Extended IO (Cont'd)

```
Terminal (and general device) control
SYNOPSIS
       #include <termios.h>
       ioctl(fildes, request, *arg);
       int fildes;
       int request;
       struct termios *arg;

       ioctl(fildes, request, arg);
       int fildes;
       int request;
       int arg;
   where:
       fildes    A valid, active file descriptor
       request   A control command
       arg       Either a pointer to a struct
                 termios, an int value or unused
                 depending on request
   returns:  value or -1
DESCRIPTION:
       The ioctl() call uses the following struct:
struct termios{
        tcflag_t   c_iflag;      /* input modes */
        tcflag_t   c_oflag;      /* output modes */
        tcflag_t   c_cflag;      /* control modes */
        tcflag_t   c_lflag;      /* local modes */
        cc_t       c_cc[NCCS];   /* control chars */
       };
```

Extended IO (Cont'd)

ioctl() (cont'd)

The ioctl() call uses the following requests:

TCGETS	The argument is a pointer to a termios structure. The current terminal parameters are fetched and stored into that structure.
TCSETS	The argument is a pointer to a termios structure. The current terminal parameters are set from the values stored in that structure. The change is immediate.
TCSETSW	The argument is a pointer to a termios structure. The current terminal parameters are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted. This form should be used when changing parameters that affect output.
TCSETSF	The argument is a pointer to a termios structure. The current terminal parameters are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted; all characters queued for input are discarded and then the change occurs.
TCSBRK	The argument is an int value. Wait for the output to drain. If the argument is 0, then send a break (zero valued bits for 0.25 seconds).
TCXONC	Start/stop control. The argument is an int value. If the argument is 0, suspend output; if 1, restart suspended output; if 2, suspend input; if 3, restart suspended input.
TCFLSH	The argument is an int value. If the argument is 0, flush the input queue; if 1, flush the output queue; if 2, flush both the input and output queues.

Extended IO (Cont'd)

ioctl() (cont'd)

The struct termio flags include:

- the c_iflag field describes the basic terminal input control:

IGNBRK	0000001	Ignore break condition.
BRKINT	0000002	Signal interrupt on break.
IGNPAR	0000004	Ignore characters with parity errors.
PARMRK	0000010	Mark parity errors.
INPCK	0000020	Enable input parity check.
ISTRIP	0000040	Strip character.
INLCR	0000100	Map NL to CR on input.
IGNCR	0000200	Ignore CR.
ICRNL	0000400	Map CR to NL on input.
IUCLC	0001000	Map upper-case to lower-case on input.
IXON	0002000	Enable start/stop output control.
IXANY	0004000	Enable any character to restart output.
IXOFF	0010000	Enable start/stop input control.
IMAXBEL	0020000	Echo BEL on input line too long.

- the c_oflag field specifies the system treatment of output:

OPOST	0000001	Postprocess output.
OLCUC	0000002	Map lower case to upper on output.
ONLCR	0000004	Map NL to CR-NL on output.
OCRNL	0000010	Map CR to NL on output.
ONOCR	0000020	No CR output at column 0.
ONLRET	0000040	NL performs CR function.
OFILL	0000100	Use fill characters for delay.
OFDEL	0000200	Fill is DEL, else NUL.

Extended IO (Cont'd)

ioctl() (cont'd)

The struct termio flags include:

- the c_cflag field describes the hardware control of the terminal:

CBAUD	0000017	Baud rate:
B0	0000000	Hang up
B50	0000001	50 baud
B75	0000002	75 baud
B110	0000003	110 baud
B134	0000004	134.5 baud
B150	0000005	150 baud
B200	0000006	200 baud
B300	0000007	300 baud
B600	0000010	600 baud
B1200	0000011	1200 baud
B1800	0000012	1800 baud
B2400	0000013	2400 baud
B4800	0000014	4800 baud
B9600	0000015	9600 baud
B19200	0000016	19200 baud
B38400	0000017	38400 baud
CSIZE	0000060	Character size:
CS5	0	5 bits
CS6	0000020	6 bits
CS7	0000040	7 bits
CS8	0000060	8 bits
CSTOPB	0000100	Send two stop bits, else one.
CREAD	0000200	Enable receiver.
PARENB	0000400	Parity enable.
PARODD	0001000	Odd parity, else even.
HUPCL	0002000	Hang up on last close.
CLOCAL	0004000	Local line, else dial-up.
CRTSCTS	0010000	Enable RTS/CTS flow control.
CIBAUD	03600000	Input baud rate, if different from output rate.

