CSC552 – Advanced UNIX Programming

Processes

Dr. L. Frye
Kutztown University

Processes

- pid
- Process table
- Parent process (ppid)

- System calls
  - getpid
  - getppid
  - getuid, geteuid
  - getgid, getegid
  - setuid, seteuid, setgid, setegid

Process Internals

- Scheduler
- Memory manager
- Magic number

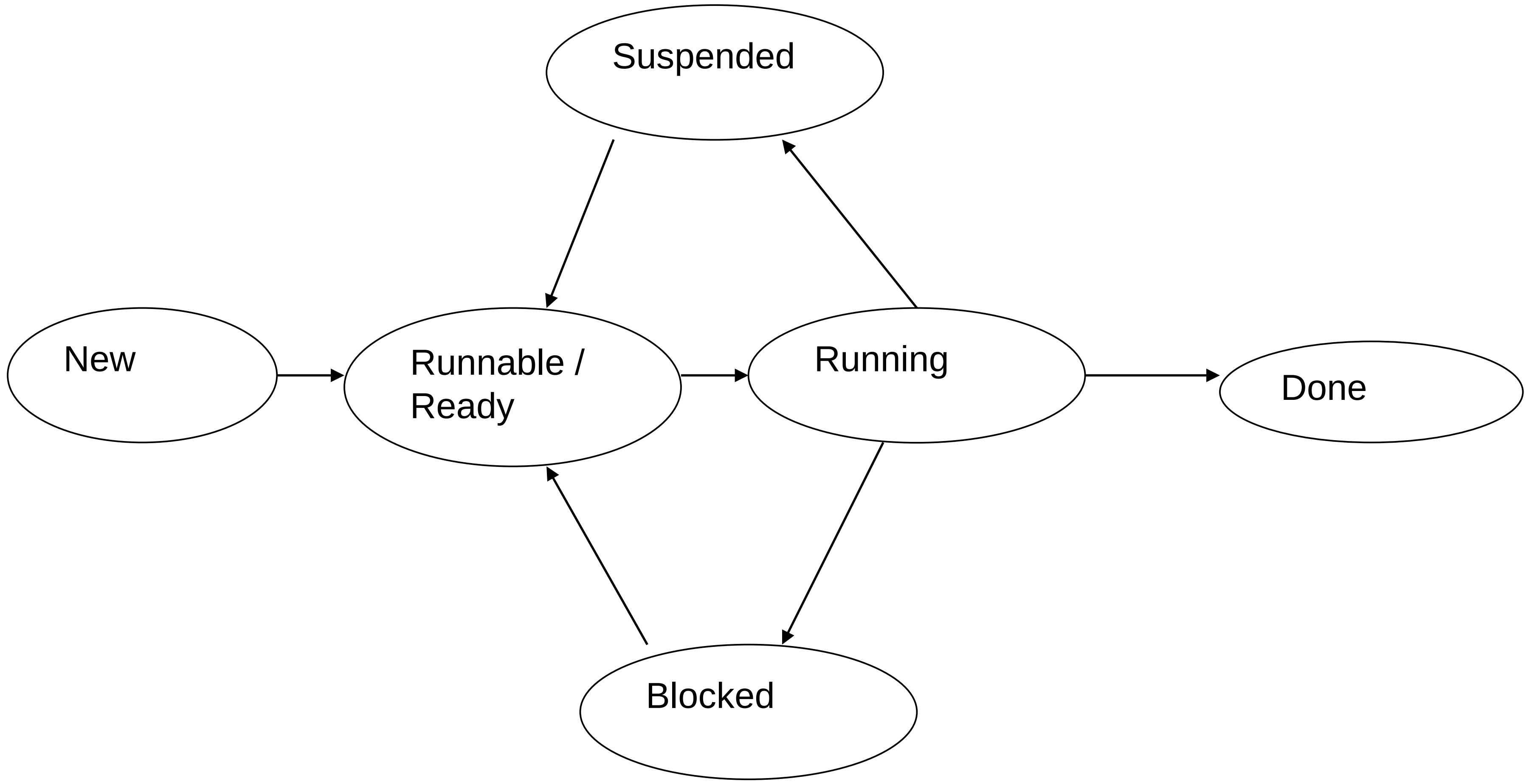
- New process – duplicate existing one

Special Processes

- sched
- init
- pageout
- getty
- login

- Modes
  - User
  - Kernel

Process States



Process Termination

- Deallocate resources
- Normal
  - Return
  - Exit
  - End of main function (implicit return)
- Abnormal
  - Abort function
  - Signal

- What is a zombie process?

