

Writing Client/Server Programs in C Using Sockets (A Tutorial) Part II

Session 5959

Greg Granger
grgran@sas.com

SAS/C & C++ Support
SAS Institute
Cary, NC

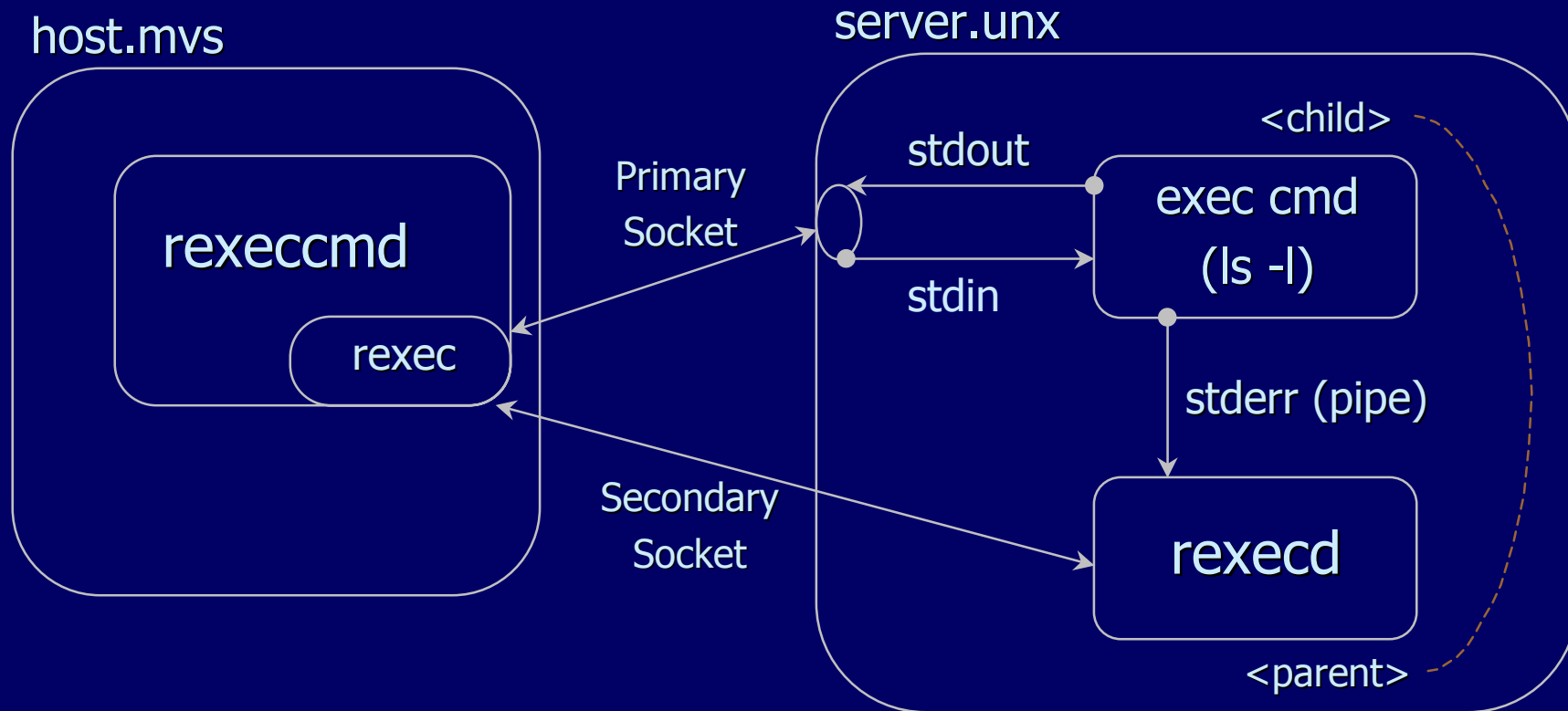
Remote Execution Sample Program

- * Remote Execution Connection Diagram
- * getservbyname() coding
- * rexec() function declaration
- * gethostbyname() and struct hostent
- * rexec() caller source code examination
- * System call return conventions
- * Socket address structures and conventions
- * recv() and send() conventions
- * File descriptor sets and select()

Remote Execution Connection Diagram

Client issues:

```
rexec server.unx mark mypass ls -l
```



REXEC Initial Connection

host.mvs



server.unx



Primary
Socket

client issues:

`rexeccmd.c`

[53] `getservbyname()`

[60] `rexec()`

`rexec.c`

[61] `gethostbyname()`

[70] `socket()`

[79] `connect()`

rexecd issues:

`socket()`

`bind()`

`listen()`

`select()`

`accept()`

getservbyname() Coding

getservbyname() returns NULL for failure or a pointer to this structure for success:

```
struct servent {
    char *s_name;      /* official name      */
    char **s_aliases; /* array of aliases */
    int  s_port;      /* well-known port  */
    char *s_proto; }; /* protocol to use (udp/tcp) */
```

