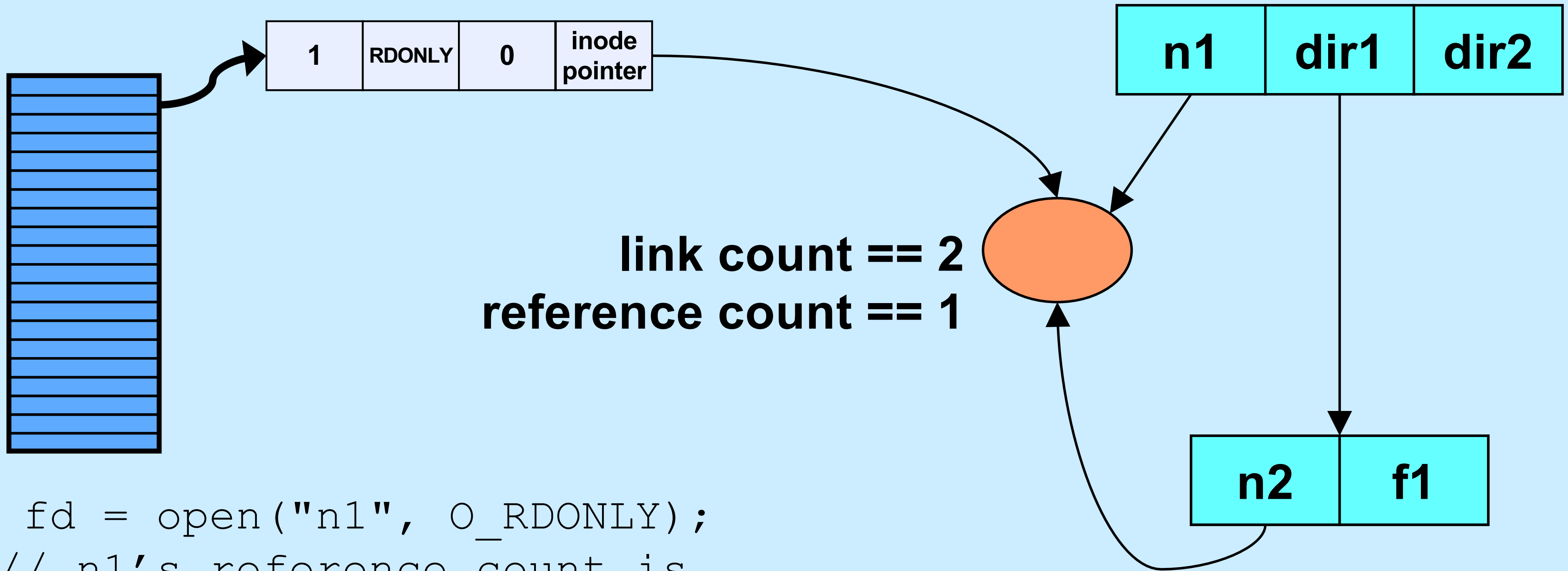
CS 33

Files Part 4

 Copyright © 2023 Thomas W. Doeppner. All rights reserved.

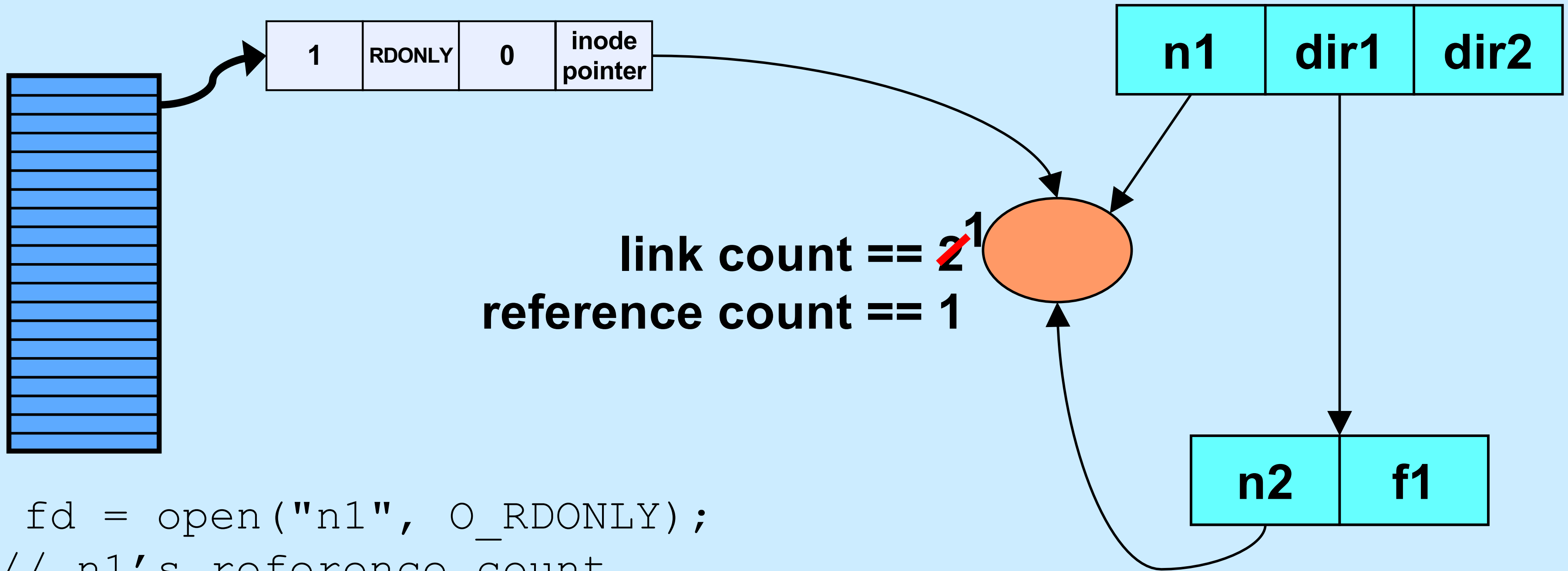
Link and Reference Counts




```
int fd = open("n1", O_RDONLY);
   // n1's reference count is
   // incremented by 1
```


