

Processes

Process ID (pid)

Synopsis

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

pid_t getpid(void) – returns the pid of the currently running process.

pid_t getppid(void) – returns the pid of the parent of the currently running process.

POSIX

PID Example

```
/* Example 2.1 */  
#include <stdio.h>  
#include <unistd.h>  
  
void main(void)  
{  
    printf("Process ID: %ld\n", (long)getpid());  
    printf("Parent process ID: %ld\n", (long)getppid());  
    return 0;  
}
```

User/Group ID

- System managers assign a unique integral user ID and an integral group ID to each user when creating the user's account
- The system uses the user and group IDs to retrieve privileges from the system database for that user
- Each process is identified with a particular user called the owner
- Each user has a unique ID (uid)
- Effective User ID (euid) can change during execution and determine access permissions to files
- System manager has a UID of 0

POSIX

User/Group ID Synopsis

SYNOPSIS

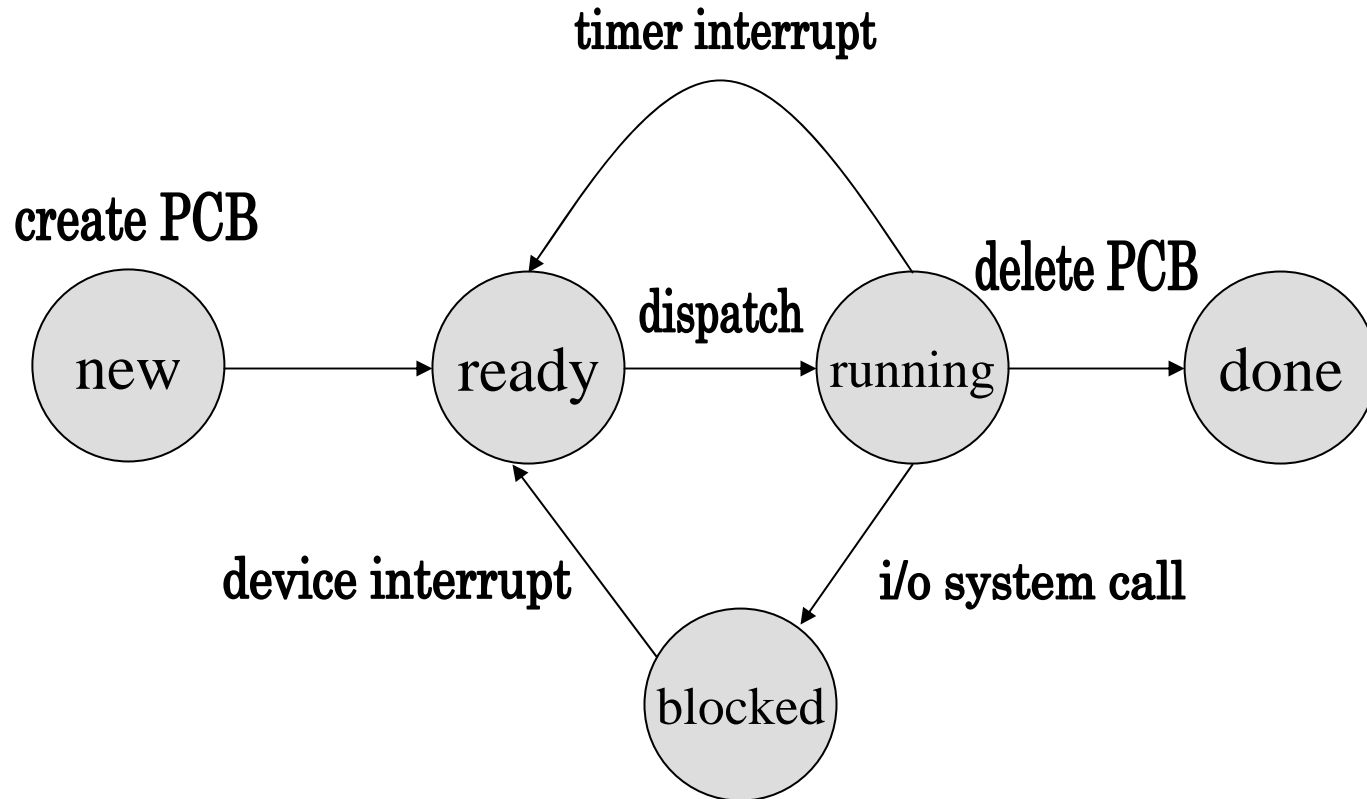
```
#include <unistd.h>  
gid_t getegid(void);  
uid_t geteuid(void);  
gid_t getgid(void);  
uid_t getuid(void);
```

