Asynchronous Signal-Safe Functions

According to Section 7.14.1.1 of the C Rationale [ISO/IEC 2003]:

When a signal occurs, the normal flow of control of a program is interrupted. If a signal occurs that is being trapped by a signal handler, that handler is invoked. When it is finished, execution continues at the point at which the signal occurred. This arrangement can cause problems if the signal handler invokes a library function that was being executed at the time of the signal.

Similarly, Section 7.14.1, paragraph 5 of C99 [ISO/IEC 9899:1999] states that if the signal occurs other than as the result of calling the abort or raise function, the behavior is undefined if the signal handler calls any function in the standard library other than the `abort()` function, the `_Exit()` function, or the `signal()` function with the first argument equal to the signal number corresponding to the signal that caused the invocation of the handler.

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