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

Link and Reference Counts

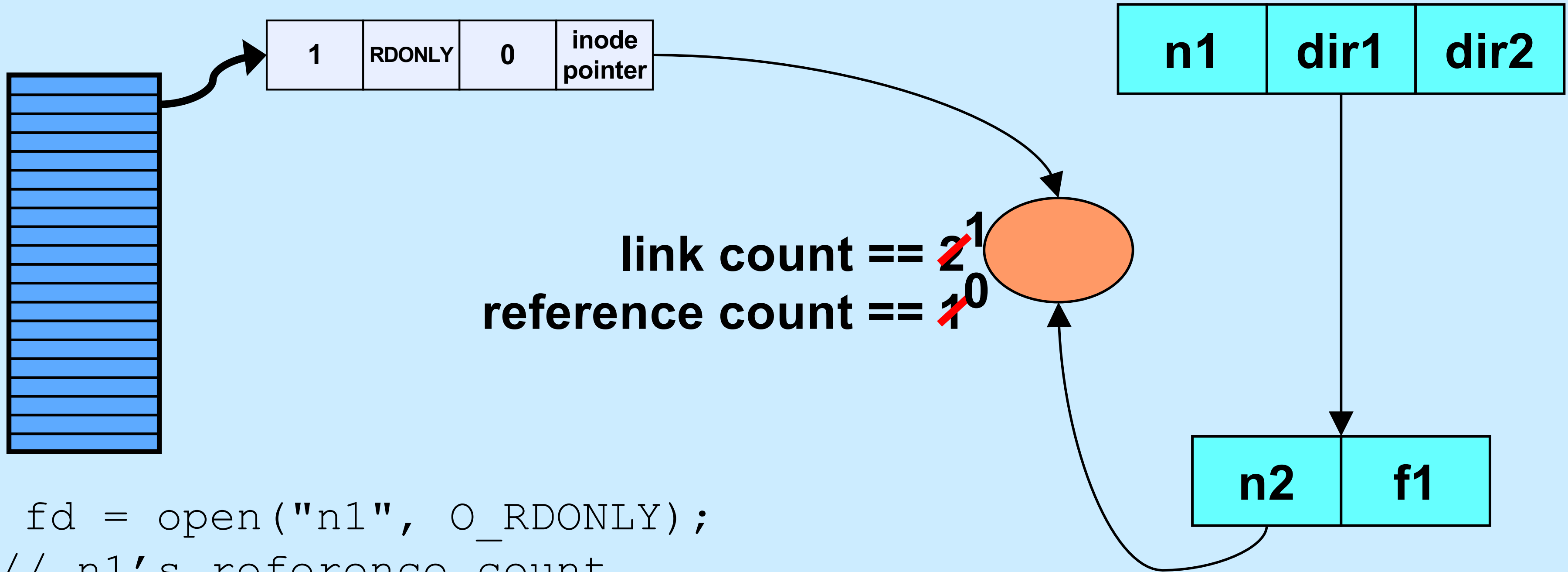



```
int fd = open("n1", O_RDONLY);
   // n1's reference count
   // incremented by 1

unlink("n1");
   // link count decremented by 1
   // same effect in shell via "rm n1"
```


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

Link and Reference Counts



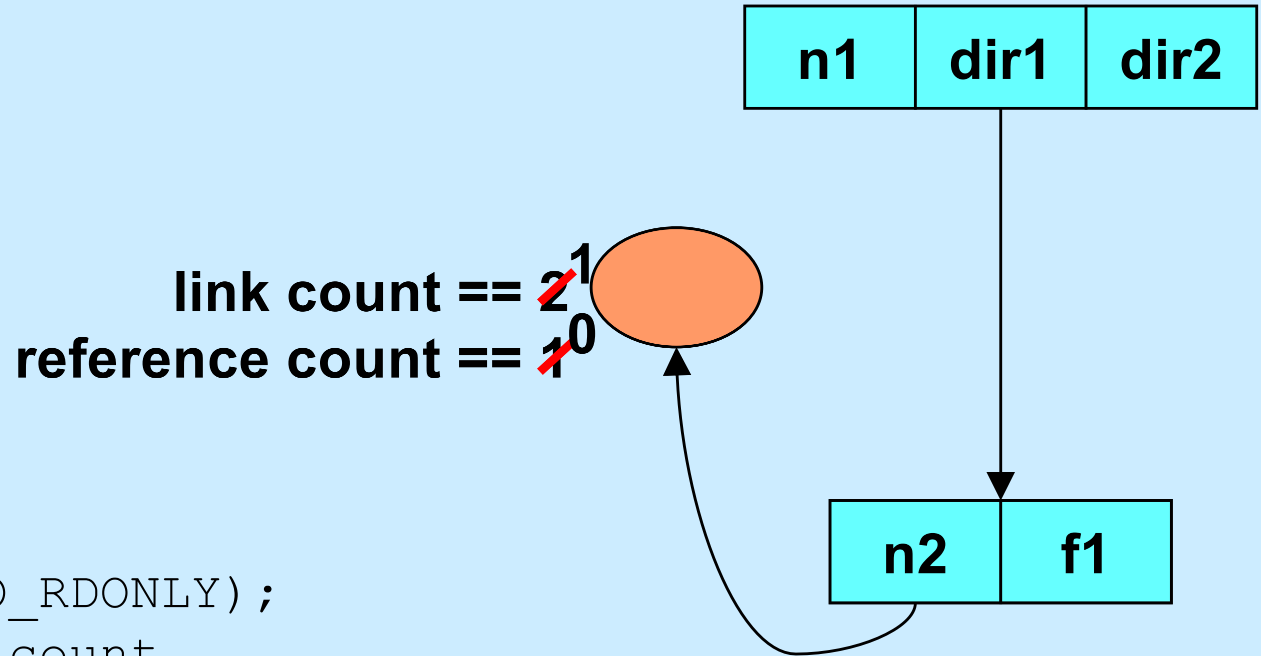

```
int fd = open("n1", O_RDONLY);
   // n1's reference count
   // incremented by 1

unlink("n1");
   // link count decremented by 1

close(fd);
   // reference count decremented by 1
```

 Copyright © 2023 Thomas W. Doeppner. All rights reserved.

