Linux C comunication beetween processes using signals and pipes

I have a program to write in C. It must create 3 processes which work all time. Those 3 processes receive signals(stop, pause, resume), when it receive one of those signals it sends other signal to the rest of processes and then writes to pipe(it must be pipe) what signal he receive. The others receive that signal and read the pipe and do what is said in pipe. I have something like that to write a pipe in signal handler:

```
void rcvkillsig(int sig){
  if(sig==SIGINT){
    int op;
    op = 1;
    close(pfd1[0]);
    close(pfd3[0]);
    write(pfd1[1], &op, 8);
    write(pfd3[1], &op, 8);
    close(pfd1[1]);
    close(pfd3[1]);
    kill(sndpid, SIGCONT);
    kill(rcvpid, SIGCONT);
    printf("End chk\n");
    kill(chkpid, SIGKILL);
  }
}
```

And something like that to read pipe:

```
void rcvinfsig(int sig){
  if(sig==SIGCONT){
    cflag=0;
    int op;
    close(pfd2[1]);
    read(pfd2[0], &op, 8);
    close(pfd2[0]);
    if(op==1){
        kill(chkpid, SIGKILL);
        printf("End chk\n");
    }
    else if(op==2){
        printf("Pause chk!\n");
        cpaused=1;
    }
    else if(op==3){
        printf("Resume chk!\n");
        cpaused=0;
    }
  }
}
```

Of course there are codes like that for every process, and received signal.

I use in this program pfd1[2], pfd2[2], pfd3[2] for each process and in main function creates pipes for them by pipe(...).

My problem is that, when the process receive first signal(for example pause) he write to a pipe, but when it receives second signal(for example resume) it don't write to pipe, and resumes only self.

Please help I need that program on Monday. The rest of code works and I don't have only that communication.

c linux signals communication pipe

asked Jun 15 '13 at 20:57
manveruPL
32 10

1 Answer

Based on posted functions, I think there are two main problems in your code.

1. You close file descriptors on each call of signal handler.

It's not good idea - close file descriptors each time after reading or writing . Unidirectional pipes don't work in such way. After creation of descriptors pair you should close first descriptor in one process and second in another process. This will create a pipe from first process to second (man pipe). In example from man page, they close descriptor right after write, because they don't need it any more.

2. Seems like you have only 3 pairs of file descriptors.

For connecting of 3 processes through unidirectional pipes (as shown on picture bellow), requires 6 pairs of descriptors:

```
                  fd 5,6
  +------------+           +------------+
  |            |<----------|            |
  | process #0 |           | process #1 |
  |            |---------->|            |
  +------------+           +------------+
       |   ^      fd 7,8       |   ^
       |   |                   |   |
fd 1,2 |   | fd 3,4    fd 9,10 |   | fd 11,12
       |   |                   |   |
       v   |                   |   |
  +------------+               |   |
  |            |<--------------+   |
  | process #2 |                   |
  |            |-------------------+
  +------------+
```

Example.

