

Operating Systems,
fall 2002

Lior Amar,
Breitgand David
(recitation)
www.cs.huji.ac.il/~os

Peculiarities of Signals (III)

- Example: signals and *read()* syscall
 - if signal is received **before** any data from the device started to arrive, the call is interrupted and the handler is called. Afterwards the call is restarted.
 - otherwise, the return value of read will not match the number of bytes requested. Errno will indicate EINTR condition.

Peculiarities of Signals (I)

- Many asynchronous events that cause signals may happen very close in time
- When a signal is caught it means only that **at least one** instance of event has occurred
- Example: a father spawns many children and continues with other stuff. When a child finishes and enters the Zombie state, SIGCHLD signal is sent to father by the kernel. More than one child may finish, but the father receives only one signal.

Signals and *fork()/exec()*

- Signal mask is inherited by a child as part of PSW
- Handlers are also available since the text segment is logically cloned.
- When *exec()* succeeds, the handlers corresponding to the signal mask of a process are cleared (why?)
- The ignored signals remain ignored after *exec()*

Peculiarities of Signals (II)

- Signals are asynchronous and thus may interfere with the system calls
- Most of the system calls are non-interruptible.
- Some system calls, *long calls*, are interruptible:
 - **read**
 - **write**
 - **wait** and some other

Who sends signals?

- One process to another using syscall *kill()*
- Kernel to its own processes and to the user's ones

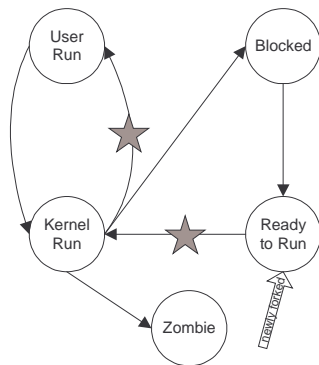
How SIGNALS are sent and received?

- *int kill(int pid, int signum);*
- Mechanism:
 - kernel accesses the entry #pid in the process table and sets “1” to the entry in a “vector of received signals”
 - Then AND is performed on the mask and this vector
 - handlers corresponding to the “1” entries of the resulting vector are called sequentially.

wait()/waitpid()

- wait() blocks the caller
- Using wait() one cannot choose a specific process to be “waited”
- waitpid() does not block the caller if WNOHANG option is used
- waitpid() allows to choose a specific target process (or group of processes)
- Both wait() and waitpid() do not allow getting the resource information about the finished son proc.

When SIGNALS are received?



wait3()/wait4()

- wait3() extends wait() by:
 - Allowing to specify an option for non-blocking
 - Allowing to read the resource information
- wait4() extends waitpid() by:
 - Allowing to read the resource information
- Note:
 - Both wait3() and wait4() are applicable to your ex1.
 - wait() and waitpid() are not!
 - Read the manual pages on wait functions family (man 2 wait)

Signals Limitations

- Signals can be used only on the same machine
- Parameters (data) other than the signal number cannot be passed to handlers
- Therefore used only in order to communicate various conditions asynchronously

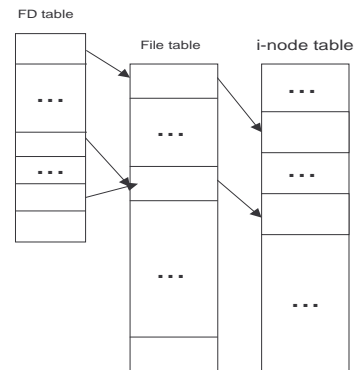
File Descriptor

- Programs refer to files using non-negative integer values termed “file descriptors”.
- Every process is allocated a table (maintained by the kernel), known as “file descriptors table”
- Individual file descriptors are simply indices into this table
- File descriptors have no meaning across different processes

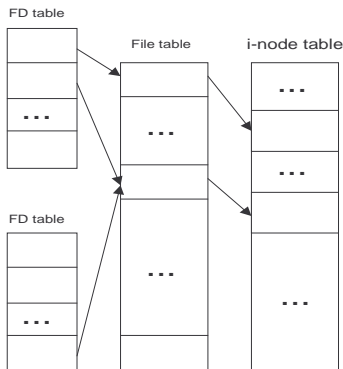
Open Files and *exec()*

- Some information about the process that calls *exec()* is retained by the process after successful *exec()* system call
- Example: PID, sigmask, pending signals, resource limits
- Open files may be left open after *exec()*, this depends on whether `FD_CLOEXEC` flag is set for the descriptor
- By default `FD_CLOEXEC` is not set

After *dup()*



File System



Example

```
int fd;  
...  
fd = open("foo", O_WRONLY);  
dup2(fd, 1);  
close(fd);  
...  
What will happen?
```

dup() and *dup2()*

```
include <unistd.h>
```

```
int dup(int fd)
```

- Duplicates the specified *fd* into the first lowest available file descriptor number. Returns *fd* of the copy.

```
int dup2(int fd1, int fd2)
```

- Duplicates *fd1* into *fd2*. Closes *fd2* first.
- Read more:
 - Richard W. Stevens "Adv. Unix Progr.", pp.61-63
 - Maurice Bach "Unix", pp. 22-24, 91-108, 117-119