Extended IO (Cont'd)

ioctl() (cont'd)

The struct termio flags include:
- the c_lflag field of the argument structure is used by the line discipline to control terminal functions.

ISIG	0000001	Enable signals.
ICANON	0000002	Canonical input (erase and kill processing).
XCASE	0000004	Canonical upper/lower presentation.
ECHO	0000010	Enable echo.
ECHOE	0000020	Echo erase character as BS-SP-BS.
ECHOK	0000040	Echo NL after kill character.
ECHONL	0000100	Echo NL.
NOFLSH	0000200	Disable flush after interrupt or quit.
TOSTOP	0000400	Send SIGTTOU for background output.
ECHOCTL	0001000	Echo control characters as ^char, delete as ^?.
ECHOPRT	0002000	Echo erase character as character erased.
ECHOKE	0004000	BS-SP-BS erase entire line on line kill.
FLUSHO	0040000	Output is being flushed.
PENDIN	0100000	Retype pending input at next read or input character.

Extended IO (Cont'd)

ioctl() (cont'd)

The c_cc[NCCS] character array includes:

0	VINTR	DEL		
1	VQUIT	FS		
2	VERSE	#		
3	VKILL	@		
4	VEOF	EOT	VMIN	^D == '\004'
5	VEOL	NUL	VTIME	== '\000'
6	VEOL2	NUL		
7	VSWTCH	NUL		
8	VSTRT	DC1		
9	VSTOP	DC3		
10	VSUSP	SUB		
11	VDSUSP	EM		
12	VREPRINT	DC2		
13	VDISCRD	SI		
14	VWERSE	ETB		
15	VLNEXT	SYN		
16-19		reserved		

Extended IO (Cont'd)

ioctl() (cont'd)

Special characters

Certain characters have special functions on input. These functions and their default character values are summarized as follows:

INTR — (Rubout or ASCII DEL) generates a SIGINT signal. SIGINT is sent to all foreground processes associated with the controlling terminal. Normally, each such process is forced to terminate, but arrangements may be made either to ignore the signal or to receive a trap to an agreed upon location. [See signal(5)].

QUIT — (CTRL-| or ASCII FS) generates a SIGQUIT signal. Its treatment is identical to the interrupt signal except that, unless a receiving process has made other arrangements, it will not only be terminated but a core image file (called core) will be created in the current working directory.

ERASE — (#) erases the preceding character. It does not erase beyond the start of a line, as delimited by a NL, EOF, EOL, or EOL2 character.

WERASE — (CTRL-W or ASCII ETX) erases the preceding ''word''. It does not erase beyond the start of a line, as delimited by a NL, EOF, EOL, or EOL2 character.

KILL — (@) deletes the entire line, as delimited by a NL, EOF, EOL, or EOL2 character.

REPRINT — (CTRL-R or ASCII DC2) reprints all characters, preceded by a newline, that have not been read.

Extended IO (Cont'd)

ioctl() (cont'd)

Special characters

EOF — (CTRL-D or ASCII EOT) may be used to generate an end-of-file from a terminal. When received, all the characters waiting to be read are immediately passed to the program, without waiting for a newline, and the EOF is discarded. Thus, if no characters are waiting (i.e., the EOF occurred at the beginning of a line) zero characters are passed back, which is the standard end-of-file indication. Unless escaped, the EOF character is not echoed. Because EOT is the default EOF character, this prevents terminals that respond to EOT from hanging up.

NL — (ASCII LF) is the normal line delimiter. It cannot be changed or escaped.

EOL — (ASCII NULL) is an additional line delimiter, like NL. It is not normally used.

EOL2 — is another additional line delimiter.

SWTCH — (CTRL-Z or ASCII EM) is used only when shl layers is invoked.

SUSP — (CTRL-Z or ASCII SUB) generates a SIGTSTP signal. SIGTSTP stops all processes in the foreground process group for that terminal.

DSUSP — (CTRL-Y or ASCII EM) It generates a SIGTSTP signal as SUSP does, but the signal is sent when a process in the foreground process group attempts to read the DSUSP character, rather than when it is typed.