Link and Reference Counts



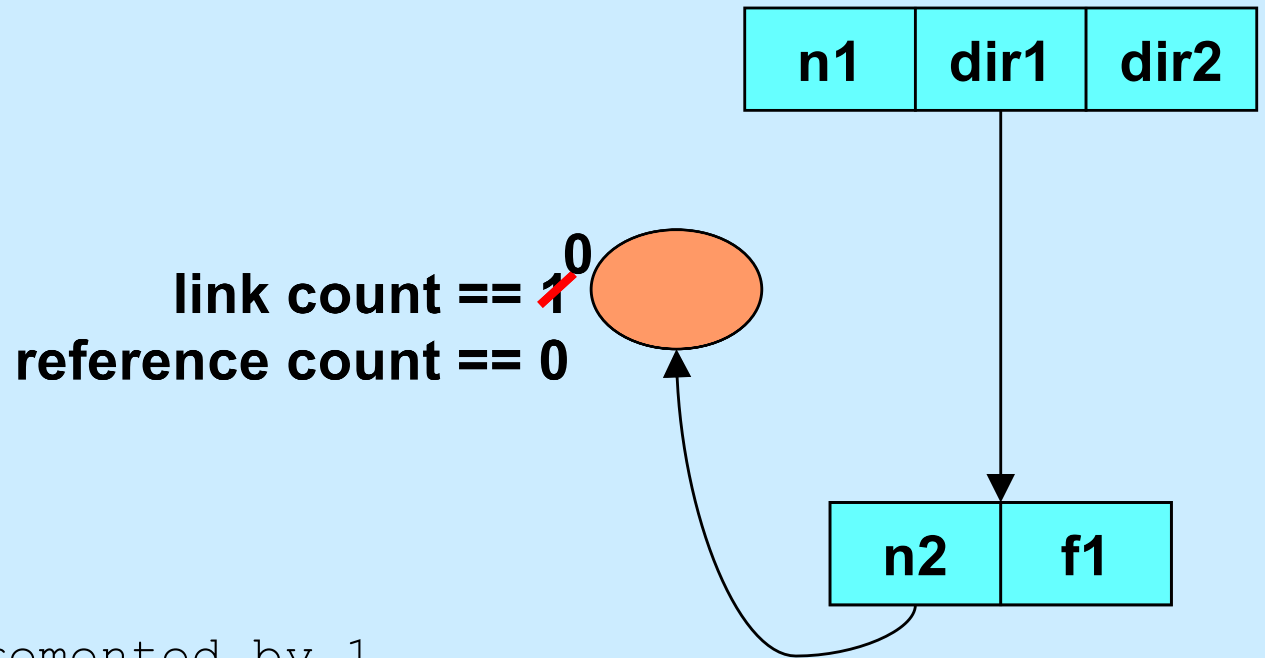

```
int fd = open("n1", O_RDONLY);
   // n1's reference count
   // incremented by 1

unlink("n1");
   // link count decremented by 1

close(fd);
   // reference count decremented by 1
```

 Copyright © 2023 Thomas W. Doeppner. All rights reserved.

Link and Reference Counts




```
unlink("dir1/n2");
   // link count decremented by 1
```


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

Quiz 1

```
int main() {
  int fd = open("file", O_RDWR|O_CREAT, 0666);
  unlink("file");
  PutStuffInFile(fd);
  GetStuffFromFile(fd);
  return 0;
}
```

Assume that *PutStuffInFile* writes to the given file, and *GetStuffFromFile* reads from the file.

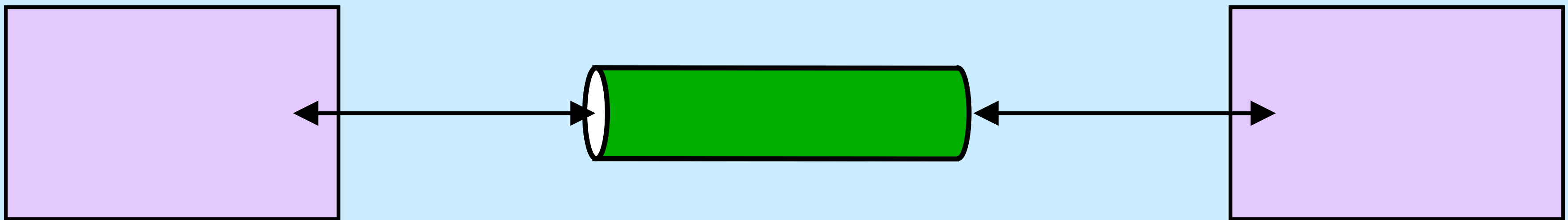
a) This program is doomed to failure, since the file is deleted before it's used

