Homework 03

CS307-Operating System (D), Chentao Wu, Spring 2020.

Name: 方泓杰(Hongjie Fang) Student ID: 518030910150 Email: galaxies@sjtu.edu.cn

- (3.1) Using the following program, explain what the output will be at LINE A.

```c
# include <sys/types.h>
# include <stdio.h>
# include <unistd.h>
int value = 5;
int main() {
    pid_t pid;
    pid = fork();
    if (pid == 0) { /* child process */
        value += 15;
        return 0;
    } else if (pid > 0) { /* parent process */
        wait(NULL);
        printf("PARENT: value = \%d", value); /* LINE A */
        return 0;
    }
}
```

Solution. The output of LINE A will be as follows.

```
PARENT: value = 5
```

The child process will copy the data from the parent process, so the modification of "value" in the child process have no influence on the "value" in the parent process. Therefore, the "value" in the parent process will remain to be 5. □

- (3.2) Including the initial parent process, how many processes are created by the following program?

```c
# include <stdio.h>
# include <unistd.h>
int main() {
    /* fork a child process */
    fork();
    /* fork another child process */
    fork();
    /* and fork another */
    fork();
    return 0;
}
```

Solution. Suppose the initial parent process is called P. In the first "fork()", it will create a child process, and let us call it A. The second "fork()" will be executed in both P and A, and create a child process for P and for A respectively, called B and C. The third "fork()" will be executed in P, A, B and C, and create a child process for each of them respectively, called D, E, F and G. So there are total 8 processes (P, A, B, C, D, E, F, G) created by the following program, including the initial parent process P. □

- (3.4) Some computer systems provide multiple register sets. Describe what happens when a context switch occurs if the new context is already loaded into one of the register sets. What happens if the new context is in memory rather than in a register set and all the register sets are in use.

  Solution. If the new context is already loaded into one of the register sets, then CPU simply changes its **current-register-set pointer** to the set which contains the new context, and this operation only takes a little time. If the context is in memory, then one of the contexts in a register set will be substituted with the new context in the memory, and the original one will be the moved to the memory. This process takes more time, since accessing memory is much slower than accessing registers. And the average time of this process also depends on the replacement strategies of the register sets. □

- (3.8) Describe the actions taken by a kernel to context-switch processes.

  Solution. Actions taken by a kernel to context-switch processes are as follows.

  – The operating system saves the registers and process state into the PCB (Process Control Block) of the current process to make sure that it can be executed normally in the future.

  – The operating system then invokes the process scheduler to find out which process will be executed next.

  – The operating system retrieves the state and restores the registers of the next process from its PCB. Then the next process will be executed from where it was previously interrupted.

  – Some special operations should be done in this process, such as flushing the instruction cache (i.e., clearing the instructions of the old process and putting some instructions of the new process in it).

  □

- (3.10) Explain the role of the **init** (or **systemd**) process on UNIX and Linux systems in regard to process termination.

  Solution. **init** is the ancestor of all other processes. When a process executes **exit()**, it will not disappear immediately. Instead, it will move to the zombie state in a short period of time. It needs its parent process to invoke a call to **wait()** to release the resources thoroughly. If its parent process will not invoke **wait()**, it will become a zombie process as long as its parent process is still running. After its parent process terminates without calling **wait()**, the **init** process will become the new parent of the zombie process. The **init** process will periodically invokes **wait()** in order to release the resources of the zombie processes. Similarly, if a process is still running but its parent process is terminated, which is called an orphan process, then the **init** process will become the new parent of the orphan process and help it terminate and release the resources. Therefore, the **init** (or **systemd**) process will clear the zombie processes and orphan processes to make the system cleaner and tidier. □