POSIX

User/Group ID Example

```
#include <stdio.h>
#include <unistd.h>
int main (void) {
    printf("My real user ID is %5ld\n", (long) getuid());
    printf("My effective user ID is %5ld\n", (long) geteuid());
    printf("My real group ID is %5ld\n", (long) getgid());
    printf("My effective group ID is %5ld\n", (long) getuid());
    return 0;
}
```

Process States



ps

Displays information about processes. The `-a` option displays information for processes associated with terminals

SYNOPSIS

```
ps [-aA] [-G grouplist] [-o format]...  
    [-p proclist] [-t termlist] [-U userlist]
```

POSIX Shells and Utilities

-

ps Fields

header	Meaning
F	Flags associated with the process
S	The process state
UID	The user ID of the process owner
PID	The process ID
PPID	The parent process ID
C	The processor utilization used for scheduling
PRI	The priority of the process
NI	The nice value
ADDR	The memory address of the process
SZ	The size of the process image
WCHAN	The address of the event if the process is sleeping
TTY	The controlling terminal
TIME	Cumulative execution time
COMMAND	Command name

fork System Call

Creates child process by copying parent's memory image

SYNOPSIS

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

```
pid_t fork(void)
```

POSIX

fork return values

- Returns 0 to child
- Returns child PID to parent
- Returns -1 on error

Fork Attributes

Child inherits:

- Parent's memory image
- Most of the parent's attributes including environment and privilege.
- Some of parent's resources such as open files.

Child does not inherit:

- Parent pid.
- Parent time clock (child clock is set to 0).

fork Example

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
void main(void)
{
    pid_t childpid;

    childpid = fork()
    if(childpid == -1 {
        perror("failed to fork");
        return 1; }
    if(childpid == 0) {
        fprintf(stderr, "I am the child, ID = %ld\n", (long)getpid());
        /* child code goes here */
    } else if (childpid > 0) {
        fprintf(stderr, "I am the parent, ID = %ld\n", (long)getpid());
        /* parent code goes here */
    }
}
```

fork (Chain)

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
void main(void)
{
    int i;
    int n;
    pid_t childpid;
    n = 4;
    for (i = 1; i < n; ++i)
        if (childpid = fork())
            break;
    fprintf(stderr, "This is process %ld with parent %ld\n",
            (long)getpid(), (long)getppid());
    sleep(1);
}
```

fork (Fan)

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

void main(void)
{
    int i;
    int n;
    pid_t childpid;

    n = 4;
    for (i = 1; i < n; ++i)
        if ((childpid = fork()) <= 0)
            break;
    fprintf(stderr, "This is process %ld with parent %ld\n",
            (long)getpid(), (long)getppid());
    sleep(1);
}
```

fork (Tree)

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

void main(void)
{
    int i;
    int n;
    pid_t childpid;

    for (i = 1; i < 4; ++i)
        if ((childpid = fork()) == -1)
            break;
    fprintf(stderr, "This is process %ld with parent %ld\n",
            (long)getpid(), (long)getppid());
    sleep(1);
}
```

wait Function

SYNOPSIS

```
#include <sys/wait.h>
```

```
pid_t wait (int *stat_loc);
```

```
pid_t waitpid(pid_t pid, int *stat_loc, int options);
```

POSIX

wait_errno

errno	cause
ECHILD	caller has no unwaited-for children (wait) or process or process group specified by pid does not exist (waitpid), or process group specified by pid does not have a member that is a child of caller (waitpid)
EINTR	function was interrupted by a signal
EINVAL	options parameter of waitpid was invalid

pid_t wait(int *stat_loc)

- Causes caller to pause until a child terminates, or stops until the caller receives a signal.
- If wait returns because a child terminates, the return value (of type pid_t) is positive and is the pid of that child.
- Otherwise wait returns -1 and sets errno.
- stat_loc is a pointer to an integer variable.
- If caller passes something other than NULL, wait stores the return status (terminate status?) of the child.
- POSIX specifies the following macros for testing the return status: WIFEXITED, WEXITSTATUS, WIFSIGNALED, WTERMSIG, WIFSTOPPED, and WSTOPSIG.
- Child returns status by calling exit, _exit, or return.