- What is an orphan process?

Process Areas

- Code area
- Data area
- Stack area
- User area
- Page tables

Process Table

- PID
- PPID
- Real and effective UID and GID
- Process state
- Location of code, data, stack and user area
- Pending signals

Scheduler

- Scheduling Algorithms
  - First Come First Serve (FCFS)
  - Shortest Job First
  - Priority Scheduling
  - Round Robin Scheduling
  - Multilevel Queue Scheduling
    - Multilevel priority queue
- Nice value

Context Switch

- Context Switch
- Process context
  - Executable code
  - Stack
  - Memory for variables
  - Registers
  - Program counter
  - Process information

- What might cause a context switch?

Memory Management System

- Pages
- Page table
  - Modified bit
  - Referenced bit
  - Age

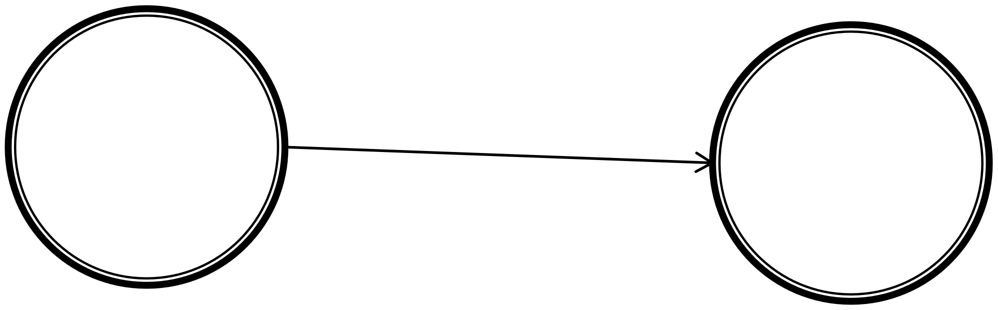
- Page daemon
- RAM table
- Swap space
- Page fault

Process Creation

- Duplicate existing process
  - fork system call
    - Returns two times
      - Parent – child’s PID
      - Child – 0

