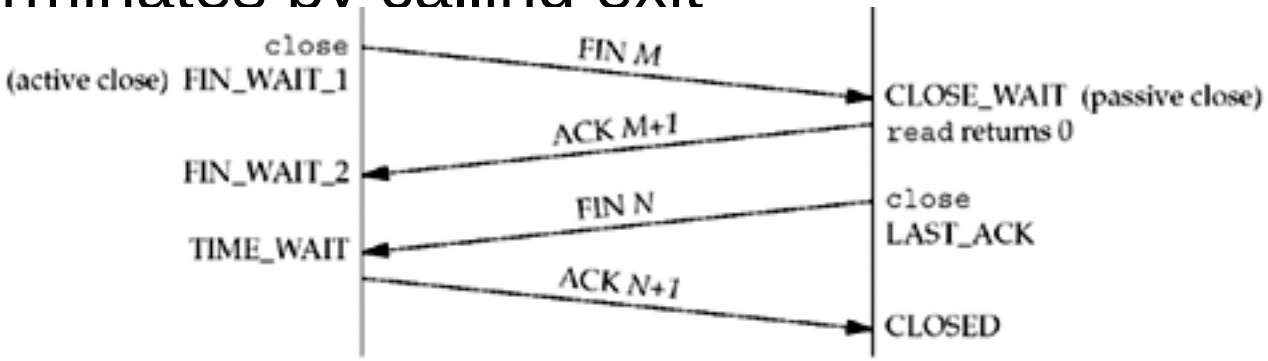
TCP Client/Server Example – part 2

Giảng viên: TS. Trần Hải Anh
Bộ môn Truyền Thông & Mạng máy tính
Khoa CNTT & TT

Intro

- In the previous practical work, after sending the EOF character => *fgets()* returns null pointer => the function *str_cli* returns => the client main function calls *exit()* => the client sends FIN to the server
- => *str_echo* function returns => server child terminates by calling exit



Issue

- Finally, the SIGCHLD signal is sent to the parent when the server child terminates.
- This occurs in this example, but we do not catch the signal in our code, and the default action of the signal is to be ignored. Thus, the child enters the zombie state.

```
linux % ps -t pts/6 -o pid,ppid,tty,stat,args,wchan
  PID  PPID TT       STAT COMMAND          WCHAN
22038 22036 pts/6    S    -bash            read_chan
17870 22038 pts/6    S    ./tcpserv01      wait_for_connect
19315 17870 pts/6    Z    [tcpserv01 <defu do_exit
```