b) Because the file is used after the unlink call, it won't be deleted

c) The file will be deleted when the program terminates

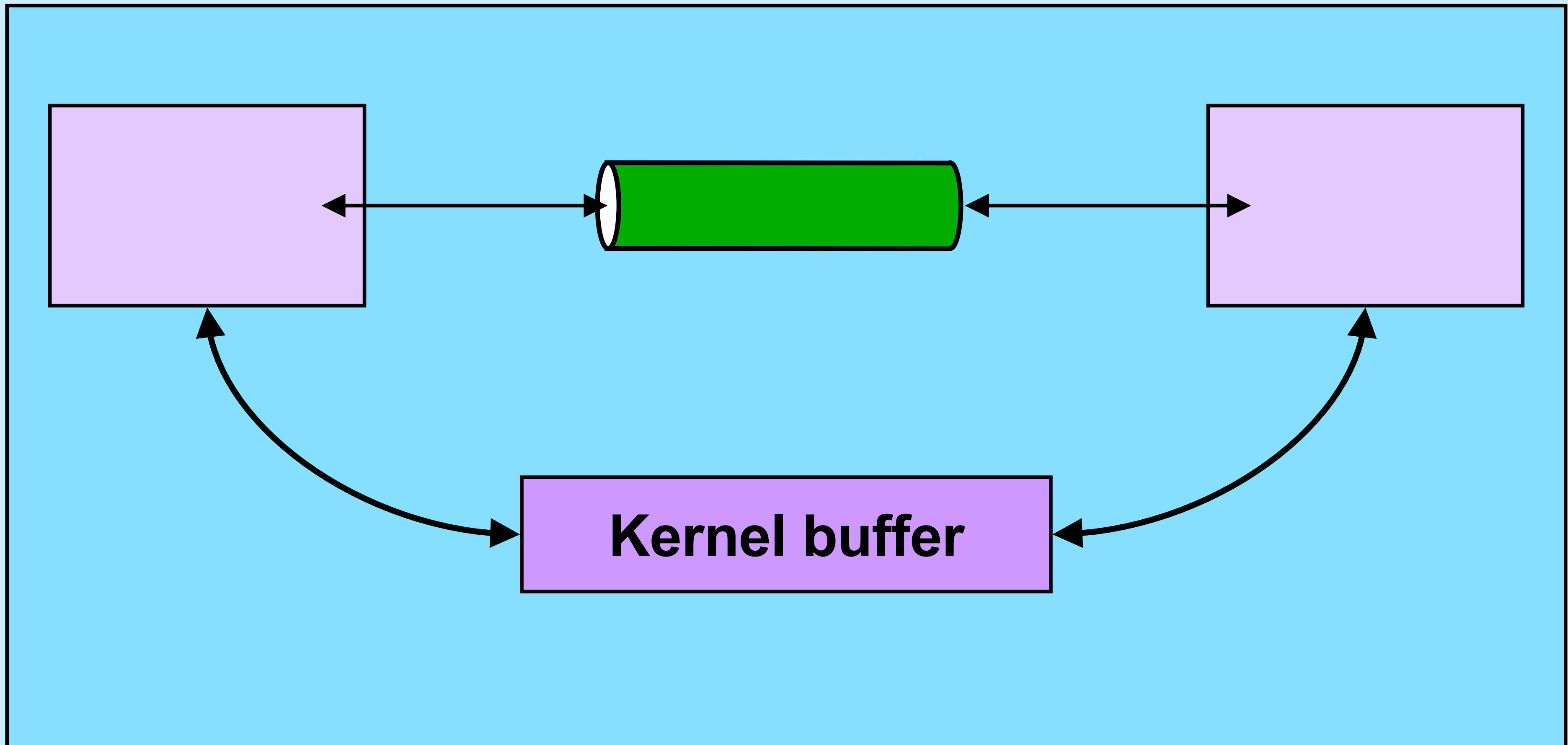
Copyright © 2023 Thomas W. Doeppner. All rights reserved.

Interprocess Communication (IPC): Pipes



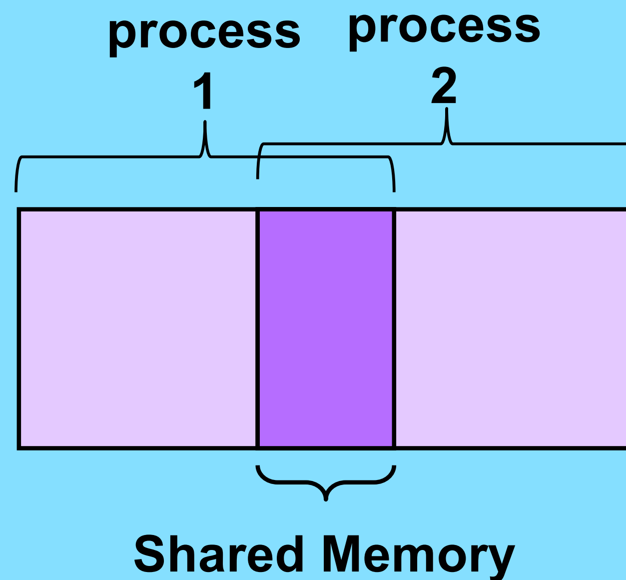
Copyright © 2023 Thomas W. Doeppner. All rights reserved.

Interprocess Communication: Same Machine I



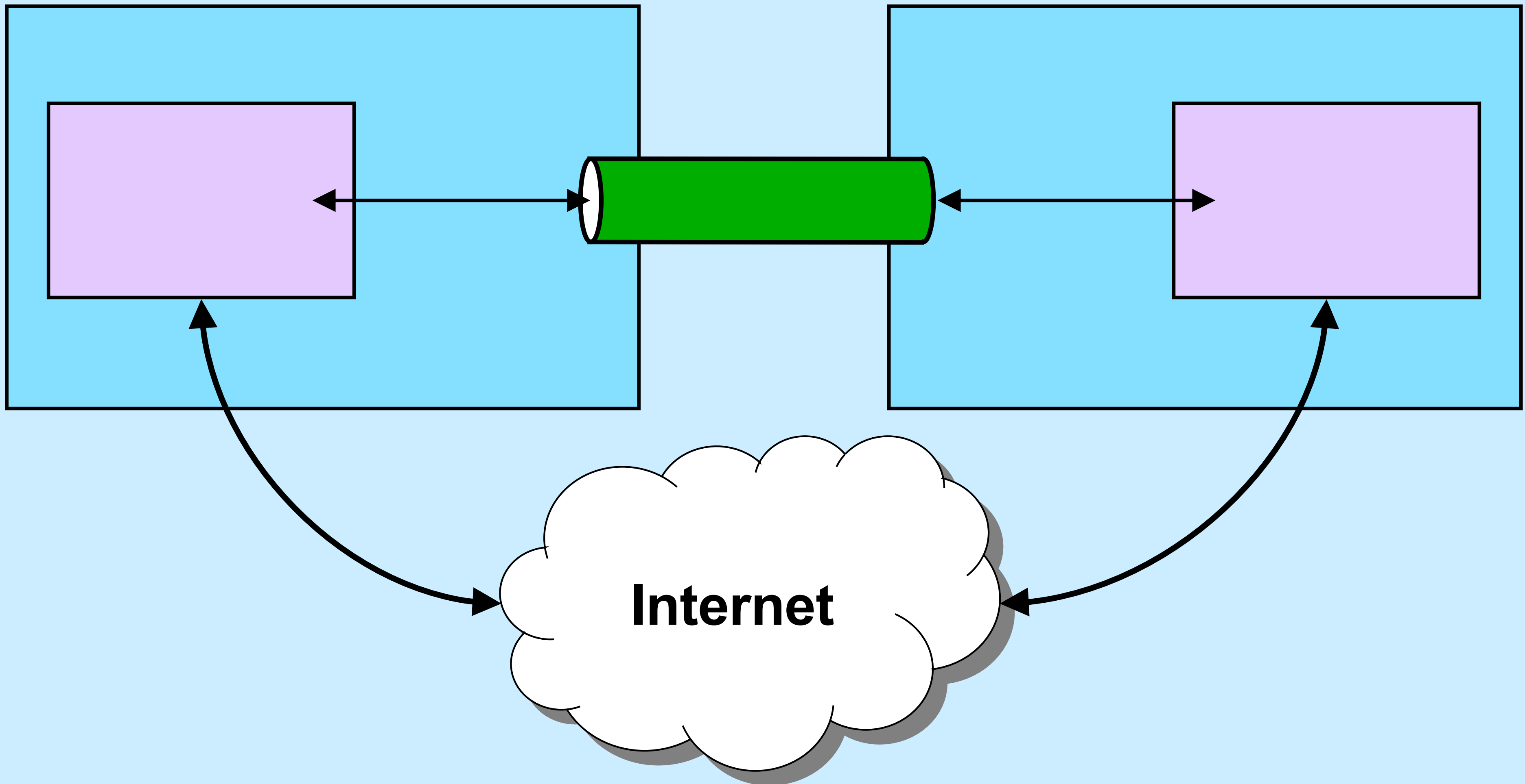
 Copyright © 2023 Thomas W. Doeppner. All rights reserved.