On UNIX the data comes from the “/etc/services” file formatted like the following:

```
exec      512/tcp      # remote execution
login     513/tcp      # remote login
who       513/udp      whod # remote who and uptime
```

rexec() Function Declaration

```
int rexec(                /* Returns primary socket */
           char **host,    /* Pointer to hostname by */
                               /* reference */
           int port,       /* Port for rexecd server */
           char *username, /* Username on remote host */
           char *passwd,   /* Password on remote host */
           char *cmd,       /* Command to execute */
           int *ptr_secondary /* Place to store secondary */)
```

gethostbyname() Coding

Gethostbyname() returns NULL for failure or a pointer to this structure for success:

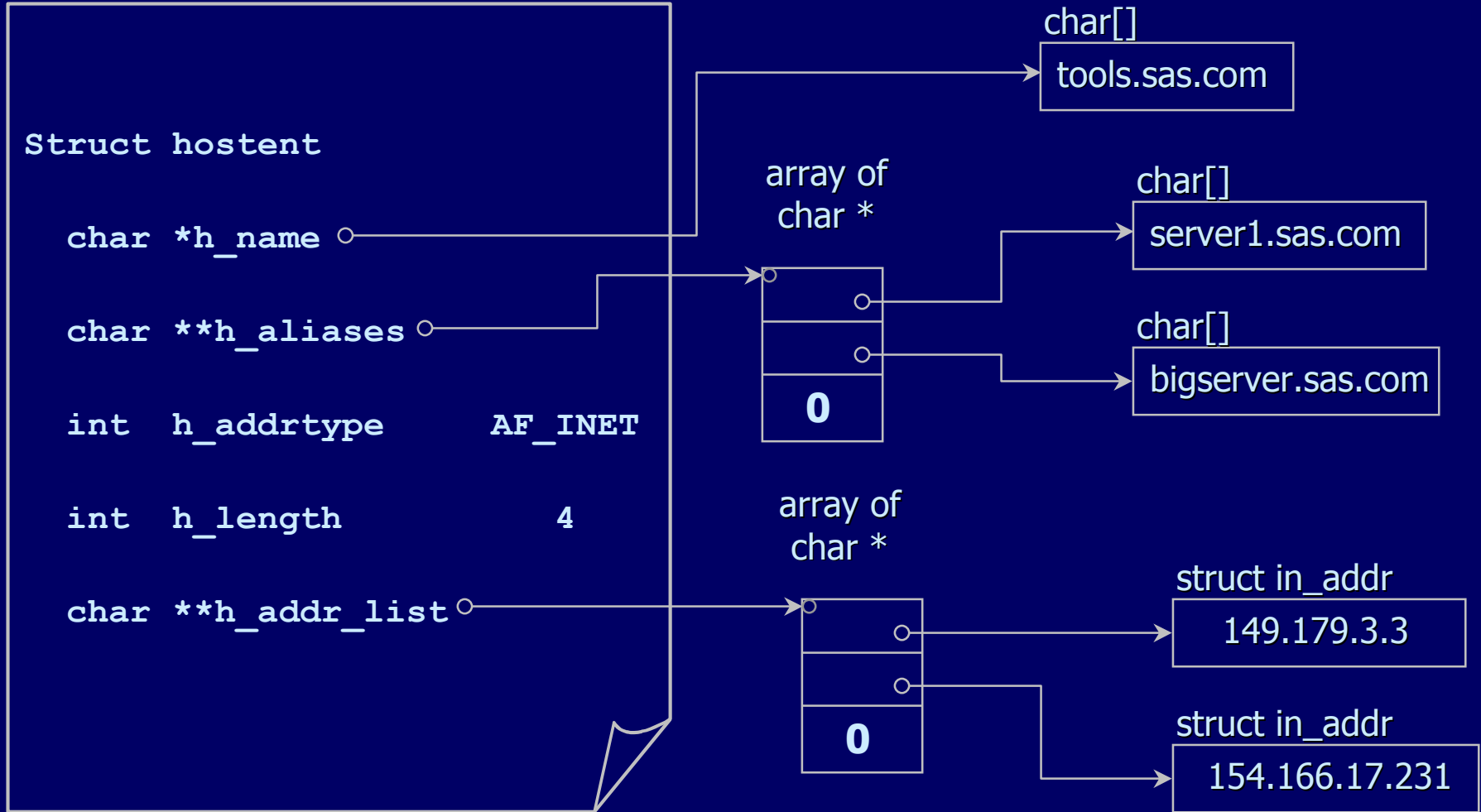
```
Struct hostent {
    char *h_name;           /* Official name */
    char **h_aliases;      /* array of aliases */
    int h_addrtype;        /* AF_INET for TCP/IP */
    int h_length;          /* sizeof(struct in_addr) for TCP/IP */
    char **h_addr_list;    /* points to array of struct in_addr */
#define h_addr h_addr_list[0]; /* Primary IP address */
```

Here is an example of printing the IP address:

```
struct hostent *he;
char * host = "ahost.unx.sas.com";
struct in_addr *ia;

he = gethostbyname(host);
ia = (struct in_addr *) (he->h_addr);
printf("IP address: %s\n",inet_ntoa(*ia));
```