POSIX Signal Handling

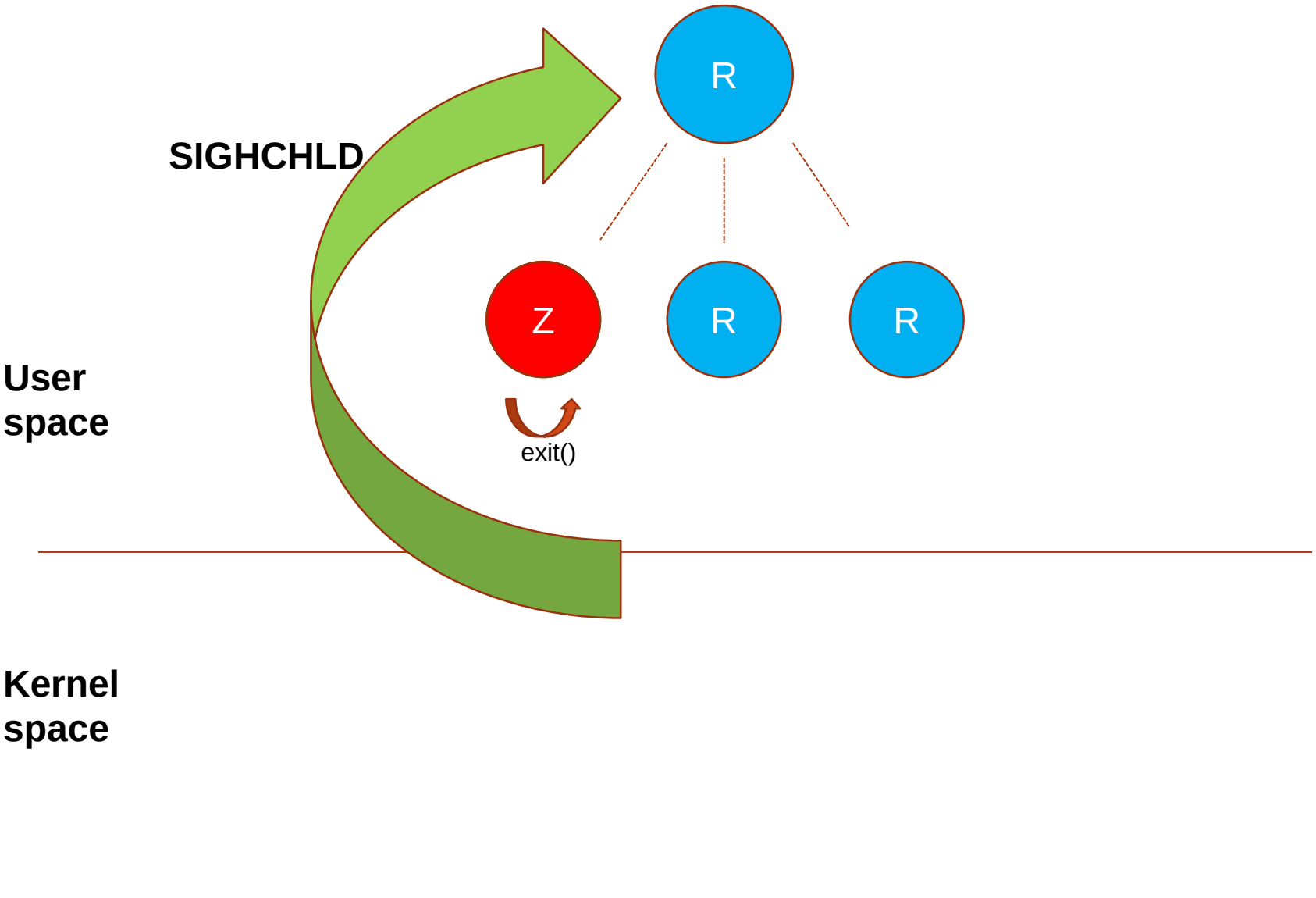
- A *signal* is a notification to a process that an event has occurred (*software interrupts*).
- *asynchronously*
- Signals can be sent
  - by one process to another process (or to itself)
  - by the kernel to a process
- Example: The SIGCHLD signal is sent by the kernel whenever a process terminates, to the parent of the terminating process.

POSIX Signal Handling

- Every signal has a *disposition* (the *action* associated with the signal). We set the disposition of a signal by calling the sigaction function.
- 3 choices for the disposition:
  - We can provide a function that is called whenever a specific signal occurs. This function is called a *signal handler* and this action is called *catching* a signal.
  - We can *ignore* a signal by setting its disposition to SIG_IGN.
  - We can set the *default* disposition for a signal by setting its disposition to SIG_DFL.

Handling SIGCHLD Signals

- The purpose of the zombie state is to maintain information about the child for the parent:
  - the process ID of the child
  - its termination status
  - information on the resource utilization of the child (CPU time, memory, etc.).
- If a process terminates, and that process has children in the zombie state, the parent process ID of all the zombie children is set to 1 (the *init* process).
- We do not want to leave zombies around:
  - They take up space in the kernel
  - We can run out of processes.
- Whenever we fork children, we must *wait* for them to prevent them from becoming zombies.
- Establish a signal handler