Interprocess Communication: Same Machine II



 Copyright © 2023 Thomas W. Doeppner. All rights reserved.

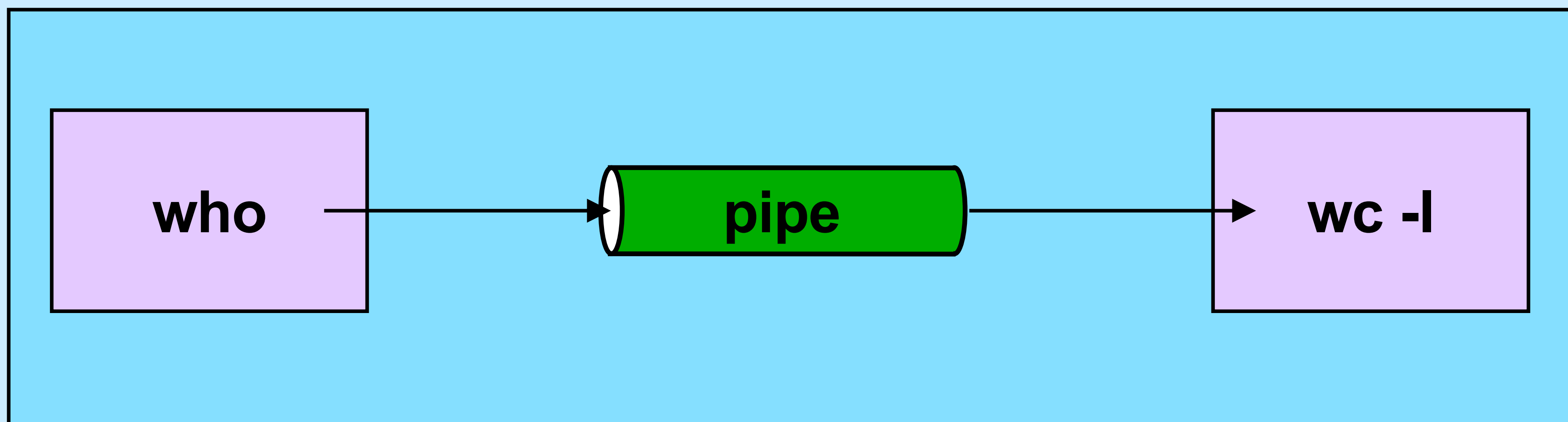
Interprocess Communication: Different Machines



Copyright © 2023 Thomas W. Doeppner. All rights reserved.

Pipes

```
$cslab2e who | wc -l
```



Copyright © 2023 Thomas W. Doeppner. All rights reserved.

Using Pipes in C

```
$cslab2e who | wc -l
```

```
int fd[2];
pipe(fd);
if (fork() == 0) {
  close(fd[0]);
  close(1);
  dup(fd[1]); close(fd[1]);
  execl("/usr/bin/who", "who", 0); // who sends output to pipe
}
if (fork() == 0) {
  close(fd[1]);
  close(0);
  dup(fd[0]); close(fd[0]);
  execl("/usr/bin/wc", "wc", "-l", 0); // wc’s input is from pipe
}
close(fd[1]); close(fd[0]);
// …
```


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

Shell 1: Artisanal Coding

```
while ((line = get_a_line()) != 0) {
    tokens = parse_line(line);
    for (int i=0; i < ntokens; i++) {
        if (strcmp(tokens[i], ">") == 0) {
            // handle output redirection
        } else if (strcmp(tokens[i], "<") == 0) {
            // handle input redirection
        } else if (strcmp(tokens[i], "&") == 0) {
            // handle "no wait"
        } ... else {
            // handle other cases
        }
    }
    if (fork() == 0) {
        // ...
        execv(...);
    }
    // ...
}
```


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

Shell 1: Non-Artisanal Coding (1)

```
while ((line = get_a_line()) != 0) {
    tokens = parse_line(line);
    for (int i=0; i < ntokens; i++) {
        // handle "normal" case
    }
    if (fork() == 0) {
        // ...
        execv(...);
    }
    // ...
}
```


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

Shell 1: Non-Artisanal Coding (2)

```
next_line: while ((line = get_a_line()) != 0) {
    tokens = parse_line(line);
    for (int i=0; i < ntokens; i++) {
        if (redirection_symbol(token[i])) {
            // ...
            if (fork() == 0) {
                // ...
                execv(...);
            }
            // ...
            goto next_line;
        }
        // handle "normal" case
    }
    if (fork() == 0) {
        // ...
        execv(...);
    }
    // ...
}
```

whoops!

(whoops!)

Copyright © 2023 Thomas W. Doeppner. All rights reserved.