"struct hostent" diagram



System Call Return Conventions

- * Most system calls have the same return conventions
- * An int value ≥ 0 indicates success; -1 indicates failure.
- * Failed calls set "errno" to indicate the cause of failure; perror() will print a message based on the "errno" value.
- * Here is the typical coding logic:

```
int s = socket(AF_INET, SOCK_STREAM, 0);
if (s < 0) {
    perror("socket() failed");
    exit(EXIT_FAILURE);
}
```

Socket Address Structures

Socket address formats vary. Each format is called an "addressing family". The generic format is:

```
struct sockaddr {
    u_short sa_family;    /* Addressing family */
    char sa_data[14]; }; /* varies */
```

AF_INET is the addressing family for TCP/IP. It's format is:

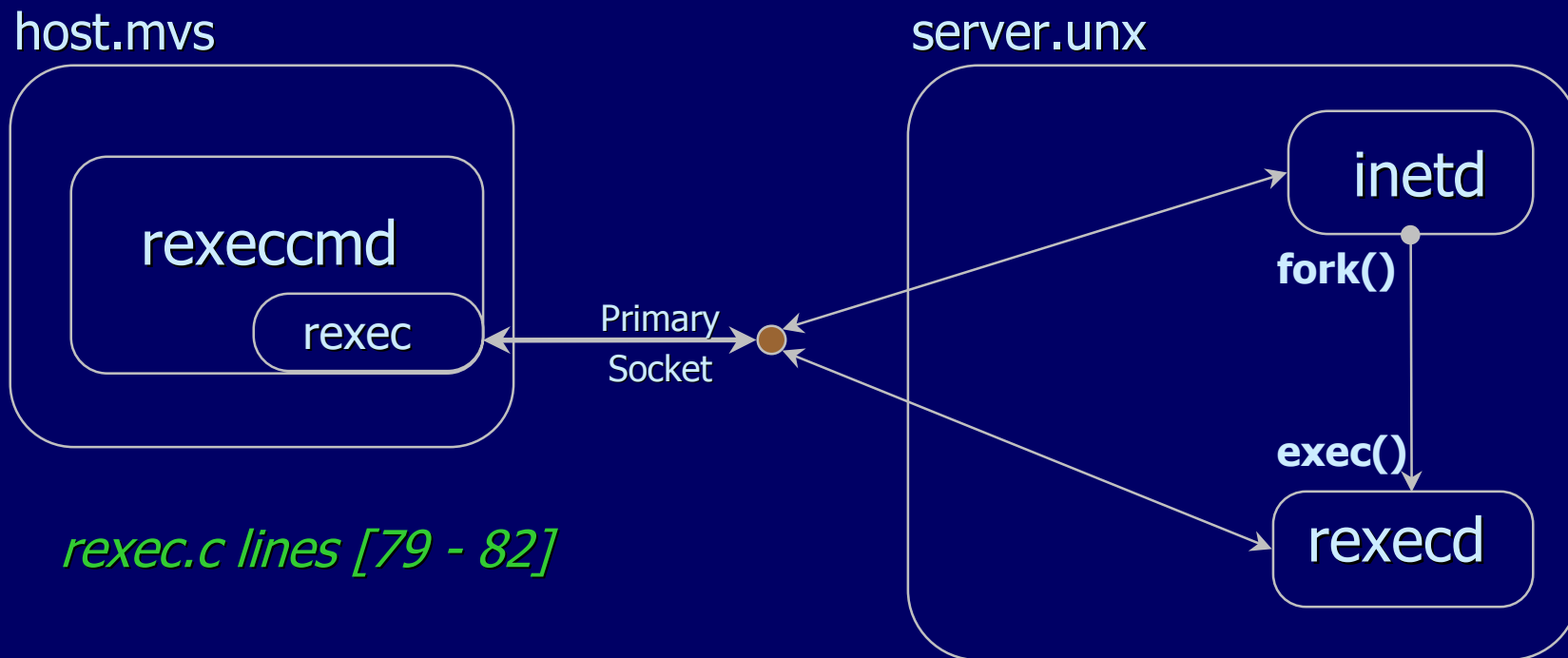
```
struct in_addr {
    u_long s_addr; }; /* net byte order */
```

```
struct sockaddr_in {
    short    sin_family;           /* AF_INET */
    u_short  sin_port;            /* TCP or UDP port */
    struct   in_addr sin_addr;    /* IP address */
    char     sin_zero[8]; };     /* varies */
```

