Unix System Programming

Process Control

Chandravva Hebbi
Department of Computer Science and Engineering
chandravvahebbi@pes.edu

Unix System Programming

Process Control

Chandravva Hebbi

Department of Computer Science and Engineering

UNIX SYSTEM PROGRAMMING

Topics to be Covered

❖Process Identifiers

❖fork()

❖vfork()

❖exec()

❖Programming examples

Process Identifiers

- Every process has a unique process ID, a non-negative integer.
- Identifier of a process is always unique.
- process IDs are reused.
- Delay in reuse.
- special processes: Process ID 0 is a swapper process(scheduler Process).
- Process ID 1: init it is invoked by the kernel at the end of bootstrap.

Process Identifiers

- The program file for this process was /etc/init in older versions of the UNIX System and /sbin/init in newer versions.
- This process is responsible for bringing up a UNIX system after the kernel has been bootstrapped.
- init usually reads the system-dependent initialization files the /etc/rc* files or /etc/inittab and the files in /etc/init.d and brings the system to a certain state, such as multiuser.
- The init process never dies. It is a normal user process, not a system process within the kernel, like the swapper.
- It runs with superuser privileges.

Process Identifiers

- process ID 2 is the *pagedaemon*.It supports paging for Virtual memory system
- Other processes and their id’s

```
#include <unistd.h>
pid_t getpid(void);
                         Returns: process ID of calling process
pid_t getppid(void);
                         Returns: parent process ID of calling process
uid_t getuid(void);
                           Returns: real user ID of calling process
uid_t geteuid(void);
                        Returns: effective user ID of calling process
gid_t getgid(void);
                        Returns: real group ID of calling process
gid_t getegid(void);
                       Returns: effective group ID of calling process
```

Note that none of these functions has an error return

fork Function

- An existing process can create a new one by calling the fork function.

```
#include <unistd.h>

pid_t fork(void);

        Returns: 0 in child, process ID of child in parent, -1 on error
```

- fork() creates a new process called child.
- Returns twice: 0 in child and child id in parent
- No user process with ID 0

fork Function

- The child is a copy of the parent. The child gets a copy of the parent's data space, heap, and stack.
- The parent and the child do not share these portions of memory.
- The parent and the child share the text segment.
- Current implementations don't perform a complete copy of the parent's data, stack, and heap, since a fork is often followed by an exec.
- Instead, a technique called copy-on-write (COW) is used.
- These regions are shared by the parent and the child and have their protection changed by the kernel to read-only.
- Copy is given when one of the process tries to modify.