Shell 1: Non-Artisanal Coding (3)

```
next_line: while ((line = get_a_line()) != 0) {
    tokens = parse_line(line);
    for (int i=0; i < ntokens; i++) {
        if (redirection_symbol(token[i])) {
            // ...
            if (fork() == 0) {
                // ...
                execv(...);
            }
            // ... deal with &
            goto next_line;
        }
        // handle "normal" case
    }
    if (fork() == 0) {
        // ...
        execv(...);
    }
    // ... also deal with & here!
}
```

 Copyright © 2023 Thomas W. Doeppner. All rights reserved.

Shell 1: Non-Artisanal Coding (Worse)

```
next_line: while ((line = get_a_line()) != 0) {
tokens = parse_line(line);
for (int i=0; i < ntokens; i++) {
if (redirection_symbol(token[i])) {
// ...
if (fork() == 0) {
// ...
execv(...);
}
// ... deal with &
goto next_line;
}
// handle "normal" case
}
if (fork() == 0) {
// ...
execv(...);
}
// ... also deal with & here!
}
```


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

Artisanal Programming

- Factor your code!
  - A; D | B; D | C; D = (A | B | C); D
- Format as you write!
  - don't run the formatter only just before handing it in
  - your code should always be well formatted

- If you have a tough time understanding your code, you'll have a tougher time debugging it and TAs will have an even tougher time helping you

Copyright © 2023 Thomas W. Doeppner. All rights reserved.

It’s Your Code

- Be proud of it!
  - it not only works; it shows skillful artisanship
- It’s not enough to merely work
  - others have to understand it
    - (not to mention you ...)
  - you (and others) have to maintain it
    - shell 2 is coming soon!

Copyright © 2023 Thomas W. Doeppner. All rights reserved.

CS 33

Signals Part 1

Copyright © 2023 Thomas W. Doeppner. All rights reserved.

An Interlude Between Shells

- Shell 1
  - it can run programs
  - it can redirect I/O
- Signals
  - a mechanism for coping with exceptions and external events
  - the mechanism needed for shell 2
- Shell 2
  - it can control running programs

Copyright © 2023 Thomas W. Doeppner. All rights reserved.

Whoops ...

```
$ SometimesUsefulProgram xyz
Are you sure you want to proceed? Y
Are you really sure? Y
Reformatting of your disk will begin
in 3 seconds.
Everything you own will be deleted.
There's little you can do about it.
Too bad …
```

 Copyright © 2023 Thomas W. Doeppner. All rights reserved.

A Gentler Approach

- Signals
  - get a process's attention
    - » send it a signal
  - process must either deal with it or be terminated
    - » in some cases, the latter is the only option

Copyright © 2023 Thomas W. Doeppner. All rights reserved.

Stepping Back …

- What are we trying to do?
  - interrupt the execution of a program
    - » cleanly terminate it

      or
    - » cleanly change its course
  - not for the faint of heart
    - » it's difficult
    - » it gets complicated
    - » (not done in Windows)

Copyright © 2023 Thomas W. Doeppner. All rights reserved.

Signals

- Generated (by OS) in response to
  - exceptions (e.g., arithmetic errors, addressing problems)
    - synchronous signals
  - external events (e.g., timer expiration, certain keystrokes, actions of other processes)
    - asynchronous signals
- Effect on process:
  - termination (possibly producing a core dump)
  - invocation of a function that has been set up to be a signal handler
  - suspension of execution
  - resumption of execution

Copyright © 2023 Thomas W. Doeppner. All rights reserved.

Signal Types

SIGABRT	*abort* called	term, core
SIGALRM	alarm clock	term
SIGCHLD	death of a child	ignore
SIGCONT	continue after stop	cont
SIGFPE	erroneous arithmetic operation	term, core
SIGHUP	hangup on controlling terminal	term
SIGILL	illegal instruction	term, core
SIGINT	interrupt from keyboard	term
SIGKILL	kill	forced term
SIGPIPE	write on pipe with no one to read	term
SIGQUIT	quit	term, core
SIGSEGV	invalid memory reference	term, core
SIGSTOP	stop process	forced stop
SIGTERM	software termination signal	term
SIGTSTP	stop signal from keyboard	stop
SIGTTIN	background read attempted	stop
SIGTTOU	background write attempted	stop
SIGUSR1	application-defined signal 1	stop
SIGUSR2	application-defined signal 2	stop

Copyright © 2023 Thomas W. Doeppner. All rights reserved.

Sending a Signal

- `int kill(pid_t pid, int sig)`
  - send signal *sig* to process *pid*
- **Also**
  - *kill* shell command
  - type ctrl-c
    - » sends signal 2 (SIGINT) to current process
  - type ctrl-\
    - » sends signal 3 (SIGQUIT) to current process
  - type ctrl-z
    - » sends signal 20 (SIGTSTP) to current process
  - do something bad
    - » bad address, bad arithmetic, etc.

Copyright © 2023 Thomas W. Doeppner. All rights reserved.

Handling Signals

```
#include <signal.h>

typedef void (*sighandler_t)(int);
sighandler_t signal(int signo,
    sighandler_t handler);

sighandler_t OldHandler;

OldHandler = signal(SIGINT, NewHandler);
```


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

Special Handlers

- **SIG_IGN**
  - **ignore the signal**
  - `signal(SIGINT, SIG_IGN);`
- **SIG_DFL**
  - **use the default handler**
    - » **usually terminates the process**
  - `signal(SIGINT, SIG_DFL);`

Copyright © 2023 Thomas W. Doeppner. All rights reserved.