Socket Address Conventions

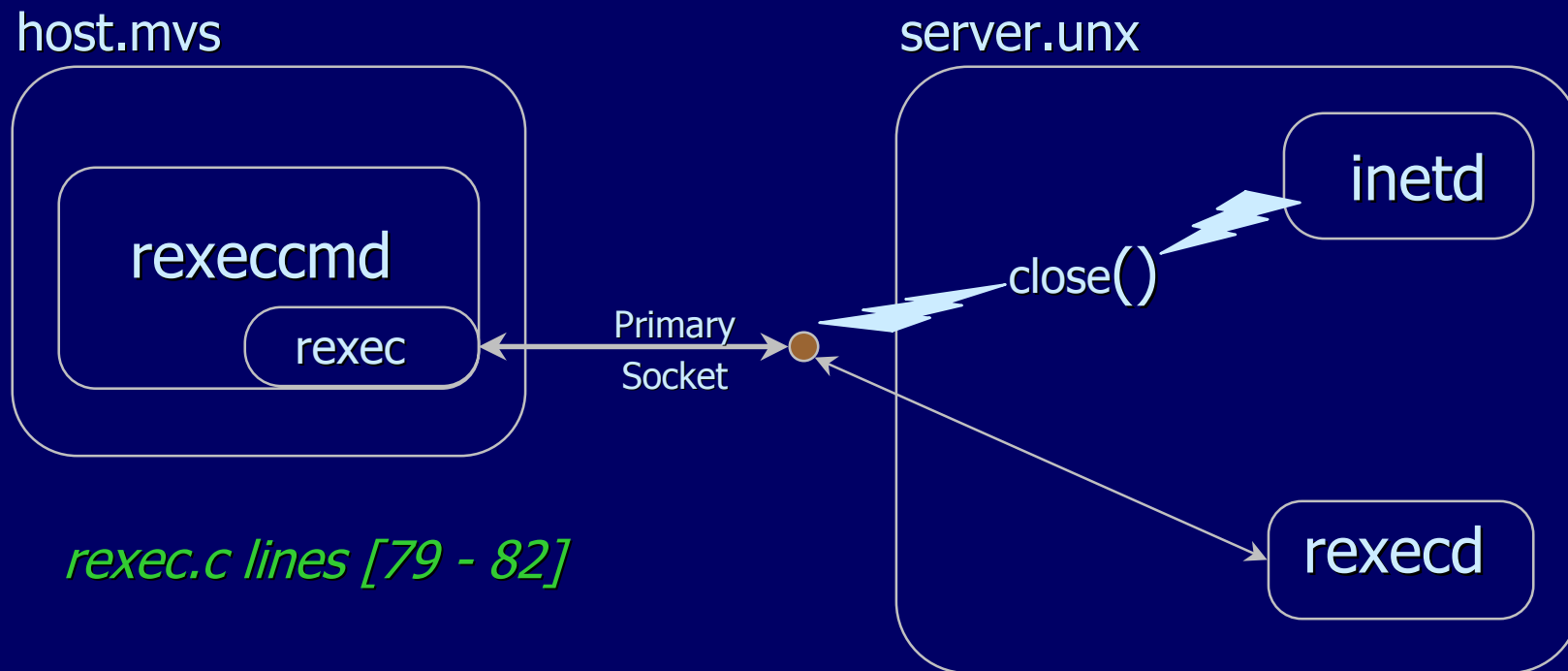
- * For `connect()`, the port and IP address must be specified.
- * For `bind()`, either may be set to zero.
- * Specifying `"sin_port"` as zero requests that the system assign a transient port. Use `getsockname()` to find out what port number was chosen.
- * Specifying `"sin_addr"` as zero binds to all IP addresses on the host (some hosts have more than one).

inetd Create Server Program



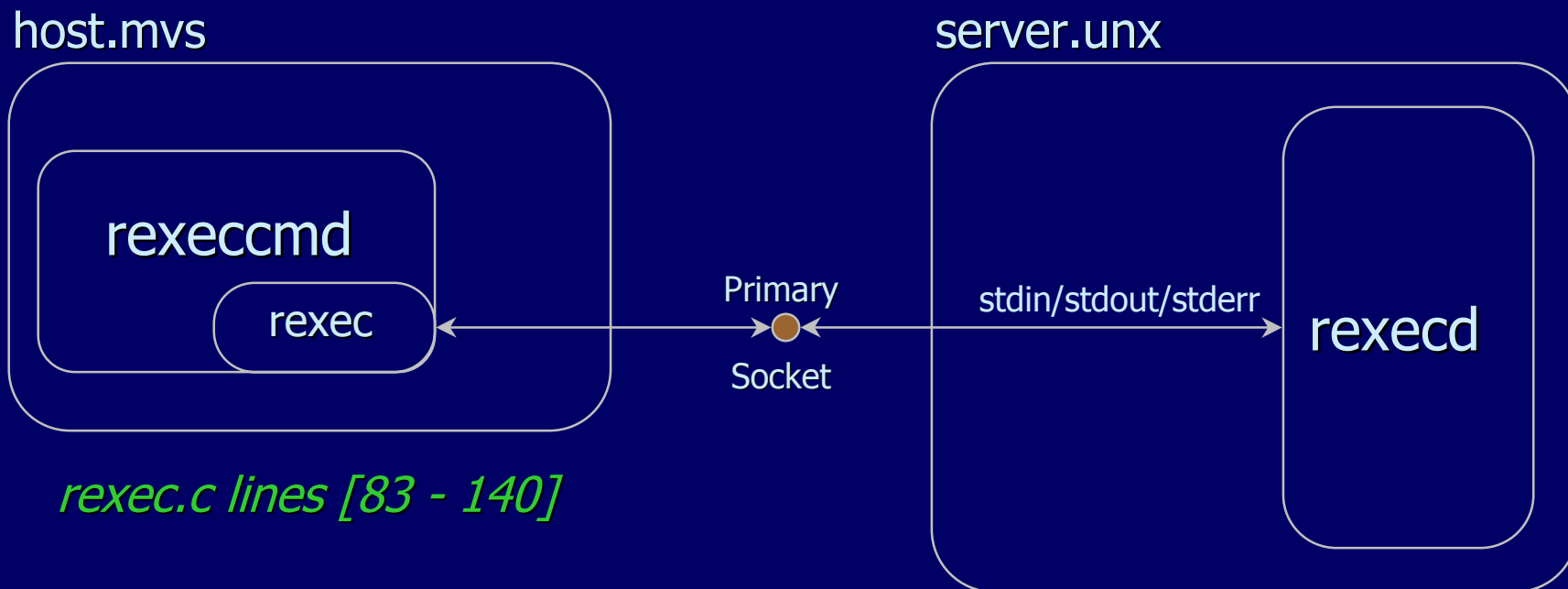
Server creation program (`inetd`) issues a `fork()` and `exec()` call to start `rexecd`. The socket descriptor is attached to `stdin`, `stdout` and `stderr` of `rexecd`.