SIGHCHLD
R
Z
R
R
exit()
User space
Kernel space

Handling SIGCHLD Signals

- We establish the signal handler by adding the function call after the call to listen function:

```
Signal (SIGCHLD, sig_chld);
```

- We define the function sig_chld

tcpcliserv/sigchldwait.c

```
 1 #include     "unp.h"

 2 void
 3 sig_chld(int signo)
 4 {
 5     pid_t   pid;
 6     int     stat;

 7     pid = wait(&stat);
 8     printf("child %d terminated\", pid);
 9     return;
10 }
```

Handling SIGCHLD Signals

- Now, try to compile and run it!

```
solaris % tcpserv02 &
[2] 16939

solaris % tcpcli01 127.0.0.1
hi there
hi there
^D
child 16942 terminated
accept error: Interrupted system call
```

Handling SIGCHLD Signals

- To handle an interrupted accept, we change the call to accept:

```
for ( ; ; ) {
    clilen = sizeof (cliaddr);
    if ( (connfd = accept (listenfd, (SA *) &cliaddr, &clilen)) < 0) {
        if (errno == EINTR)
            continue;          /* back to for () */
        else
            err_sys ("accept error");
   }
```

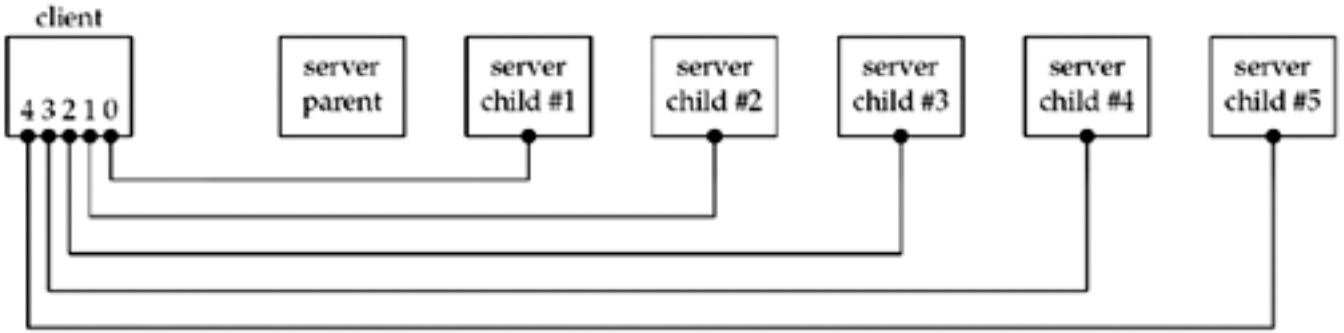
wait and *waitpid* functions

```
#include <sys/wait.h>
pid_t wait (int *statloc);
pid_t waitpid (pid_t pid, int *statloc,
int options);
```

- Both return: process ID if OK, 0 or–1 on error
- The waitpid gives more control:
  - which process to wait for (*pid* argument)
  - whether or not to block (*options* argument)

Difference between wait and waitpid

- **Client with five established connections to same concurrent server.**



tcpcliserv/tcpcli04.c

```
 1 #include     "unp.h"

 2 int
 3 main (int argc, char **argv)
 4 {
 5      int     i, sockfd[5];
 6      struct sockaddr_in servaddr;

 7      if (argc != 2)
 8          err_quit ("usage: tcpcli <IPaddress>";

 9      for (i = 0; i < 5; i++) {
10          sockfd[i] = Socket (AF_INET, SOCK_STREAM, 0);

11          bzero (&servaddr, sizeof (servaddr));
12          servaddr.sin_family = AF_INET;
13          servaddr.sin_port = htons (SERV_PORT);
14          Inet_pton (AF_INET, argv[1], &servaddr.sin_addr);

15          Connect (sockfd[i], (SA *) &servaddr, sizeof (servaddr));
16     }

17     str_cli (stdin, sockfd[0]);  /* do it all */

18     exit(0);
19 }
```

Problem!!!

```
linux % tcpserv03 &
[1] 20419
linux % tcpcli04 127.0.0.1
hello
hello
^D
child 20426 terminated
```

Only one child process is terminated

Why?

