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");
```

fork() - 创建一个子进程

概要

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

pid_t fork(void);
```

fork 创建一个与父进程完全相同的子进程，仅在 PID 和 PPID 上与父进程不同。文件锁和未处理的信号不会被继承。

成功时，在父进程的执行线程中返回子进程的 PID，在子进程的执行线程中返回 0。失败时，父进程的上下文中将返回 -1，不会创建子进程，且 errno 将被适当地设置。

```
/* 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);

Whre 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.

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

信号处理： 僵尸状态的目的是为父进程保留有关子进程的信息，以便父进程在稍后某个时间获取。这些信息包括子进程的进程 ID、其终止状态，以及子进程的资源使用情况（如 CPU 和内存）。每当我们通过 fork() 创建子进程时，必须等待它们，以防止它们变成僵尸进程。为此，我们实现一个信号处理器来捕获 SIGCHLD，并在信号处理器中调用 wait。该信号处理器通过以下调用添加

Signal (SIGCHLD, sig_chld);

其中 sig_chld 可以是以下函数：

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

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

wait() 和 waitpid() 函数。

pid_t wait (int *statloc)

wait 函数会暂停当前进程的执行，直到某个子进程退出，或直到收到一个信号（该信号的操作是终止当前进程或调用信号处理函数）。如果在调用时子进程已经退出（即所谓的“僵尸”进程），则该函数立即返回。子进程所使用的任何系统资源都将被释放。

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.

pid_t waitpid (pid_t pid, int *statloc, int options)

waitpid 函数会暂停当前进程的执行，直到由 pid 参数指定的子进程退出，或直到收到一个信号（该信号的操作是终止当前进程或调用信号处理函数）。如果由 pid 指定的子进程在调用时已经退出（即所谓的“僵尸”进程），则该函数立即返回。子进程所使用的任何系统资源都将被释放。

pid 的值可以是以下之一：

< -1
: 表示等待其进程组 ID 等于 pid 绝对值的任意子进程。

-1
: 表示等待任意子进程；这与 wait 所表现的行为相同。

0
: 表示等待其进程组 ID 与调用进程相同的任意子进程。

> 0
: 表示等待进程 ID 等于 pid 值的子进程。

exec

exec 函数族会用新的进程镜像替换当前的进程镜像。这些函数的第一个参数是要执行的文件的路径名。

练习任务：修改 forkedserver，使其能够处理 SIGCHLD 信号。