Multiprocessing: processes; fork

COSC 208, Introduction to Computer Systems, 2022-04-19

Outline

- Process abstraction
- Creating processes

Process abstraction

- Process — running program and its corresponding machine state (code, memory, and register values)
    - Program is static code and static data; process is dynamic instance of code and data
    - Cooking analogy
        - Recipe == program
        - Following a recipe == process
        - Ingredients == program inputs
        - Prepared food == program outputs
        - Cabinets, fridge, etc. == persistent storage
        - Prep area (e.g., counter) == registers & main memory
        - Contents and status of the prep area; current step of recipe == machine state
        - Chef == CPU
    - Can have multiple processes all running different instances of the same program
        - Cooking analogy — chef may be making multiple batches of the same recipe
- OS is responsible for...
    - Creating processes — when a user or another application requests it
    - Scheduling processes — i.e., deciding when/which process should be allowed to use the CPU
    - Switching processes — i.e., saving the machine state of one process and restoring the machine state of another process; called context switching
    - Cleaning-up processes — i.e., releasing any resources they are using when the process is done
    - Interacting with I/O devices (e.g., disks, NICs, graphics card) on behalf of processes

Warm-up

- Q1a: *Consider building a Lego kit as an analogy for operating systems' process abstraction. Match each component of the analogy with the corresponding component of a real computer system.*

    - Instruction booklet == program
    - Following the assembly instructions == process
    - Lego bricks == program inputs
    - Completed kit == program outputs
    - Cabinet/drawers for storing Legos == persistent storage
    - Building area (e.g., tabletop) == registers and main memory
    - Current step for the instruction booklet == program counter
    - You == CPU

- Q1b: *True or False*

1. Code stored on secondary storage (e.g., a solid state drive) is called a process -- false; it is a program
2. Each process has its own code, heap, stack, and register values -- true
3. The CPU is in user mode when executing application code, and kernel mode when executing OS code -- true
4. A process can directly execute instructions on the CPU -- true; user mode uses Limited Restricted Execution (OS isn't an intermediary until privileged instructions)
5. A process can directly access input and output ports -- false; indirect, need a system call

Creating processes

- `int fork()`
  - Creates an exact copy of the running process, except for the return value from `fork` —
    return `0` to child (i.e., new) process;
    return child's process ID to parent process (i.e., process that called fork)
  - Both child and parent resume execution from place where `fork` was called
- Q2: *What does the following code output?*

```c
int main(int argc, char **argv) {
    printf("Before fork\n");
    int pid = fork();
    printf("After fork\n");
    return 0;
}
```

```
Before fork
After fork
After fork
```

- Q3: *What does the following code output (assuming the new process has PID 1819)?*

```c
int main(int argc, char **argv) {
    printf("Before fork");
    int pid = fork();
    if (pid == 0) {
        printf("Child gets %d\n", pid);
    } else {
        printf("Parent gets %d\n", pid);
    }
    return 0;
}
```

```
Before fork
Child gets 0
Parent gets 1819
```

OR

```
Before fork
Parent gets 1819
Child gets 0
```