Design and Implementation of an Alternative to SSH

Index

Introduction

Task:

- Alternative to SSH (core function)
- Prototype (design & implementation)
- Target platform: GNU/Linux
- Implementation language: Go (Golang)

Precursory Works

Telnet

- telnet(1)
- Old (RFC15 1969, RFC854 1983)
- Port 23
- No secure connection (TELNETS)
- Go-Telnet

Frequently used Linux commands:

- login(1)
- sh(1)/bash(1)
- cp(1)
- who(1)
- stat(1)
- uptime(1)

Precursory Works

Berkeley r-Commands

- rlogin(1)
- rsh(1)
- rexec(1)*
- rcp(1)
- rwho(1)
- rstat(1)
- ruptime(1)

- Useful (scripts)
- No secure connection

- Replaces telnet(1) and Berkeley `r`-commands
- Port 22
- Secure connection (own protocol)
- Plethora of features:
  - Remote user login
  - Auth via keys
  - Port forwarding
  - X11-forwarding
  - Auth agent connection forwarding (!)
  - Compression (used by `rsync`(1))

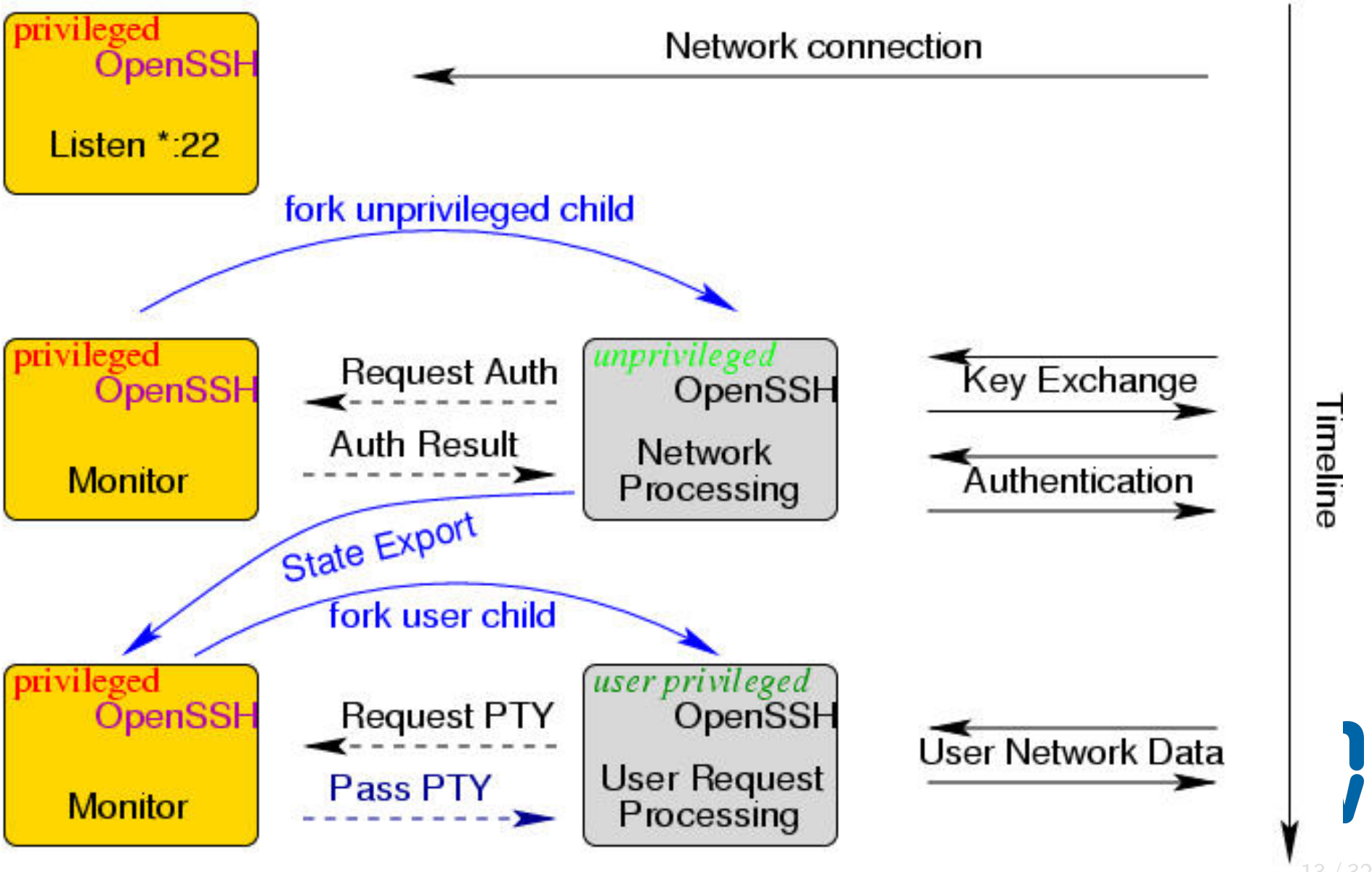
    ⋮

- Prevent MITM, provide integrity & privacy
- TLS 1.3
- Server: `openssl`(1) → key & X.509 certificate
- `crypto/tls`
- Encrypted channel
- Self signed server certificate: Ignores trust chain
- No client certificates (!) → Cannot authenticate the connecting client

- `/etc/passwd` (!)
- PAM
- No Go-package for PAM
- Failure in test environment → `login(1)`
- Failure in same environment using `login(1)`
  Too time consuming to switch back
- `login(1)` allows root login
- Prefetch credentials on client