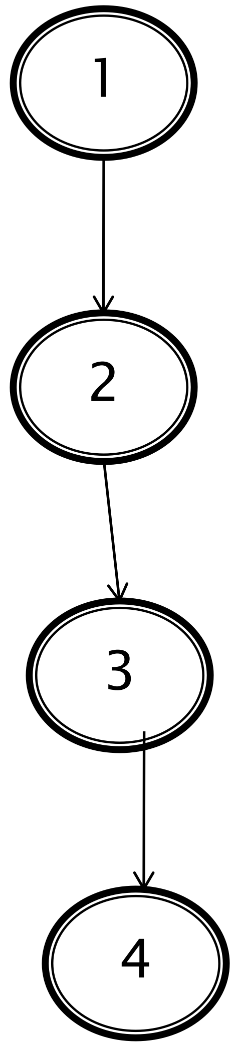
Process Chain

- Circle → process
- Edge → is-a-parent relationship



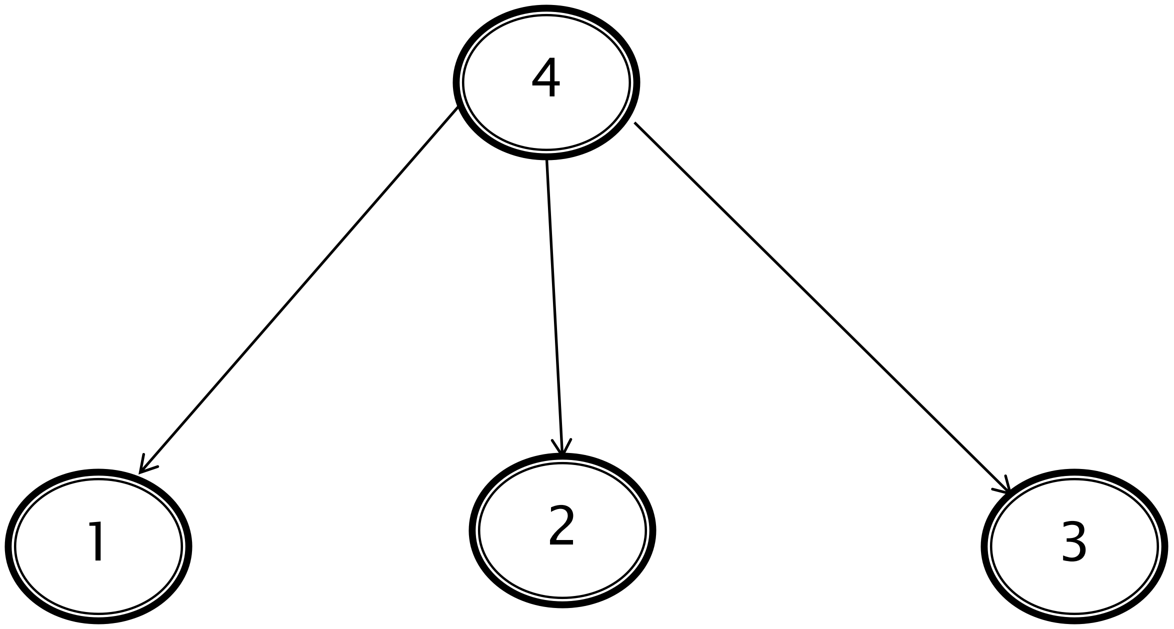
Process Chain Example 1

- processes/simplechain.c



Process Chain Example 2

- processes/simplefan.c


$$\begin{tikzcd}
 & 4 \arrow[dl] \arrow[d] \arrow[dr] & \\
1 & 2 & 3
\end{tikzcd}$$

Inherited Characteristics

- uid, gid, euid, guid
- suid and sgid bits
- Environment variables
- Open file descriptors and file offsets
- umask value
- SID and PGRP ID
- Controlling terminal
- Nice value
- Current working directory
- Resource limits

