

## fork() - create a child process

### SYNOPSIS

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

```
pid_t fork(void);
```

fork creates a child process that is duplicate of parent process and differs from the parent process only in its PID and PPID. File locks and pending signals are not inherited.

On success, the PID of the child process is returned in the parent's thread of execution, and a 0 is returned in the child's thread of execution. On failure, a -1 will be returned in the parent's context, no child process will be created, and errno will be set appropriately.

```
/* this is a socket program that uses fork() to create multi-client
server*/
```

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

```
main() {
    int sock, length;
    struct sockaddr_in server;
    int msgsock;
    char buf[1024];
    int rval;
    int i;

    sock = socket (AF_INET, SOCK_STREAM, 0);
    if (sock < 0) {
        perror("opening stream socket");
    }

    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = 8888;

    if (bind (sock, (struct sockaddr *)&server, sizeof server) < 0) {
        perror ("binding stream socket");
    }
    listen (sock, 5);
    while(1){
        msgsock = accept(sock, (struct sockaddr *)0,
                        (socklen_t *)0);

        if (msgsock == -1) {
            perror("accept");
        }
    }
}
```

```

    }
    if (fork() == 0) {
        if ((rval = read(msgsock, buf, 1024)) < 0){
            perror("reading socket");
        }else {
            printf("%s\n",buf);
            exit(0);
        }
        close (msgsock);
    }
}
}

```

**Signal Handling:** The purpose of the zombie state is to maintain information about the child for the parent to fetch at some time later. The information includes the process ID of the child, its termination status, and information on the resource utilization of the child such as CPU and memory. Whenever we fork() children we must wait for them to prevent them from becoming zombies. To do this we implement a signal handler to catch SIGCHLD and within the signal handler we call wait. The signal handler is added through the call

Signal (SIGCHLD, sig\_chld);



Where sig\_chld could be the following function:

```

void sig_chld (int signo)
{
    pid_t pid;
    int stat;

    pid = wait (&stat);
    printf("Child %d terminated\n",pid);
    return;
}

```

### **wait() and waitpid() functions.**

```
pid_t wait (int *statloc)
```

The **wait** function suspends execution of the current process until a child has exited, or until a signal is delivered whose action is to terminate the current process or to call a signal handling function. If a child has already exited by the time of the call (a so-called "zombie" process), the function returns immediately. Any system resources used by the child are freed.

pid\_t waitpid (pid\_t pid, int \*statloc, int options)

The **waitpid** function suspends execution of the current process until a child as specified by the *pid* argument has exited, or until a signal is delivered whose action is to terminate the current process or to call a signal handling function. If a child as requested by *pid* has already exited by the time of the call (a so-called "zombie" process), the function returns immediately. Any system resources used by the child are freed.

The value of *pid* can be one of:

< -1

which means to wait for any child process whose process group ID is equal to the absolute value of *pid*.

-1

which means to wait for any child process; this is the same behaviour which **wait** exhibits.

0

which means to wait for any child process whose process group ID is equal to that of the calling process.

> 0

which means to wait for the child whose process ID is equal to the value of *pid*.

## **exec**

The **exec** family of functions replaces the current process image with a new process image. The initial argument for these functions is the pathname of a file which is to be executed.

Practice Assignment: Modify the forkedserver so that it handles SIGCHLD signal.