- Now, we execute the ps command:

```
PID TTY          TIME CMD
20419 pts/6    00:00:00 tcpserv03
20421 pts/6    00:00:00 tcpserv03 <defunct>
20422 pts/6    00:00:00 tcpserv03 <defunct>
20423 pts/6    00:00:00 tcpserv03 <defunct>
```

- Establishing a signal handler and calling *wait* from that handler are insufficient for preventing zombies
- The problem is that all five signals are generated before the signal handler is executed, and the signal handler is executed only one time because **Unix signals are normally not queued**.

Use *waitpid*

Final (correct) version of *sig_chld* function that calls *waitpid*

tcpcliserv/sigchldwaitpid.c

```
 1 #include    "unp.h"

 2 void
 3 sig_chld(int signo)
 4 {
 5     pid_t    pid;
 6     int      stat;

 7     while ( (pid = waitpid(-1, &stat, WNOHANG)) > 0)
 8         printf("child %d terminated\n", pid);
 9     return;
10 }
```

This tells waitpid not to block if there are running c
that have not yet terminated

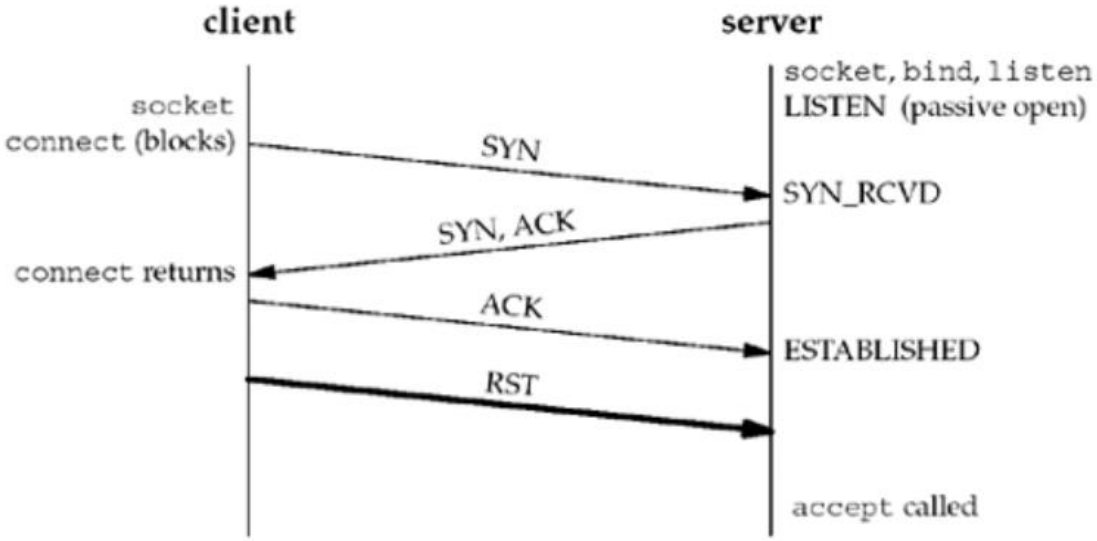
A value of -1 says to wait for the first of our children to te

Conclusion

- We must catch the SIGCHLD signal when forking child processes.
- We must handle interrupted system calls when we catch signals.
- A SIGCHLD handler must be coded correctly using waitpid to prevent any zombies from being left around.

Connection Abort before *accept* Returns

- Receiving an RST for an ESTABLISHED connection before *accept* is called.
- The aborted connection completely within the kernel
- That returns an error to the process as the return from *accept* (ECONNABORTED)



Some another issues

- Termination of Server Process
- Crashing of Server Host
- Crashing and Rebooting of Server Host
- Shutdown of Server Host

- **Practical Works**