STOP — (CTRL-S or ASCII DC3) can be used to suspend output temporarily. It is useful with CRT terminals to prevent output from disappearing before it can be read. While output is suspended, STOP characters are ignored and not read.

Extended IO (Cont'd)

ioctl() (cont'd)

Special characters

START — (CTRL-Q or ASCII DC1) is used to resume output. Output has been suspended by a STOP character. While output is not suspended, START characters are ignored and not read.

DISCARD — (CTRL-O or ASCII SI) causes subsequent output to be discarded. Output is discarded until another DISCARD character is typed, more input arrives, or the condition is cleared by a program.

LNEXT — (CTRL-V or ASCII SYN) causes the special meaning of the next character to be ignored. This works for all the special characters mentioned above. It allows characters to be input that would otherwise be interpreted by the system (e.g. KILL, QUIT).

The character values for INTR, QUIT, ERASE, WERASE, KILL, REPRINT, EOF, EOL, EOL2, SWTCH, SUSP, DSUSP, STOP, START, DISCARD, and LNEXT may be changed to suit individual tastes. If the value of a special control character is _POSIX_VDISABLE (0), the function of that special control character is disabled. The ERASE, KILL, and EOF characters may be escaped by a preceding \ character, in which case no special function is done. Any of the special characters may be preceded by the LNEXT character, in which case no special function is done.

Extended IO (Cont'd)

ioctl() (cont'd)

EXAMPLE:

```
   int oldchflags, newchflags;
   struct termios oldt, newt;

   if((oldchflag = fcntl(0, F_GETFL, 0)) == -1){
            perror("fcntl F_GETFL failed");
            exit(1);
   }

   newchflags = oldchflag;
   newchflags |= O_NONBLOCK;
   if((x = (fcntl(0, F_SETFL, newchflags)) == -1){
            perror("fcntl F_SETFL failed");
            exit(1);
   }

   if(ioctl(0, TCGETS, &oldt) == -1){
            perror("ioctl TCGETS failed");
            exit(1);
   }

   newt = oldt;
   newt.c_iflag &= ~(INLCR | ICRNL | IUCLC | ISTRIP |
                     IXON | BRKINT);
   newt.c_oflag &= ~OPOST;
   newt.c_lflag &= ~(ICANON | ISIG | ECHO);
   newt.c_cc[4]  =  5; /* VMIN  */
   newt.c_cc[5]  =  2; /* VTIME */
   if(ioctl(0, TCSETSW, &newt) == -1){
            perror("ioctl TCSETSW failed");
            exit(1);
   }
```

Extended IO (Cont'd)

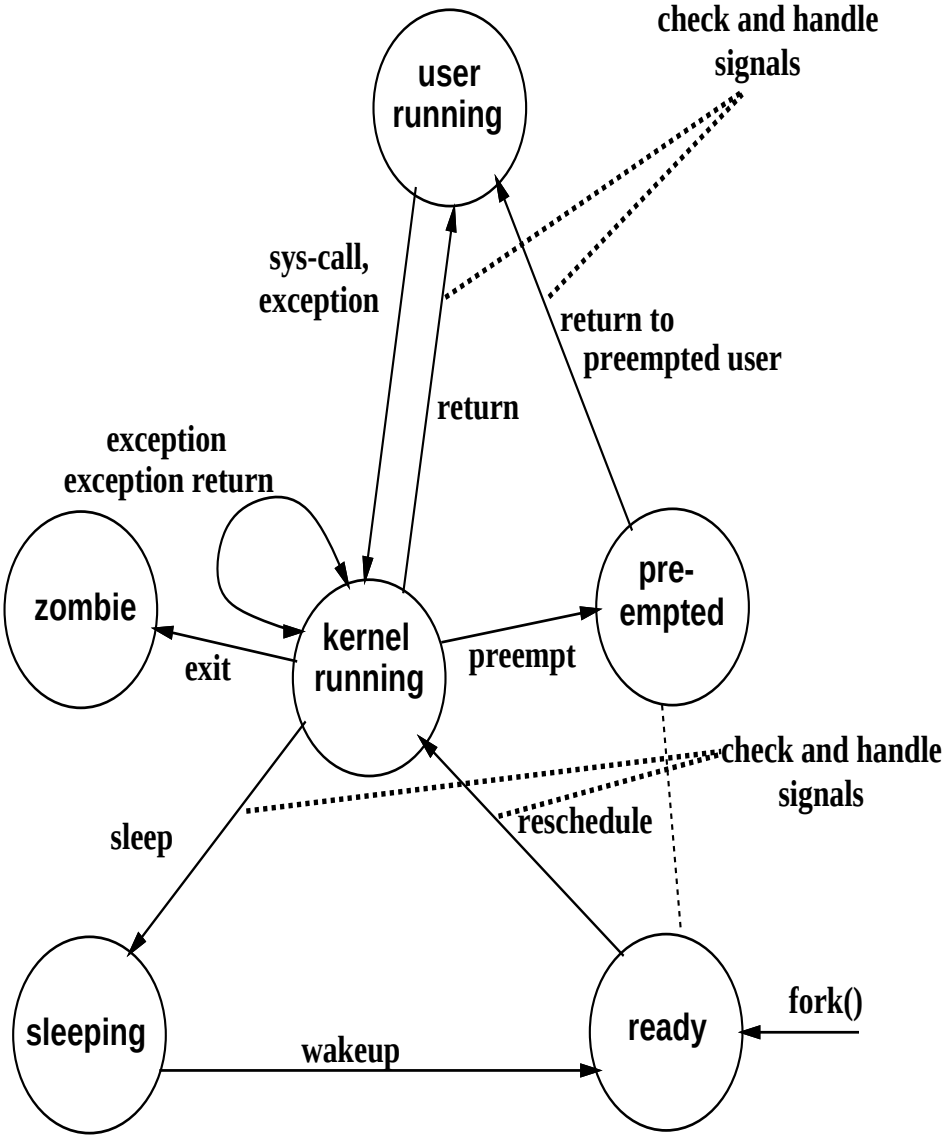
ioctl() (cont'd)