Variations in the fork function

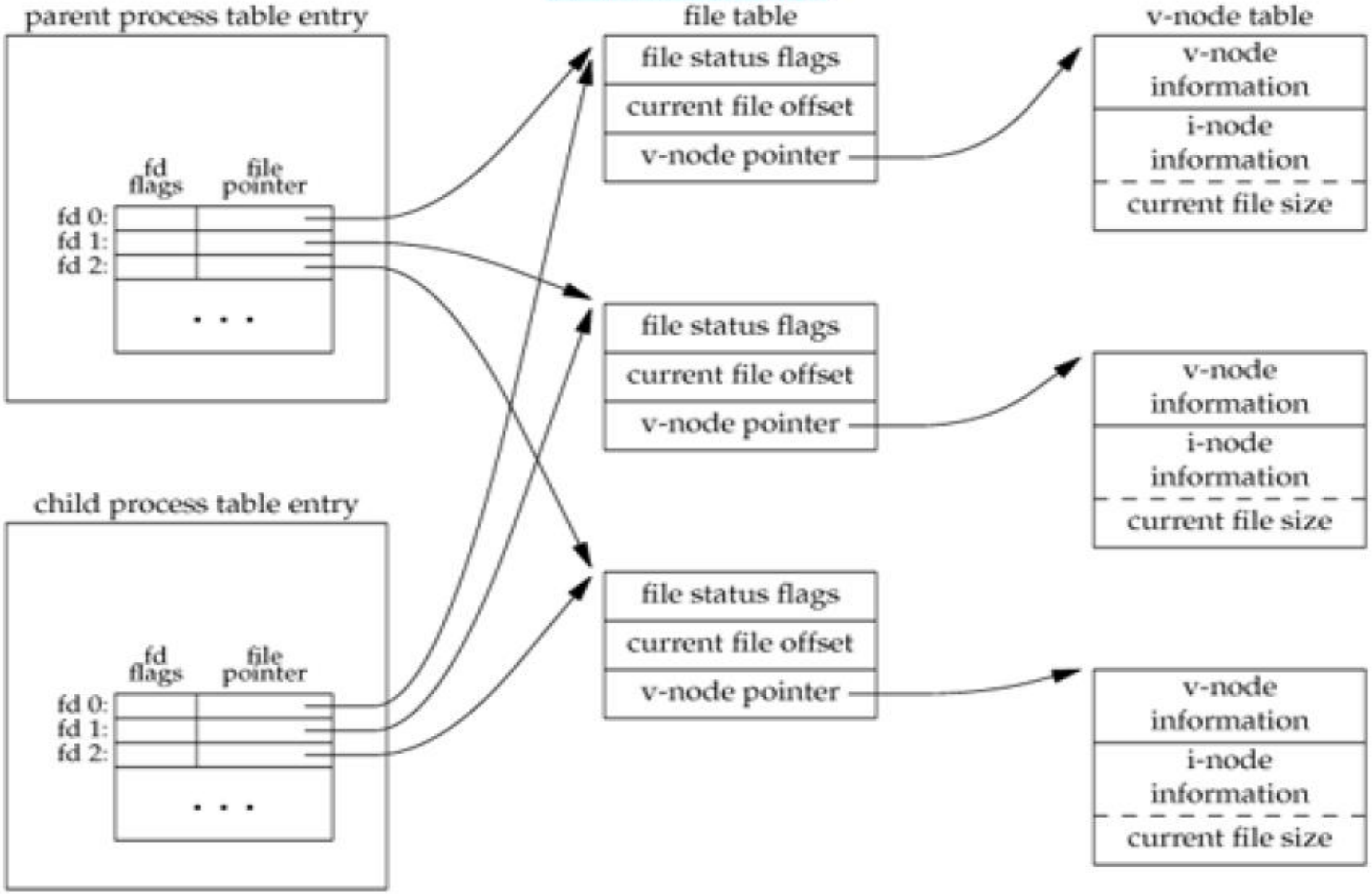
- Linux 2.4.22 also provides new process creation through the clone(2) system

  call
- FreeBSD 5.2.1 provides the rfork(2) system call.
- Solaris provides the fork1 function: creates a process that duplicates only the calling thread.

UNIX SYSTEM PROGRAMMING

Fork function: File Sharing

[View full size image]



Fork function: File Sharing

Two normal cases for handling the descriptors after a fork.

- The parent waits for the child to complete
- Both the parent and the child go their own ways.

UNIX SYSTEM PROGRAMMING

fork function: File Sharing

Properties of the parent that are inherited by the child:

Real user ID, real group ID, effective user ID, effective group ID

Supplementary group IDs

Process group ID

Session ID

Controlling terminal

The set-user-ID and set-group-ID flags

Current working directory

Root directory

File mode creation mask

Signal mask and dispositions

The close-on-exec flag for any open file descriptors

Environment

Attached shared memory segments

Memory mappings

Resource limits

fork function: File Sharing

The differences between the parent and child are.

- The return value from fork
- The process IDs are different
- The two processes have different parent process IDs: the parent process ID of the child is the
- parent; the parent process ID of the parent doesn't change
- The child's tms_utime, tms_stime, tms_cutime, and tms_cstime values are set to 0
- File locks set by the parent are not inherited by the child
- Pending alarms are cleared for the child
- The set of pending signals for the child is set to the empty set

Fork function

Reasons for fork failure

(a) if too many processes are already in the system

(b) if the total number of processes for this real user ID exceeds the system's limit.

Uses for fork

Application 1: Networks

- When a process wants to duplicate itself so that the parent and child can each execute different sections of code at the same time.
- Application: In network servers the parent waits for a service request from a client.
- When the request arrives, the parent calls fork and lets the child handle the request.
- The parent goes back to waiting for the next service request to arrive.

Application 2: shells

- When a process wants to execute a different program.
- This is common for shells.
- In this case, the child does an exec right after it returns from the fork.

PES UNIVERSITY ONLINE

THANK YOU

Chandravva Hebbi
Department of Computer Science and Engineering

chandravvahebbi@pes.edu