Lecture 26: Process creation in xv6

Mythili Vutukuru
IIT Bombay
https://www.cse.iitb.ac.in/~mythili/os/
New process creation in xv6

- Init process: first process created by xv6 after boot up
  - This init process forks shell process, which in turn forks other processes to run user commands
  - The init process is the ancestor of all processes in Unix-like systems
- After init, every other process is created by the fork system call, where a parent forks/spawns a child process
- The function “allocproc” called during both init process creation and in fork system call
  - Allocates new process structure, PID etc
  - Sets up the kernel stack of process so that it is ready to be context switched in by scheduler
allocproc

- Find unused entry in ptable, mark is as embryo
  - Marked as runnable after process creation completes
- New PID allocated
- New memory allocated for kernel stack
- Go to bottom of stack, leave space for trapframe (more later)
- Push return address of “trapret”
- Push context structure, with eip pointing to function “forkret”
- Why? When this new process is scheduled, it begins execution at forkret, then returns to trapret, then returns from trap to userspace
- Allocproc has created a hand-crafted kernel stack to make the process look like it had a trap and was context switched out in the past
  - Scheduler can switch this process in like any other
Init process creation

- Alloc proc has created new process
  - When scheduled, it runs function forkret, then trapret
- Trapframe of process set to make process return to first instruction of init code (initcode.S) in userspace
- The code “initcode.S” simply performs “exec” system call to run the init program
Init process

- Init program opens STDIN, STDOUT, STDERR files
  - Inherited by all subsequent processes as child inherits parent’s files
- Forks a child, execs shell executable in the child, waits for child to die
- Reaps dead children (its own or other orphan descendants)
Forking new process

- Fork allocates new process via allocproc
- Parent memory and file descriptors copied (more later)
- Trapframe of child copied from that of parent
  - Result: child returns from trap to exact line of code as parent
  - Different physical memory but same virtual address (location in code)
  - Only return value in eax is changed, so parent and child have different return values from fork
- State of new child set to runnable, so scheduler thread will context switch to child process sometime in future
- Parent returns normally from trap/system call, child runs later when scheduled
Summary of new process creation

• New process created by marking a new entry in ptable as RUNNABLE, after configuring the kernel stack, memory image etc of new process

• Neat hack: kernel stack of new process made to look like that of a process that had been context switched out in the past, so that scheduler can context switch it in like any other process
  – No special treatment for newly forked process during “swtch”