Operating Systems II

Process Management

Process Management

- We already talked about the `task_struct` data structure that holds all of the information about the task.
- Also talked about
  - the states a task can exist in.
  - the queues the task can be held in.
- What about the other aspects of process management.

Task Creation

- Every task is created from another task.
  - Maybe from the system idle task?
- A new task is created when a parent process invokes either a `fork()` or `clone()` system call.
  - `clone()` – inherits more from the parent task.
- The system calls eventually call the `do_fork()` function that creates a new task.

`do_fork()`

- Allocate a new `task_struct` data structure.
- Link the `task_struct` into the process table.
- Create a new kernel space stack for execution when inside the kernel.
- Copy the fields from the parent's `task_struct` into the child's.

`do_fork()` (cont.)

- Modify the child's task_struct fields specific to itself.
  - New process identifier
  - Links to the tasks parent and siblings
  - initialize the process specific timers. (creation, time quantum, and so on)

`do_fork()` (cont.)

- Copy other data structures that are in parent and should be replicated for the child task.
  - File table and new file descriptor for each open file.
  - Create a new user data segment and copy the data.
  - Copy signal and signal handling information.
  - Copy virtual memory tables.
- Change the child's state to `RUNNING`.

Task Termination

- Several ways to terminate a task
  - task making the `exit()` system call.
  - delivered a signal with the signal handler disposition to die.
  - Being forced to die under certain exceptions.
- Termination work is done by `do_exit()`.
- What does `do_exit()` do?

do_exit()

- Sets a global kernel lock.
- Sets the task state to ZOMBIE
- Notifies any child of process termination.
- Notifies the parent with an exit signal (usually SIGCHILD).
- Releases any resources allocated by do_fork() (such as open files, ...)
- Calls schedule() (never returns).

Task Scheduling

- Tasks are scheduled via the `schedule()` function.
- What does `schedule()` do?
  - Exactly what happens is dependent on the scheduling algorithm.

`Schedule()` - Overview

- Performs various periodic work (such as running the bottom halves of interrupt handlers)
- Inspect the set of tasks in the running state.
- Chose one task to execute based on the scheduling policy.
- Dispatch the task to run on the CPU until an interrupt occurs.

Schedule()

- Release the global kernel lock.
- Do any software interrupts (2nd stage).
- Grab current process and current CPU.
- Set state appropriately.
- Find next process to schedule.
- Use switch_to() macro to perform the transfer (saves state of old task, loads state of new task).
  - Architecture Specific!!!

Process Table Support

- Some common functions found in a kernel to assist process management
  - `init_task` – The first task in the process table.
  - `find_task_by_pid(x)` – given a pid will return the `task_struct` pointer.
  - `for_each_task(p)` – For loop construct that will allow you to iterate over each task in the process table.