Orphan vs Zombie vs Daemon processes

Wed, Aug 15, 2012

Just finished reading Working With Unix Processes

see ruby examples I noted.

Pros

- I'd recommend it to anyone interested in Ruby and Unix as it's easy to understand and follow.

Cons

- `Kernel#select` wasn't not covered
- $27 is a bit too much as for 143 PDF document in my opinion.

Process kinds

Just to clarify terminology what is what(quoted from Wikipedia).

Process

> *In computing, a process is an instance of a computer program that is being executed. It contains the program code and its current activity. Depending on the operating system (OS), a process may be made up of multiple threads of execution that execute instructions concurrently.*

Parent Process

> *In the operating system Unix, every process except process 0 (the swapper) is created when another process executes the `fork()` system call. The* ***process that invoked `fork` is the parent process*** *and the newly-created process is the child process. Every process (except process 0) has one parent process, but can have many child processes.*
>
> *The operating system kernel identifies each process by its process identifier. Process 0 is a special process that is created when the system boots; after forking a child process (process 1), process 0 becomes the swapper process (sometimes also known as the "idle task"). Process 1, known as `init`, is the ancestor of every other process in the system.*

Child process

> *A child process in computing is a* ***process created by another process (the parent process)****.*

> *A child process inherits most of its attributes, such as open files, from its parent. In UNIX, a child process is in fact created (using `fork`) as a copy of the parent. The child process can then overlay itself with a different program (using exec) as required.*
>
> *Each process may create many child processes but will have at most one parent process; if a process does not have a parent this usually indicates that it was created directly by the kernel. In some systems, including UNIX based systems such as Linux, the very first process (called `init`) is started by the kernel at booting time and never terminates (see Linux startup process); other parentless processes may be launched to carry out various daemon tasks in userspace. Another way for a process to end up without a parent is if its parent dies, leaving an orphan process; but in this case it will shortly be adopted by `init`.*
>
> *System call `fork()` is used to create processes. The purpose of `fork()` is to create a new process, which becomes the child process of the caller.*

Orphan Process

> *An orphan process is a computer **process whose parent process has finished or terminated** , though it remains running itself.*
>
> *In a Unix-like operating system any orphaned process will be immediately adopted by the special `init` system process. This operation is called re-parenting and occurs automatically. Even though technically the process has the `init` process as its parent, it is still called an orphan process since the process that originally created it no longer exists.*
>
> *A process can be orphaned unintentionally, such as when the parent process terminates or crashes. The process group mechanism in most Unix-like operation systems can be used to help protect against accidental orphaning, where in coordination with the user's shell will try to terminate all the child processes with the `SIGHUP` process signal, rather than letting them continue to run as orphans.*
>
> *A process may also be intentionally orphaned so that it becomes detached from the user's session and left running in the background; usually to allow a long-running job to complete without further user attention, or to start an indefinitely running service. Under Unix, the latter kinds of processes are typically called daemon processes. The Unix `nohup` command is one means to accomplish this.*

Daemon Process

> *In Unix and other multitasking computer operating systems, a **daemon is a computer program that runs as a background process**, rather than being under the direct control of an interactive user. Typically daemon names end with the letter d: for example, `syslogd` is the daemon that implements the system logging facility and `sshd` is a daemon that services incoming SSH connections.*
>
> *In a Unix environment, the parent process of a daemon is often, but not always, the `init` process. A daemon is usually created by a process forking a child process and then immediately exiting, thus causing `init` to adopt the child process. In addition, a daemon or the operating system typically must perform other operations, such as dissociating the process from any controlling terminal (tty). Such procedures are often implemented in various convenience routines such as daemon(3) in Unix.*

Daemon process is **a process orphaned intentionally**.

Zombie Process

> *On Unix and Unix-like computer operating systems,* ***a zombie process or defunct process is a process that has completed execution but still has an entry in the process table****. This entry is still needed to allow the parent process to read its child's exit status. The term zombie process derives from the common definition of zombie — an undead person. In the term's metaphor, the child process has "died" but has not yet been "reaped". Also, unlike normal processes, the kill command has no effect on a zombie process.*
>
> *When a process ends, all of the memory and resources associated with it are deallocated so they can be used by other processes. However, the process's entry in the process table remains. The parent can read the child's exit status by executing the wait system call, whereupon the zombie is removed. The wait call may be executed in sequential code, but it is commonly executed in a handler for the* `SIGCHLD` *signal, which the parent receives whenever a child has died.*
>
> *After the zombie is removed, its process identifier (PID) and entry in the process table can then be reused. However, if a parent fails to call wait, the zombie will be left in the process table. In some situations this may be desirable, for example if the parent creates another child process it ensures that it will not be allocated the same PID. On modern UNIX-like systems (that comply with SUSv3 specification in this respect), the following special case applies: if the parent explicitly ignores* `SIGCHLD` *by setting its handler to* `SIG_IGN` *(rather than simply ignoring the signal by default) or has the* `SA_NOCLDWAIT` *flag set, all child exit status information will be discarded and no zombie processes will be left*
>
> *A* ***zombie process is not the same as an orphan process****. An orphan process is a process that is still executing, but whose parent has died. They do not become zombie processes; instead, they are adopted by* `init` *(process ID 1), which waits on its children.*