Here an example of code, that doing same thing. It's far away from good example (bad signals processing, `printf` in signals handler), but I hope that it will help a little.

```
int g_pfd[3][2][2];  // pairs of file desriptors.

volatile sig_atomic_t g_ignore_sighup = 0; // flag for ignoring SIGHUP

void ( * g_handlers[3] ) ( int );  // array of signal handlers

void sig_handler ( int signo, int id )
{
    if ( signo == SIGHUP )
    {
        if ( g_ignore_sighup )
        {
            g_ignore_sighup = 0;
            return;
        }

        printf ( "SIGHUP recvd, pid = %d\n", getpid () );

        int rd1 = ( id );
        int rd2 = ( id == 0 ? 2 : id - 1 );

        // choose, which process sent SIGHUP.

        fd_set rfds;
        FD_ZERO ( &rfds );
        FD_SET ( g_pfd[rd1][1][0], &rfds );
        FD_SET ( g_pfd[rd2][0][0], &rfds );

        int rv = select ( FD_SETSIZE, &rfds, NULL, NULL, NULL );

        if ( rv == -1 )
        {
            perror ( "select" );
            return;
        }
        else if ( rv == 0 )
        {
            return;
        }

        int fd = -1;
        if ( FD_ISSET ( g_pfd[rd1][1][0], &rfds ) ) fd = g_pfd[rd1][1][0];
        if ( FD_ISSET ( g_pfd[rd2][0][0], &rfds ) ) fd = g_pfd[rd2][0][0];

        int i;

        if ( read ( fd, &i, sizeof ( int ) ) == -1 )
        {
            perror ( "read" );
        }

        printf ( "recvd data through pipe = %d\n", i );
        return;
    }

    if ( signo == SIGINT )
    {
        int wr1 = ( id );
        int wr2 = ( id == 0 ? 2 : id - 1 );

        printf ( "SIGINT recvd, pid = %d\n", getpid () );
```

```c
        printf ( "write: %d to %d\n", getpid (), g_pfd[wr1][0][1] );
        printf ( "write: %d to %d\n", getpid (), g_pfd[wr2][1][1] );

        int pid = getpid ();
        if ( write ( g_pfd[wr1][0][1], &pid, sizeof ( int ) ) == -1 ||
             write ( g_pfd[wr2][1][1], &pid, sizeof ( int ) ) == -1 )
        {
            perror ( "write" );
        }

        g_ignore_sighup = 1;  // flag for ignorig own signal

        // send SIGHUP to parent and all its children.

        if ( kill ( 0, SIGHUP ) == -1 )
        {
            perror ( "kill" );
        }

        return;
    }
}

void sig_handler_0 ( int signo ) { sig_handler ( signo, 0 ); }
void sig_handler_1 ( int signo ) { sig_handler ( signo, 1 ); }
void sig_handler_2 ( int signo ) { sig_handler ( signo, 2 ); }

int create_process ( int *pid, int id )
{
    *pid = fork ();

    if ( *pid == -1 )
    {
        perror ( "fork" );
        return 1;
    }

    if ( *pid != 0 )  // parent
    {
        return 0;
    }

    // close appropriate descriptors

    int i1 = ( id );
    int i2 = ( id == 0 ? 2 : id - 1 );

    close ( g_pfd[i1][0][0] );
    close ( g_pfd[i2][0][1] );
    close ( g_pfd[i1][1][1] );
    close ( g_pfd[i2][1][0] );

    if ( signal ( SIGINT, g_handlers[id] ) == SIG_ERR ||
         signal ( SIGHUP, g_handlers[id] ) == SIG_ERR )
    {
        perror ( "signal" );
        return 1;
    }

    while ( 1 ) sleep ( 1 );
    exit  ( 0 );
}

int main ( int argc, char *argv [] )
{
    // fill array of signal handlers.

    g_handlers[0] = sig_handler_0;
    g_handlers[1] = sig_handler_1;
    g_handlers[2] = sig_handler_2;

    if ( signal ( SIGHUP, SIG_IGN ) == SIG_ERR )
    {
        perror ( "signal" );
        return 1;
    }

    int pid [3];
    int i, j;

    // create pairs of descriptors

    for ( i = 0; i < 3; i++ )
    {
        for ( j = 0; j < 2; j++ )
        {
            if ( pipe ( g_pfd[i][j] ) == -1 )
            {
                perror ( "pipe" );
                return 1;
            }
        }
    }

    if ( create_process ( &pid[0], 0 ) != 0 ||
         create_process ( &pid[1], 1 ) != 0 ||
         create_process ( &pid[2], 2 ) != 0 )
    {
        return 1;
    }

    sleep ( 1 );

    kill ( pid[0], SIGINT ); sleep ( 3 );
    kill ( pid[1], SIGINT ); sleep ( 3 );
    kill ( pid[2], SIGINT );
    wait ( NULL );

    for ( i = 0; i < 3; i++ )
    {
        for ( j = 0; j < 2; j++ )
        {
            close ( g_pfd[i][j][0] );
            close ( g_pfd[i][j][1] );
        }
    }

    return 0;
}
```

edited Jun 16 '13 at 4:09

answered Jun 16 '13 at 3:20

0xff
2,283 1 10 15

Thanks for that I hope it will work for me. – manveruPL Jun 16 '13 at 9:13