Server Creation Socket Close



Server creation program (inetd) issues a `close()` to close its connection to the socket, then waits for a new client connection.

Client Establishes stderr port

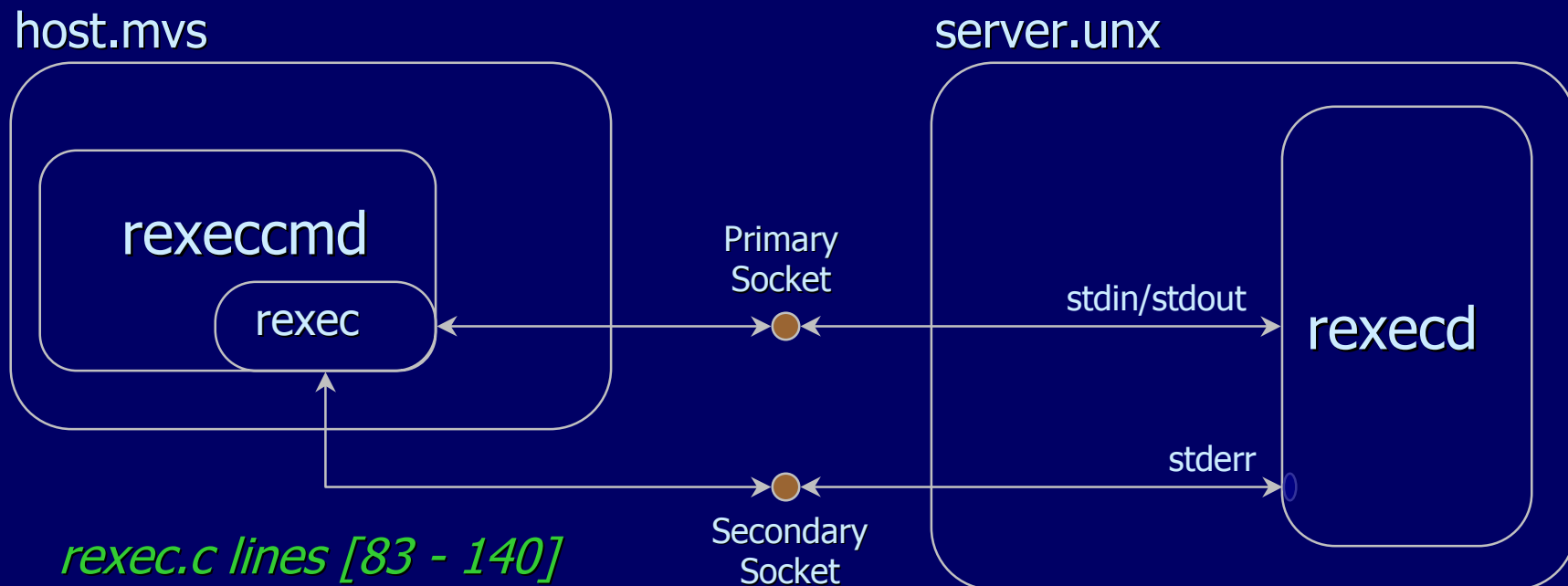


Rexecd reads stdin until a null character is found, if any characters are read before the null they are interpreted as an optional secondary port address to be used for stderr.