Chain with wait

```
#include ...
void main(void) {
    int i;
    int n;
    pid_t childpid;
    int status;
    pid_t waitreturn;
    n = 4;
    for (i = 1; i < n; ++i)
        if (childpid = fork())
            break;
    while(childpid != (waitreturn = wait(&status)))
        if ((waitreturn == -1) && (errno != EINTR))
            break;
    fprintf(stderr, "I am process %ld, my parent is %ld\n",
            (long)getpid(), (long)getppid()); }
}
```

r_wait

Restarts the wait function if it is interrupted by a signal

Status Values

SYNOPSIS

```
#include <sys/wait.h>
```

```
WIFEXITED (int stat_val)
```

```
WIFEXITSTATUS (int stat_val)
```

```
WIFSIGNALED (int stat_val)
```

```
WTERMSIG (int stat_val)
```

```
WIFSTOPPED (int stat_val)
```

```
WIFSTOPSIG (int stat_val)
```

Use of Status Values

```
wait(&status);
```

```
if(WIFSTOPPED(status)) printf("Child  
stopped due to a signal");
```

waitpid (pid,status,options)

If the pid parameter is:

- greater than 0, wait for child with pid number
- equal to 0, wait for any child in the same process group as the caller
- equal to -1, wait for any child
- less than -1, wait for any child in the process group specified by the absolute value of pid

waitpid Status Parameter

- The second waitpid parameter is used the same way as the wait parameter (i.e., NULL or &status)
- The second parameter can be tested by status (WIF...) operators

waitpid Options Parameter

- The options parameter is the bitwise or of one or more flags
- It is set to 0 if there are no options
- `WNOHANG` causes `waitpid` to return even no child has terminated
- `WUNTRACED` causes `waitpid` to report the status of unreported child processes that have been stopped

waitpid Return Values

- waitpid returns:
 - a value of -1 if an error occurs
 - a value of 0 if there are possible unwaited-for children that have not terminated (applies when WNOHANG option is set)
 - a value of the terminating child pid if a child terminates

waitpid examples

waitreturn = waitpid (apid, &status,0);

wait for the specific child apid

waitreturn = waitpid (-1, NULL, NOHANG);

wait for any child, but do not delay if a child is currently not available

waitreturn = waitpid(0, NULL,0)

wait for any child in the process group of the calling process

waitreturn = waitpid(-4828,NULL,0)

wait for any child in the process group 4828

waitpid – WNOHANG

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <errno.h>
void main(void) {
    int status;
    pid_t childpid;
    pid_t waitreturnpid;
    childpid = fork();
    if (childpid == 0) {
        fprintf(stderr, "Child %ld will sleep for 5 seconds\n", (long)getpid());
        sleep(5);
        fprintf(stderr, "Child %ld will now exit\n", (long)getpid());
        exit(0); }
    sleep(8);
    fprintf(stderr, "Parent %ld will wait for any child\n", (long)getpid());
    while(waitreturnpid = waitpid(-1, &status, WNOHANG))
        if (!(waitreturnpid == -1) && (errno != EINTR))
            break;
    fprintf(stderr, "Parent will exit after receiving %ld from waitpid\n",
        waitreturnpid); }
```

exec1, execlp, execl

“l” – Passes command directly as a parameter in exec.

- **exec1** searches for command in fully qualified pathname passed as exec parameter or in current directory
- **execlp** uses PATH environment variable to find command
- **execl** uses environment passed as exec parameter to find command

execv, execvp,execve

“v” – Passes command as member of argument array (i.e., argv[] or makeargv[])

- **execv** searches for arg array command in fully qualified pathname passed in exec or in current directory
- **execvp** uses PATH environment variable to find arg array command
- **execve** uses environment passed as exec parameter to find arg array command

exec1 Example