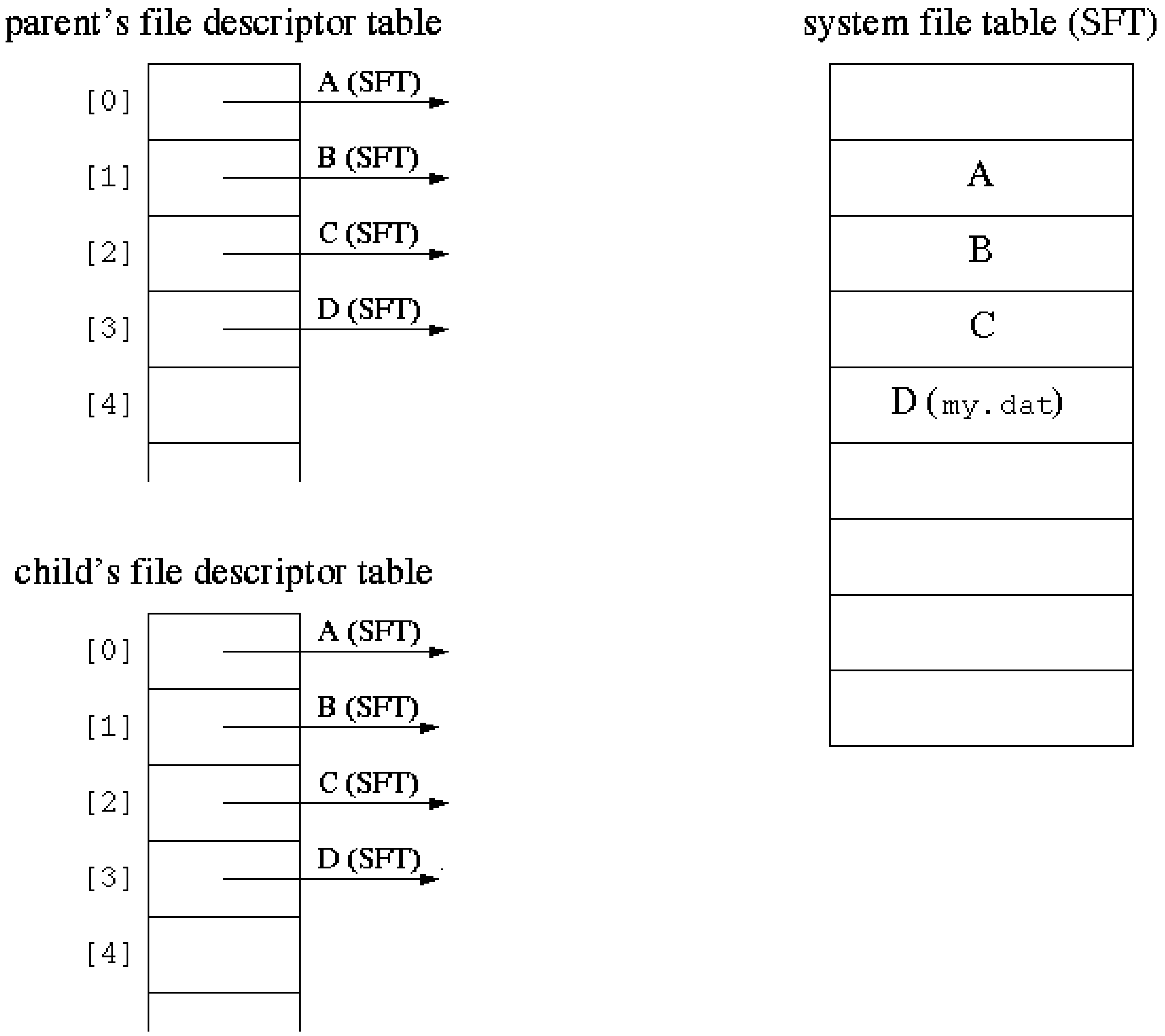
Different Characteristics

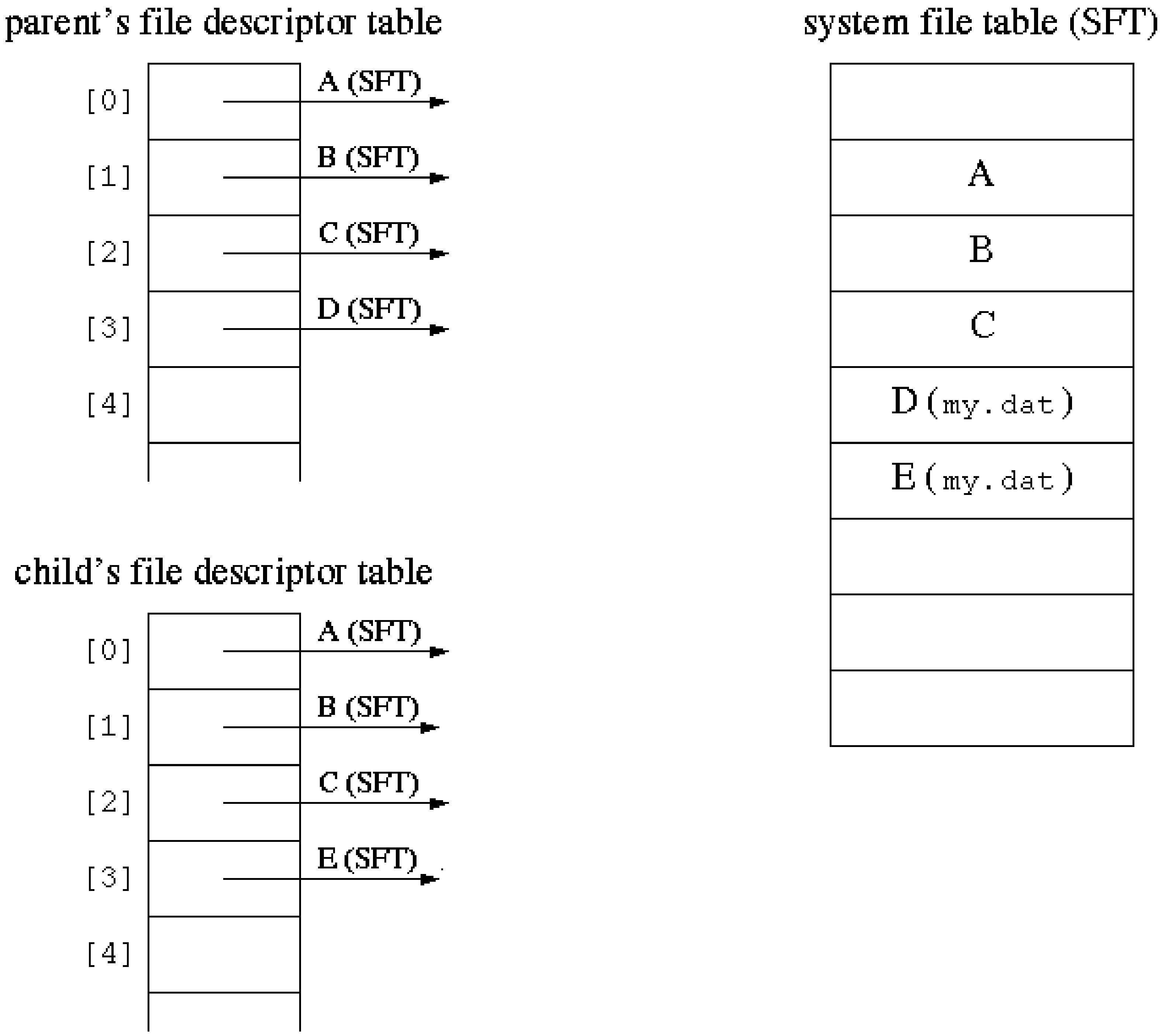
- PID
- PPID
- Own copy of parent’s file descriptors
- No file locks from parent
- Pending signal set initialized to empty set
- Own copy of parent’s data area
- Own copy of parent’s stack area

- Share code area
  - Copy–on–write (COW)

Process Examples

- processes/fork1.c
- processes/fork2.c


parent's file descriptor table
[0]
[1]
[2]
[3]
[4]
A (SFT)
B (SFT)
C (SFT)
D (SFT)
child's file descriptor table
[0]
[1]
[2]
[3]
[4]
A (SFT)
B (SFT)
C (SFT)
D (SFT)
system file table (SFT)
A
B
C
D (my.dat)


parent's file descriptor table
[0]
A (SFT)
[1]
B (SFT)
[2]
C (SFT)
[3]
D (SFT)
[4]
child's file descriptor table
[0]
A (SFT)
[1]
B (SFT)
[2]
C (SFT)
[3]
E (SFT)
[4]
system file table (SFT)
A
B
C
D (my.dat)
E (my.dat)

Terminate Process

- System calls
  - exit
  - _exit

- Why have two different exit function calls?

Termination Status

- System calls
  - wait
  - waitpid

- processes/waitpid_ex.c

Process Examples

- Create a program that would result in an orphan process.
  - processes/forkOrphan.c

- Create a program that would result in a zombie process.
  - processes/forkZombie.c

Family of exec calls

- execl
- execv
- execle
- execlp
- execvp
- execve

- processes/forkexec.c

system function

- Shorfalls
  - Inefficient
  - Security

- Calls
  - fork
  - exec
  - waitpid

- So, why use the system call?