A set of library routines exist to provide a more object oriented interface to terminal control and is documented under termios(3C):

SYNOPSIS

```
#include <termios.h>
int tcgetattr(int fildes, struct termios *termios_p);
int tcsetattr(int fildes, int optional_actions,
     const struct termios *termios_p);
int tcsendbreak(int fildes, int duration);
int tcdrain(int fildes);
int tcflush(int fildes, int queue_selector);
int tcflow(int fildes, int action);
speed_t cfgetospeed(struct termios *termios_p);
int cfsetospeed(const struct termios *termios_p,
     speed_t speed);
speed_t cfgetispeed(struct termios *termios_p);
int cfsetispeed(const struct termios *termios_p,
     speed_t speed);

#include <sys/types.h>
#include <termios.h>
pid_t tcgetpgrp(int fildes);
int tcsetpgrp(int fildes, pid_t pgid);
pid_t tcgetsid(int fildes);
```

UNIX Process States and Transitions



Process Management

Changing a process's data, stack and program segments (loading a new program)

SYNOPSIS

```
#include <unistd.h>

 int execl (const char *path,
            const char *arg0,...,const char *argn,
            (char *)0);

 int execv (const char *path,
            char *const *argv);

 int execle (const char *path,
             const char *arg0,...,const char *argn,
             (char *)0,
             const char *envp[]);

 int execve (const char *path,
             char *const *argv,
             char *const *envp);

 int execlp (const char *file,
             const char *arg0,...,const char *argn,
             (char *)0);

int execvp (const char *file,
             char *const *argv);
```

Process Management

exec..() (cont'd)

where:

path A pointer to a pathname that identifies the new process file.

file A pointer to the new process file. If file does not contain a slash character, the path prefix for this file is obtained by a search of the directories passed in the PATH environment variable [see environ(5)]. The environment is supplied typically by the shell [see sh(1)]. If the new process file is not an executable object file, execlp and execvp use the contents of that file as standard input to sh(1).

arg (0 through n) Pointers to null-terminated character strings. These strings constitute the argument list available to the new process image. Minimally, arg0 must be present. It will become the name of the process, as displayed by the ps command. Conventionally, arg0 points to a string that is the same as path (or the last component of path). The list of argument strings is terminated by a (char *)0 argument.

argv An array of character pointers to null-terminated strings. These strings constitute the argument list available to the new process image. By convention, argv must have at least one member, and it should point to a string that is the same as path (or its last component). argv is terminated by a null pointer.