recv() and send() conventions

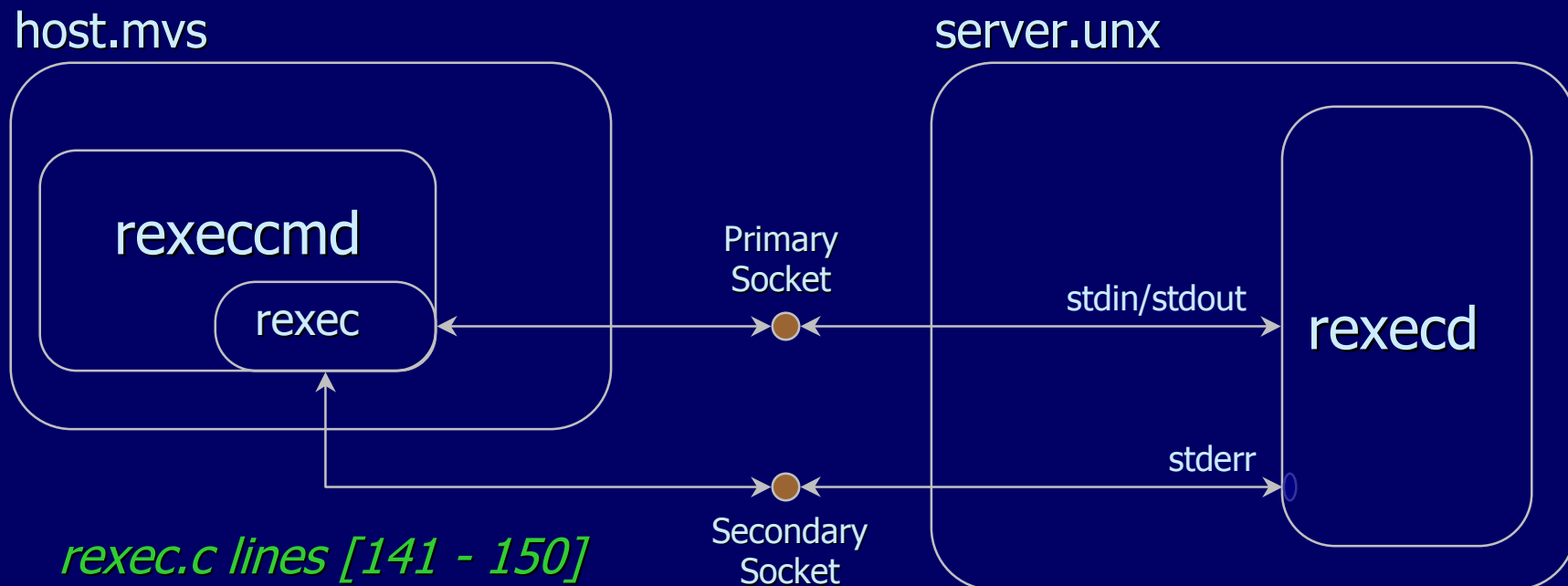
- * `recv()` returns 0 for EOF; >0 for number of bytes read
- * `recv()` often returns fewer bytes than requested; call `recv()` in a loop until you have gotten them all
- * For TCP, `send()` does not create record boundaries; `recv()` may get the result of more or less than one `send()`
- * `recv()/send()` behavior is a common programming error with TCP/IP and socket programming.

rexecd connect() stderr Port



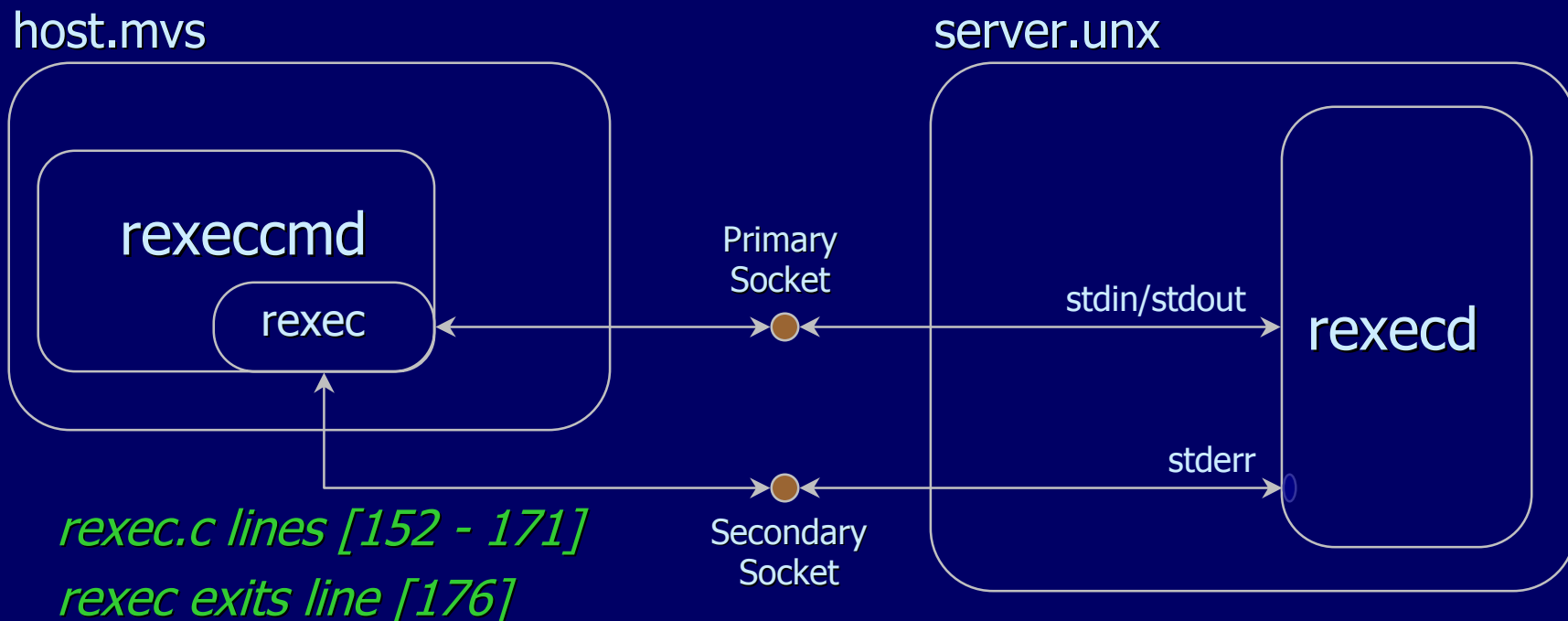
Rexecd connects `stderr` to port created and specified by `rexec`. `Stderr` is used to return error information to `rexec` and to send signal requests (kill calls) to `rexecd`.