```
/* Program 2.6 */
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
void main(void) {
    pid_t childpid;
    int status;
    if ((childpid = fork()) == -1) {
        perror("Error in the fork");
        exit(1);
    } else if (childpid == 0) {          /* child code */
        if (exec1("/usr/bin/ls", "ls", "-l", NULL) < 0) {
            perror("Exec of ls failed");
            exit(1); }
    } else if (childpid != wait(&status)) /* parent code */
        perror("A signal occurred before the child exited");
    exit(0); }
```

execvp Example

```
/* Program 2.7 */
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
void main(int argc, char *argv[]) {
    pid_t childpid, waitreturn;
    int status;
    if ((childpid = fork()) == -1) {
        perror("The fork failed");
        exit(1);
    } else if (childpid == 0) {          /* child code */
        if (execvp(argv[1], &argv[1]) < 0) {
            perror("The exec of command failed");
            exit(1); }
    } else                               /* parent code */
        while(childpid != (waitreturn = wait(&status)))
            if ((waitreturn == -1) && (errno != EINTR))
                break;
    exit(0); }
```

errno for exec

errno
E2BIG
EACCESEINVAL
ELOOP
ENAMEOOLONG
ENOENT
ENOEXEC
ENOTDIR

Use of makeargv

```
#include <sys/wait.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include "restart.h"
int makeargv(char *s, char *delimiters, char ***argvp);
void main(int argc, char *argv[]) {
    char **myargv;
    char delim[] = " \t";
    pid_t childpid;
    if (argc != 2) {
        fprintf(stderr, "Usage: %s string\n", argv[0]);
        return 1; }
    childpid = fork();
    if (childpid == -1) {
        perror("Failed to fork");
        return 1 }
    if (childpid == 0) {
        /* child code */
        if (makeargv(argv[1], delim, &myargv) == -1) {
            perror ("Child failed to constuct argument array"); }
        else {
            execvp(myargv[0], &myargv[0]);
            perror("Child failed to exec command"); }
        return 1; }
    /* parent code */
    if (childpid != r_wait(NULL)) {
        perror("Parent failed to wait");
        return 1; }
    return 0 }
```

Attributes Preserved by Calls to exec

Preserved Attribute	Relevant System Call
Process ID	getpid()
Parent process ID	getppid()
Process group ID	getpgid()
Session ID	getsid()
Real user ID	getuid()
Real group ID	getgid()
Supplementary group IDs	getgroups()
Time left on alarm signal	alarm()
Current working directory	getcwd()
Root directory	
File mode creation mask	unmask()
File size limit*	ulimit()
Process signal mask	sigprocmask()
Pending signals	sigpending()
Time elapsed	times() Continued on next slide

exec Attributes (Continued)

Preserved Attribute	Relevant System Call
resource limits*	getrlimit(), setrlimit()
controlling terminal*	open(), tcgetpgrp()
interval timers*	ualarm()
nice values*	nice()
semadj values*	semop()

An * indicates an attribute that is inherited in the POSIX:XSI Extension

Background Processes

- Child process becomes background process when it executes `setsid()`.
- Child that becomes background process never returns to parent.
- Background processes cannot be interrupted with `ctrl-c`.

Daemon

A daemon is a background process that runs indefinitely.

Examples:

- Solaris 2 pageout daemon
- Mailer daemon

Background Processes

```
#include <sys/wait.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include "restart.h"
int makeargv(char *s, char *delimiters, char ***argvp);
void main(int argc, char *argv[]) {
    char **myargv;
    char delim[] = " \t";
    pid_t childpid;
    if (argc != 2) {
        fprintf(stderr, "Usage: %s string\n", argv[0]);
        return 1; }
    childpid = fork();
    if (childpid == -1) {
        perror("The fork failed");
        return 1 }
    if (childpid == 0) { /* child becomes a background process */
        if (setsid() == -1)
            perror("Could not become a session leader");
        else if (makeargv(argv[1], delim, &myargv) == -1)
            fprintf(stderr, "Argument array could not be constructed\n");
        else {
            execvp(myargv[0], &myargv[0]);
            perror("Child failed to exec command"); }
        return 1; } /* child should never return */
    return 0 } /* parent exits */
```

Critical Sections

- Interleaved printing.
- Multi-process to shared memory where at least one process is writing.