envp An array of character pointers to null-terminated strings. These strings constitute the environment for the new process image. envp is terminated by a null pointer. For execl, execv, execvp, and execlp, the C run-time start-off routine places a pointer to the environment of the calling process in the global object extern char **environ, and it is used to pass the environment of the calling process to the new process.

returns: does not return on success or returns -1

Process Management (Cont'd)

exec..() (cont'd)

EXAMPLE:

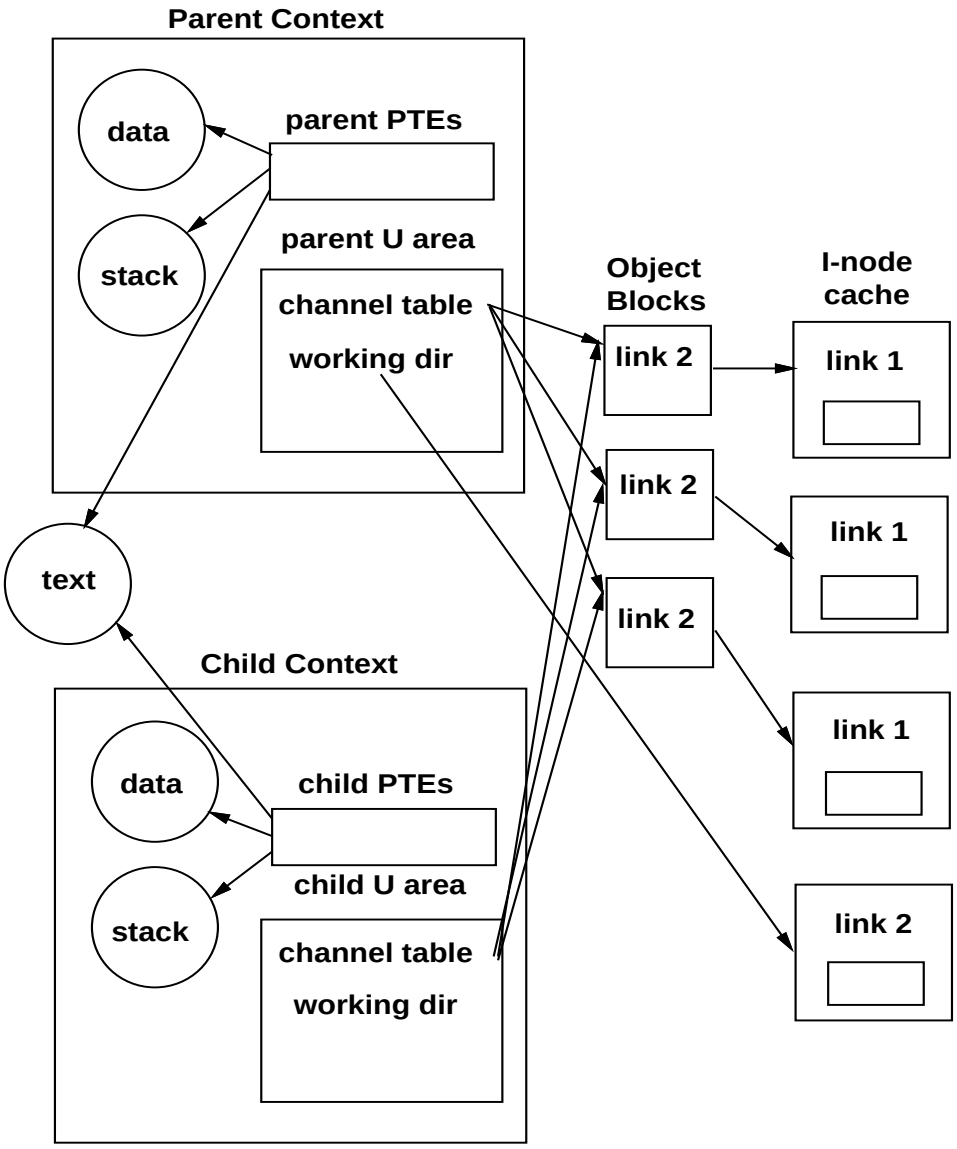
The exec family provides the only way to execute a program from its entry point (first imperative statement in main())

```
/*  print with this program   */
/*    and the echo program    */

main(){
 printf(" The quick brown fox jumped over ");
 execl("/bin/echo", "echo", "the", "lazy",
        dogs", NULL);
 perror("execl failed");
}
```

