



Embedded Operating Systems

Trainer: Kiran Jaybhave

• Process Creation System Calls

- Windows: CreateProcess()
- UNIX: fork()
- BSD UNIX: fork(), vfork()
- Linux: clone(), fork(), vfork()

• fork() syscall

- To execute certain task concurrently we can create a new process (using fork() on UNIX).
- fork() creates a new process by duplicating calling process.
- The new process is called as "child process", while calling process is called as "**parent process**". "child" process is exact duplicate of the "parent" process except few points pid, parent pid, etc.
- **pid = fork();**
 - On success, fork() returns pid of the child to the parent process and 0 to the child process.
 - On failure, fork() returns -1 to the parent.
- Even if child is copy of the parent process, after its creation it is independent of parent and both these processes will be scheduled separately by the scheduler.
- Based on CPU time given for each process, both processes will execute concurrently.



• **How fork() return two values i.e. in parent and in child?**

- fork() creates new process by duplicating calling process.
- The child process PCB & kernel stack is also copied from parent process. So child process has copy of execution context of the parent.
- Now fork() write 0 in execution context (r0 register) of child process and child's pid into execution context (r0 register) of parent process.
- When each process is scheduled, the execution context will be restored (by dispatcher) and r0 is return value of the function.

• **getpid() vs getppid()**

- pid1 = getpid(); // returns pid of the current process
- pid2 = getppid(); // returns pid of the parent of the current process



• When fork() will fail?

- When no new PCB can be allocated, then fork() will fail.
- Linux has max process limit for the system and the user.
- When try to create more processes, fork() fails. terminal> cat /proc/sys/kernel/pid_max

• Orphan process

- If parent of any process is terminated, that child process is known as orphan process.
- The ownership of such orphan process will be taken by "init" process.

• Zombie process

- If process is terminated before its parent process and parent process is not reading its exit status, then even if process's memory/resources is released, its PCB will be maintained. This state is known as "**zombie state**".
- To avoid zombie state parent process should read exit status of the child process. It can be done using wait() syscall.



• **wait() syscall**

- **ret = wait(&s);**
 - arg1: out param to get exit code of the process.
 - returns: pid of the child process whose exit code is collected.
- **wait() performs 3 steps:**
 - Pause execution parent until child process is terminated.
 - Read exit code from PCB of child process & return to parent process (via out param).
 - Release PCB of the child process.
- The exit status returned by the wait() contains exit status, reason of termination and other details.
- Few macros are provided to access details from the exit code.
 - **WEXITSTATUS()**

• **waitpid() syscall**

- This extended version of wait() in Linux.
- **ret = waitpid(child_pid, &s, flags);**
 - arg1: pid of the child for which parent should wait.
 - -1 means any child.
 - arg2: out param to get exit code of the process.
 - arg3: extra flags to define behaviour of waitpid().
- returns: pid of the child process whose exit code is collected.
 - -1: if error occurred.



• exec() syscall

- syscall exec() syscall "loads a new program" in the calling process's memory (address space) and replaces the older (calling) one.
- If exec() succeed, it does not return (rather new program is executed).
- There are multiple functions in the family of exec ():
 - execl(),execlp(), execle(),
 - execv(), execvp(), execve(), execvpe()
 - exec() family multiple functions have different syntaxes but same functionality.





Thank you!

Kiran Jaybhave

email – kiran.jaybhave@sunbeaminfo.com