Send user, password and command



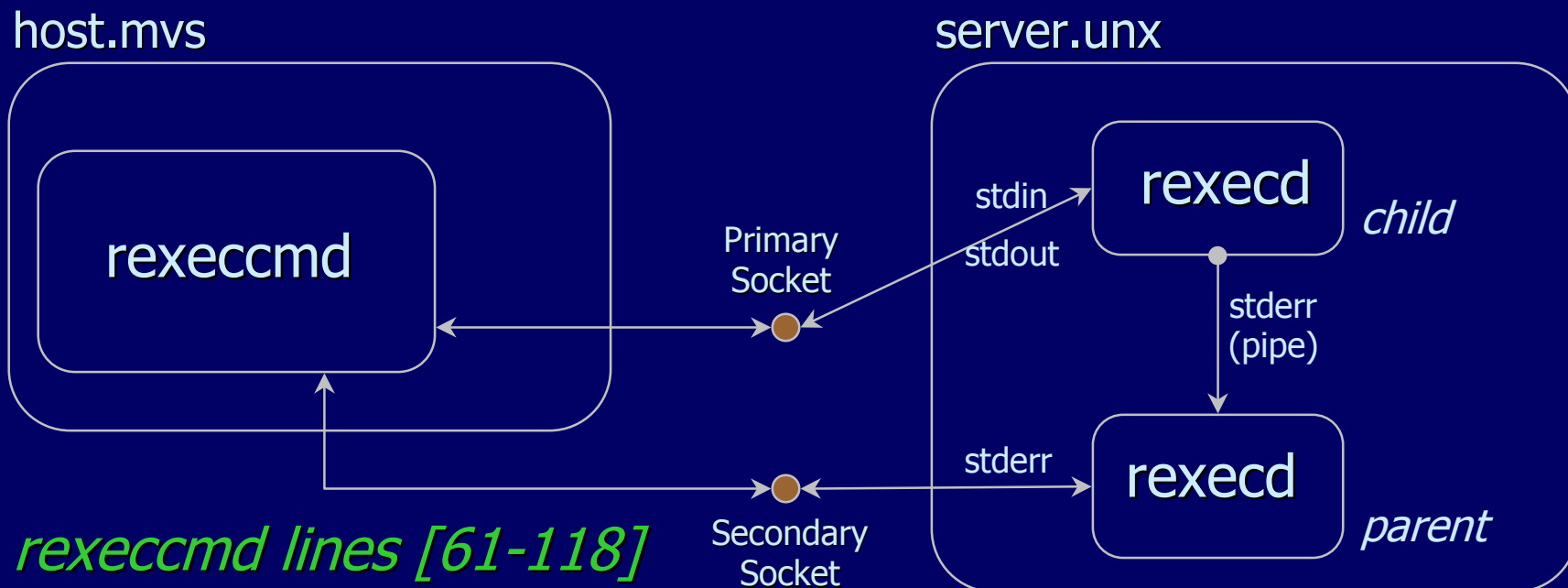
Client `rexec` writes the `userid`, `password` and `command` to execute to the primary socket, where `rxeccd` receives them via reads of the primary socket (which is `stdin`).