Example

```
void sigloop() {
  while(1)
    ;
}

int main() {
  void handler(int);
  signal(SIGINT, handler);
  sigloop();
  return 1;
}
void handler(int signo) {
  printf("I received signal %d. "
     "Whoopee!!\n", signo);
}
```


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

Digression: Core Dumps

- Core dumps
  - files (called "core") that hold the contents of a process's address space after termination by a signal
  - they're large and rarely used, so they're often disabled by default
  - use the ulimit command in bash to enable them

```
ulimit –c unlimited
```

  - use gdb to examine the process (post-mortem debugging)

```
gdb sig core
```


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

sigaction

```
int sigaction(int sig, const struct sigaction *new,
              struct sigaction *old);
struct sigaction {
   void (*sa_handler)(int);
   void (*sa_sigaction)(int, siginfo_t *, void *);
   sigset_t sa_mask;
   int sa_flags;
};

int main() {
   struct sigaction act; void myhandler(int);
   sigemptyset(&act.sa_mask); // zeroes the mask
   act.sa_flags = 0;
   act.sa_handler = myhandler;
   sigaction(SIGINT, &act, NULL);
   …
}
```


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

Example

```
int main() {
  void handler(int);
  struct sigaction act;
  act.sa_handler = handler;
  sigemptyset(&act.sa_mask);
  act.sa_flags = 0;
  sigaction(SIGINT, &act, 0);

  while(1)
    ;
  return 1;
}

void handler(int signo) {
  printf("I received signal %d. "
     "Whoopee!!\n", signo);
}
```

 Copyright © 2023 Thomas W. Doeppner. All rights reserved.

Quiz 1

```
int main() {
  void handler(int);
  struct sigaction act;
  act.sa_handler = hand
  sigemptyset(&act.sa_m
  act.sa_flags = 0;
  sigaction(SIGINT, &ac

  while(1)
    ;
  return 1;
}

void handler(int signo) {
  printf("I received signal %d. "
     "Whoopee!!\n", signo);
}
```

You run the example program, then quickly type ctrl-C. What is the most likely explanation if the program then terminates?

- **a) this "can't happen"; thus there's a problem with the system**
- **b) you're really quick or the system is really slow (or both)**
- **c) what we've told you so far isn't quite correct**

Copyright © 2023 Thomas W. Doeppner. All rights reserved.

Waiting for a Signal …

```
signal(SIGALRM, RespondToSignal);

…

struct timeval waitperiod = {0, 1000};
      /* seconds, microseconds */
struct timeval interval = {0, 0};
struct itimerval timerval;
timerval.it_value = waitperiod;
timerval.it_interval = interval;

setitimer(ITIMER_REAL, &timerval, 0);
      /* SIGALRM sent in ~one millisecond */
pause();   /* wait for it */
printf("success!\n");
```

 Copyright © 2023 Thomas W. Doeppner. All rights reserved.

Quiz 2

This program is guaranteed to print “success!”.

a) no

b) yes

```
signal(SIGALRM, RespondToSignal);

…

struct timeval waitperiod = {0, 1000};
      /* seconds, microseconds */
struct timeval interval = {0, 0};
struct itimerval timerval;
timerval.it_value = waitperiod;
timerval.it_interval = interval;

setitimer(ITIMER_REAL, &timerval, 0);
      /* SIGALRM sent in ~one millisecond */
pause();  /* wait for it */
printf("success!\n");
```


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

Masking Signals

```
setitimer(ITIMER_REAL, &timerval, 0);
      /* SIGALRM sent in ~one millisecond */
```

No signals here, please!

```
pause();  /* wait for it */
```


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

Masking Signals

mask SIGALRM

```
setitimer(ITIMER_REAL, &timerval, 0);
      /* SIGALRM sent in ~one millisecond */
```

No signals here

unmask and wait for SIGALRM

 Copyright © 2023 Thomas W. Doeppner. All rights reserved.

Doing It Safely

```
sigset_t set, oldset;
sigemptyset(&set);
sigaddset(&set, SIGALRM);
sigprocmask(SIG_BLOCK, &set, &oldset);
      /* SIGALRM now masked */
…
setitimer(ITIMER_REAL, &timerval, 0);
      /* SIGALRM sent in ~one millisecond */

sigsuspend(&oldset);    /* unmask sig and wait */
/* SIGALRM masked again */

sigprocmask(SIG_SETMASK, &oldset, (sigset_t *)0);
      /* SIGALRM unmasked */
printf("success!\n");
```

 Copyright © 2023 Thomas W. Doeppner. All rights reserved.

Signal Sets

- **To clear a set:**

```
int sigemptyset(sigset_t *set);
```

- **To add or remove a signal from the set:**

```
int sigaddset(sigset_t *set, int signo);
int sigdelset(sigset_t *set, int signo);
```

- **Example: to refer to both SIGHUP and SIGINT:**

```
sigset_t set;

sigemptyset(&set);
sigaddset(&set, SIGHUP);
sigaddset(&set, SIGINT);
```


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

Masking (Blocking) Signals

```
#include <signal.h>
int sigprocmask(int how, const sigset_t *set,
      sigset_t *old);
```

- used to examine or change the signal mask of the calling process
  - » *how* is one of three commands:
    - SIG_BLOCK
      - the new signal mask is the union of the current signal mask and set
    - SIG_UNBLOCK
      - the new signal mask is the intersection of the current signal mask and the complement of set
    - SIG_SETMASK
      - the new signal mask is set

Copyright © 2023 Thomas W. Doeppner. All rights reserved.

Signal Handlers and Masking

- **What if a signal occurs while a previous instance is being handled?**
  - **inconvenient …**
- **Signals are masked while being handled**
  - **may mask other signals as well:**

```
struct sigaction act; void myhandler(int);
sigemptyset(&act.sa_mask); // zeroes the mask
sigaddset(&act.sa_mask, SIGQUIT);
   // also mask SIGQUIT
act.sa_flags = 0;
act.sa_handler = myhandler;
sigaction(SIGINT, &act, NULL);
```

 Copyright © 2023 Thomas W. Doeppner. All rights reserved.