Process Management (Cont'd)

```
Creating a new process
SYNOPSIS
       #include <sys/types.h>
       pid_t   fork ()

   returns:  PID of child to parent, 0 to
             child or -1

             Called once, returns twice if
             successful

EXAMPLE:

  main(){
    int   pid;

    printf("Start of test \n");
    pid = fork();
   if (pid == 0)
      printf("Child checking in \n");
   else
      printf("Parent checking in \n");
  }
```

UNIX fork() Operation



Process Management (Cont'd)

fork() (cont'd)

EXAMPLE:

```
main(argc, argv)
  int argc;
  char *argv[];
{
  int pid;

  printf("Hi, I'm PID %d \n", getpid());
  switch(pid = fork()){
   case -1:   printf("Can't create new \
                      process");
              exit(1);

   case  0:   printf("New child \n");
              execvp(argv[0], argv);
              printf("Can't execute \n");
              exit(1);

   default:   printf("New child is %d \n",pid);
              if (wait(NULL) == -1){
                 perror("wait failed");
                 exit(2);
              }
  }  /*switch  */
}  /*program */
```

Process Management (Cont'd)

```
Exit with a return status
Wait for termination of child
SYNOPSIS
       #include <unistd.h>
       void _exit(int status);
   where:
       status  An integer indicating the status
               to be returned
   returns:  does not return

SYNOPSIS
       #include <sys/types.h>
       #include <sys/wait.h>
       pid_t wait(stat_loc)
       int *stat_loc;

       pid_t waitpid (pid, stat_loc, options)
       pid_t pid;
       int *stat_loc;
       int options;
   where:
       pid       A process identifier
       stat_loc  A location for returning a
                 process status
       options   0 or a positive integer
 (i.e. WNOHANG for async op)
   return: returns child pid, 0 or -1
```

Process Management (Cont'd)

wait(), waitpid() (cont'd)

The status return intrger is described in wait.h as shown below

```
union wait      {
 int     w_status;     /* used in syscall */
    /* Terminated process status */
 struct {
      unsigned short w_Termsig:7;  /* term signal */
      unsigned short w_Coredump:1; /* core dump ind */
      unsigned short w_Retcode:8;  /* exit code   */
  } w_T;
};
```

```
#define w_termsig       w_T.w_Termsig
#define w_coredump      w_T.w_Coredump
#define w_retcode       w_T.w_Retcode
```

Process Management (Cont'd)

wait(), waitpid() (cont'd)

DESCRIPTION

The wait() and waitpid() functions allow the calling process to obtain status information pertaining to one of its child processes. Various options permit status information to be obtained for child processes that have terminated or stopped. If status information is available for two or more child processes, the order in which their status is reported is unspecified.

The wait() function shall suspend execution of the calling process until status information for one of its terminated child processes is available, or until delivery of a signal whose action is either to execute a signal-catching function or to terminate the process. If status information is available prior to the call to wait(), return shall be immediate.

The waitpid() function shall behave identically to the wait() function, if the pid argument has a value of -1 and the options argument has a value of zero. Otherwise, its behavior shall be modified by the values of the pid and options arguments.

The pid argument specifies a set of child processes for which status is requested. The waitpid() function shall only return the status of a child process from this set.

(1) If pid is equal to -1, status is requested for any child process. In this respect, waitpid() is then equivalent to wait().

(2) If pid is greater than zero, it specifies the process ID of a single child process for which status is requested.

(3) If pid is equal to zero, status is requested for any child process whose process group ID is equal to that of the calling process.

(4) If pid is less than -1, status is requested for any child process whose process group ID is equal to the absolute value of pid.

Process Management (Cont'd)

wait(), waitpid() (cont'd)

Option Flags

The options argument is constructed from the bitwise inclusive OR of zero or more of the following flags, defined in the header <sys/wait.h>:

WUNTRACED If the implementation supports job control, the status of any child processes specified by pid that are stopped, and whose status has not yet been reported since they stopped, shall also be reported to the requesting process.

WCONTINUED
Also report the status of any continued child process specified by pid whose status has not been reported since it continued.

WNOHANG The waitpid() function shall not suspend execution of the calling process if status is not immediately available for one of the child processes specified by pid.

WNOWAIT Keep the process whose status is returned in stat_loc in a waitable state. The process may be waited for again with identical results.