Server confirms



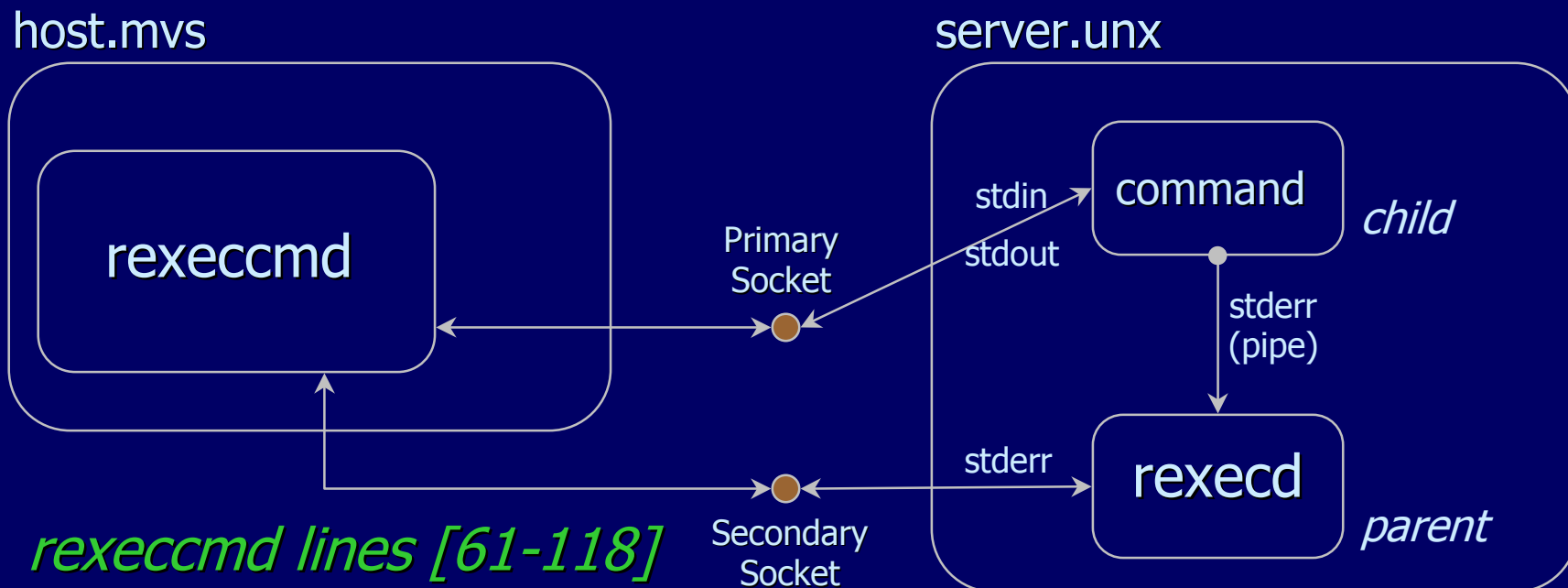
Server **rexecd** writes a one byte value of binary zero to the primary socket (**stdout**) if authorization is successful, otherwise it writes a one byte value of binary one. Client **rexec** receives this via a read of the primary socket

Server creates command executor



rexecd creates a pipe (for stderr) then forks creating a child process in which to run the command. Note that rexecd closes it's connection to the primary socket and that the child process closes it's connection to the secondary socket.

Child Process Executes Command



The child process then configures its environment to match the userid and runs the command using the `execl` call. Output from the command is sent to `rexeccmd` through the primary socket.

File Descriptor Sets

- * A file descriptor is an integer [returned by open() or socket()] which identifies a file or socket.
- * Descriptors are numbered 0 through FD_SETSIZE-1
- * "fd_set" is a type which contains a bit for each descriptor
- * The following macros manipulate file descriptor sets:

```
#include <sys/types.h>
int desc;
fd_set fds;
FD_ZERO(&fds); /* zero entire set */
FD_CLR(desc, &fds); /* clear desc bit */
FD_SET(desc, &fds); /* set desc bit */
FD_ISSET(desc, &fds); /* test desc bit */
```

The select() call

- * The select() system call awaits activity on descriptor sets

```
int select(                /* returns number active */
    int nfd,              /* highest desc used + 1 */
    fd_set *read,         /* Await read on these */
    fd_set *write,        /* await write on these */
    fd_set *except,       /* seldom used */
    struct timeval *timeout /* inactivity timeout */)
```

- * A "timeout" parameter of NULL specifies indefinite waiting
- * A zero time value sets select() non-blocking
- * A non-zero time value, sets a maximum wait time

Summary

- * Reviewed various socket API function calls
- * REXEC / REXECD dialog
- * Useable TCP/IP Socket Application
- * Questions ?

Bibliography

- * Internetworking with TCP/IP: Volumes I, II & III, Douglas Comer, Prentice Hall, 1991 (ISBN Vol I: 0134685059, Vol III: 0138487146)
- * The Whole Internet User's Guide & Catalog by Ed Kroll; O'Reilly & Associates
- * UNIX Network Programming by W. Richard Stevens; Prentice Hall, 1990 (ISBN 0139498761)
- * Socket API Programmer's Reference
- * UNIX "man" pages
- * TCP/IP Illustrated: Volumes 1 & 2, W. Richard Stevens (v2 with Gary R. Wright); Addison-Wesley Publishing Company, 1994