become terminate state.