CS:APP2e Web Aside ECF:SAFETY: Async-signal-safety*

Randal E. Bryant
David R. O'Hallaron

June 5, 2012

Notice

The material in this document is supplementary material to the book Computer Systems, A Programmer's Perspective, Second Edition*, by Randal E. Bryant and David R. O'Hallaron, published by Prentice-Hall and copyrighted 2011. In this document, all references beginning with "CS:APP2e " are to this book. More information about the book is available at* `csapp.cs.cmu.edu`.

1 The Problem

The examples in Figures CS:APP2e-8.31 – CS:APP2e-8.33 contain a bug that introduces a potential deadlock, where the program waits for an event that will never occur.[1] The problem exists because (1) the main routine calls a function that is not "safe" (in this case `printf`) in a code section where it can be interrupted by the receipt of a SIGCHLD signal, and (2) the SIGCHLD handler calls the same unsafe `printf` function.

The `printf` function acquires a console lock, calls the `write` system call, and then releases the console lock. If the `printf` function in the main routine is interrupted by the receipt of a SIGCHLD signal after it acquires the console lock and before it releases the lock, then the process will deadlock when the `printf` function in the SIGCHLD handler tries to acquire the same console lock.

To understand how to avoid such problems, we first need to introduce the idea of async-signal-safe functions.

[1] Deadlocks are discussed in Section CS:APP2e-12.7.5.

2 Async-signal-safe Functions

The general rule is that you should only call functions that are *async-signal-safe* from within signal handlers. A function is said to be async-signal-safe if it is either reentrant or non-interruptible by signals.[2] The functions listed in Figure 1 are guaranteed by the Posix standard to be async-signal-safe, and thus can be safely called from within signal handlers. Notice that `printf` is not included in this list, while `write` is.

In general, if the receipt of a signal interrupts the execution of an arbitrary unsafe function, and the resulting signal handling code then calls that unsafe function, then the behavior of the program is said to be *undefined*. Depending on exactly when the unsafe function call is interrupted, the program might run correctly, produce incorrect results, or even deadlock.

3 Solution Approaches

There are a couple of of ways to fix the bug in Figures CS:APP2e-8.31 – CS:APP2e-8.33. One approach is to use `sigprocmask` to block SIGCHLD around all calls to `printf` that could be interrupted by the receipt of a SIGCHLD signal. While safe, this approach has the undesirable effect of delaying the receipt of SIGCHLD across wide swaths of the program's execution, which defeats the purpose of signals as low-overhead notification mechanism. Further, in general it could be quite cumbersome to modify all invocations of popular functions like `printf`.

A simpler and more efficient approach is to replace calls to `printf` in the SIGCHLD handler

```
printf("Handler reaped child %d\n", (int)pid);
```

with calls to `snprintf` and `write`

```
snprintf(buf, MAXBUF, "Handler reaped child %d\n", (int)pid);
write(1, buf, strlen(buf));
```

The `write` function is async-signal-safe, and the `snprintf` implementation is (highly likely) reentrant, and thus async-signal-safe.

Acknowledgments

Thanks to Prof. Godmar Back, Virginia Tech, for identifying the bug and suggesting the different solution approaches.

[2] Reentrant functions are discussed in Section CS:APP2e-12.7.2.

_Exit	fpathconf	read	sigset
_exit	fstat	readlink	sigsuspend
abort	fsync	recv	sockatmark
accept	ftruncate	recvfrom	socket
access	getegid	recvmsg	socketpair
aio_error	geteuid	rename	stat
aio_return	getgid	rmdir	symlink
aio_suspend	getgroups	select	sysconf
alarm	getpeername	sem_post	tcdrain
bind	getpgrp	send	tcflow
cfgetispeed	getpid	sendmsg	tcflush
cfgetospeed	getppid	sendto	tcgetattr
cfsetispeed	getsockname	setgid	tcgetpgrp
cfsetospeed	getsockopt	setpgid	tcsendbreak
chdir	getuid	setsid	tcsetattr
chmod	kill	setsockopt	tcsetpgrp
chown	link	setuid	time
clock_gettime	listen	shutdown	timer_getoverrun
close	lseek	sigaction	timer_gettime
connect	lstat	sigaddset	timer_settime
creat	mkdir	sigdelset	times
dup	mkfifo	sigemptyset	umask
dup2	open	sigfillset	uname
execle	pathconf	sigismember	unlink
execve	pause	sleep	utime
fchmod	pipe	signal	wait
fchown	poll	sigpause	waitpid
fcntl	posix_trace_event	sigpending	write
fdatasync	pselect	sigprocmask	
fork	raise	sigqueue	

Figure 1: **Async-signal-safe Functions.**