Timed Out!

```
int TimedInput( ) {
   signal(SIGALRM, timeout);
   …
   alarm(30);    /* send SIGALRM in 30 seconds */
   GetInput();   /* possible long wait for input */
   alarm(0);     /* cancel SIGALRM request */
   HandleInput();
   return(0);
nogood:
   return(1);
}

void timeout( ) {
   goto nogood;   /* not legal but straightforward */
}
```


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

Doing It Legally (but Weirdly)

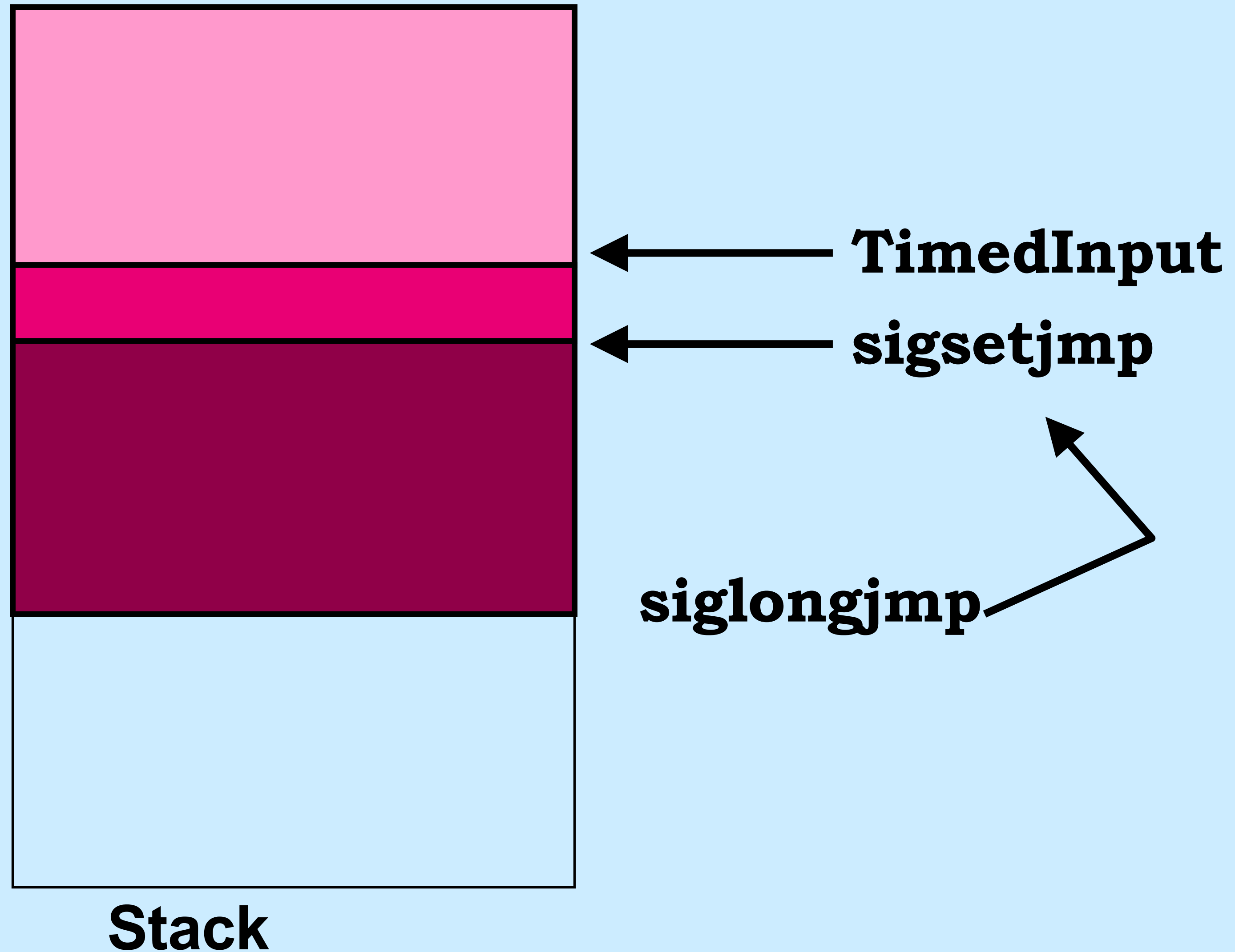
```
sigjmp_buf context;

int TimedInput( ) {
   signal(SIGALRM, timeout);
   if (sigsetjmp(context, 1) == 0) {
      alarm(30);  // cause SIGALRM in 30 seconds
      GetInput(); // possible long wait for input
      alarm(0);   // cancel SIGALRM request
      HandleInput();
      return 0;
   } else
      return 1;
}

void timeout() {
   siglongjmp(context, 1);    /* legal but weird */
}
```


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

sigsetjmp/siglongjmp



 Copyright © 2023 Thomas W. Doeppner. All rights reserved.

Exceptions

- **Other languages support exception handling**

```
try {
  something_a_bit_risky();
} catch(ArithmeticException e) {
  deal_with_it(e);
}
```

- **Can we do something like this in C?**

Copyright © 2023 Thomas W. Doeppner. All rights reserved.

Exception Handling in C

```
void Exception(int sig) {
  THROW(sig)
}

int computation(int a) {
  return a/(a-a);
}
```

```
int main() {
  signal(SIGFPE, Exception);
  signal(SIGSEGV, Exception);
  TRY {
    computation(1);
  } CATCH(SIGFPE) {
    fprintf(stderr,
      "SIGFPE\n");
  } CATCH(SIGSEGV) {
    fprintf(stderr,
      "SIGSEGV\n");
  } END

  return 0;
}
```


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

Exception Handling in C

```
#define TRY \
 { \
   int excp; \
   if ((excp = \
     sigsetjmp(ctx, 1)) == 0)

#define CATCH(a_excp) \
   else if (excp == a_excp)

#define END }

#define THROW(excp) \
  siglongjmp(ctx, excp);
```

 Copyright © 2023 Thomas W. Doeppner. All rights reserved.

Exception Handling in C

```
sigjmp_buf ctx;

int main() {
  ...
  {
    int excp;
    if ((excp = sigsetjmp(ctx, 1)) == 0) {
      computation(1);
    } else if (excp == SIGFPE) {
      fprintf(stderr, "SIGFPE\n");
    } else if (excp == SIGSEGV) {
      fprintf(stderr, "SIGFPE\n");
    }
  }
  return 0;
}
```

```
void exception(int sig) {
  siglongjmp(ctx, sig);
}
```

THROW (`siglongjmp(ctx, sig);`)

TRY (`if ((excp = sigsetjmp(ctx, 1)) == 0) {`)

CATCH (`} else if (excp == SIGFPE) {`)

CATCH (`} else if (excp == SIGSEGV) {`)

END

Copyright © 2023 Thomas W. Doeppner. All rights reserved.