Processes

Computer Science 240
Laboratory 12

Operating System

The set of software that controls the overall operation of a computer system, typically by performing such tasks as memory allocation, job scheduling, and input/output control.

Kernel

The ***kernel*** is the center of the operating system and manages everything. It runs in *privileged* or *supervisory* mode (has access to all instructions and memory in the system) and interacts with hardware to actually load and run programs, access files, allocate memory, and spawn off processes.

Shell

The kernel has no user interface; you can only communicate with it by using another program as an intermediary, which we call a ***shell***. The shell prints a prompt, reads a line of input from the user, and then interprets it as one or more commands to manipulate files or run other programs (usually by forking another ***process*** to make a low-level call to the kernel).

Process

A ***process*** is an instance of a program in execution. A process provides the illusion that the program has exclusive use of the processor and exclusive use of the memory system. In Linux, when you run a program by typing the name of an executable object file to the shell, the shell creates a new process with the help of the kernel, and the new process actually runs the program.

Context

A program runs in the ***context*** of some process, where the context is the *state* needed to run correctly. State consists of:

- Program's code and data stored in memory
- Stack
- Registers
- Program Counter
- Environment variables
- Set of open file descriptors

Context Switch

The kernel maintains a context for each process. When the kernel pre-empts the running process with a new process or a previously running process, it is called a ***context switch***: the context of the curent process must be saved, the context of the new process must be asserted, and then control is passed to the preempting process.

Zombies

When a process terminates, it is not immediately removed from the system by the kernel. Instead, it is kept until the parent *reaps* the terminated child, at which point the kernel passes the child's exit status to the parent. Until it is reaped, it is called a zombie.

A zombie is not running, but does use memory resources to maintain some of its state.

System Calls

fork

The **fork**() function is called by a *parent* process to create a new running *child* process. The child process is almost identical to the parent (it inherits an identical (but separate) copy of the address space, and all open files). The main difference is that the child has a different PID (process ID).

Fork is called by the parent process, but returns twice: once to the parent process, returning the value of the child PID, and once to the child, with a return value of 0.

The parent and child processes run concurrently, and their instruction flows can be interleaved by the kernel in an arbitrary way.

exec

The **exec ()** function replaces the current process' code and context (registers, memory) with that of a different program.

waitpid

The **waitpid(pid)** function pauses execution of the process which calls it, and waits until the process with the specified pid terminates. It can be used to enforce a given order of execution for different processes.

getpid

The **getpid()** function returns the pid of the process that calls it.

Diagrams for Understanding Process Execution