- Public key cryptography
- Client $\leftrightarrow$ Server
- Random, high entropy secret
- Store authorized public keys on server
- `openssl`(1)
- Authorized keys stored in `/root/.gosh` (plain-text)
  $\rightarrow$ Hash in `~/.gosh/authorized_keys`
  $\rightarrow$ Important for privilege separation

- Shell should run with appropriate permissions
- `setuid`(2) & `setgid`(2)
- Failure to drop privileges after login (operation not supported)
  Thank you, Go $\rightarrow$ spawn shell with appropriate UID & GID
- SSH more sophisticated


privileged
OpenSSH
Listen *:22
Network connection
fork unprivileged child
privileged
OpenSSH
Monitor
Request Auth
Auth Result
unprivileged
OpenSSH
Network
Processing
Key Exchange
Authentication
Timeline
State Export
fork user child
privileged
OpenSSH
Monitor
Request PTY
Pass PTY
user privileged
OpenSSH
User Request
Processing
User Network Data

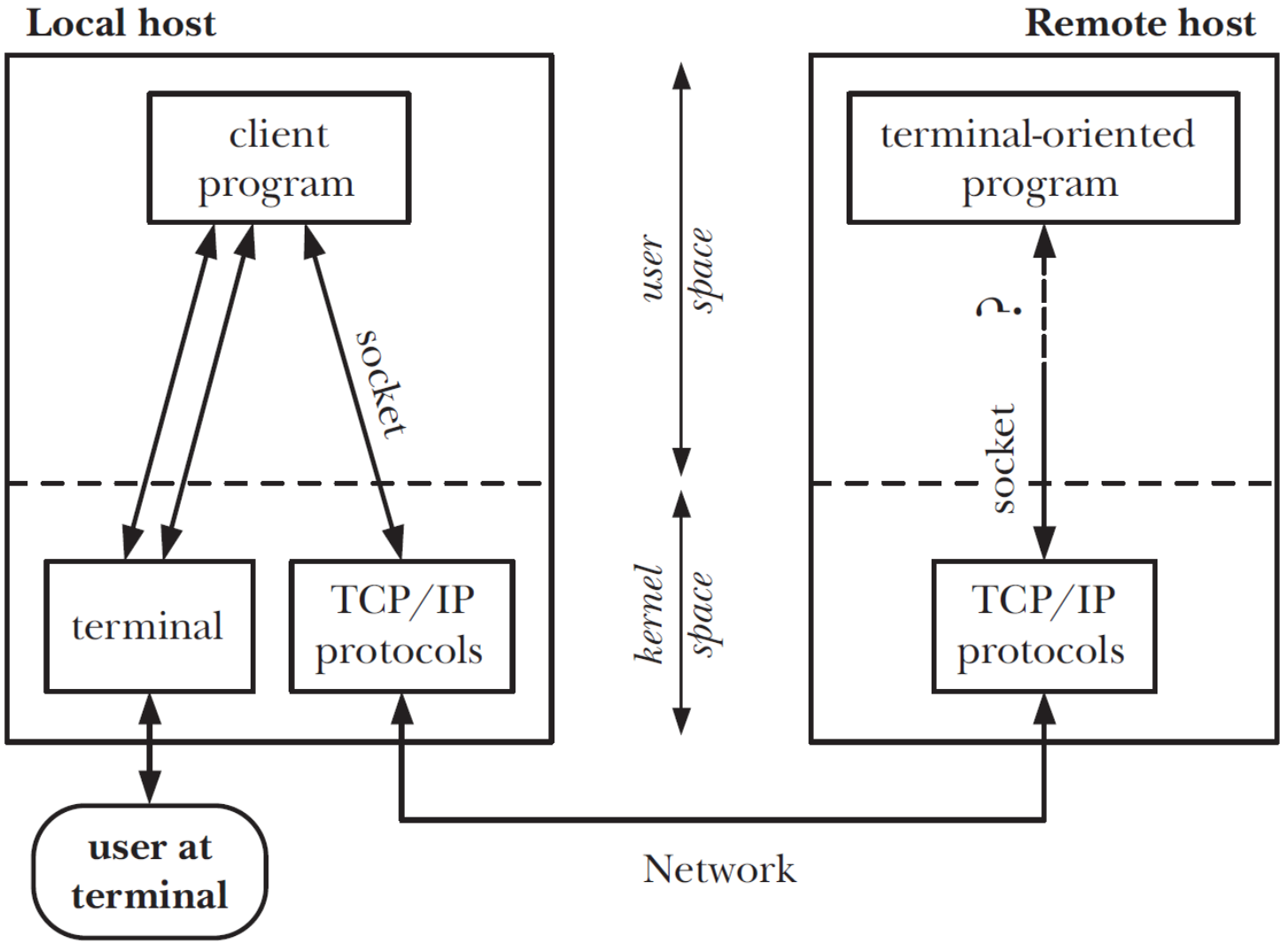
- Server spawns child to handle connection
- `fork`(2)
- Go: No support for forking
- CGO fork fails
- `syscall.ForkExec`
  → High level connection object gets corrupted
- Create host application
- Transfer `fd` as argument to child
  → Low level socket from `x/sys/unix` (x-package!)
- Prospect: Implement proper privilege separation

- Not implemented, **but**
- `utmpx` → `w`/`who`
- PAM: `pam_open_session`(3)/`pam_close_session`(3)

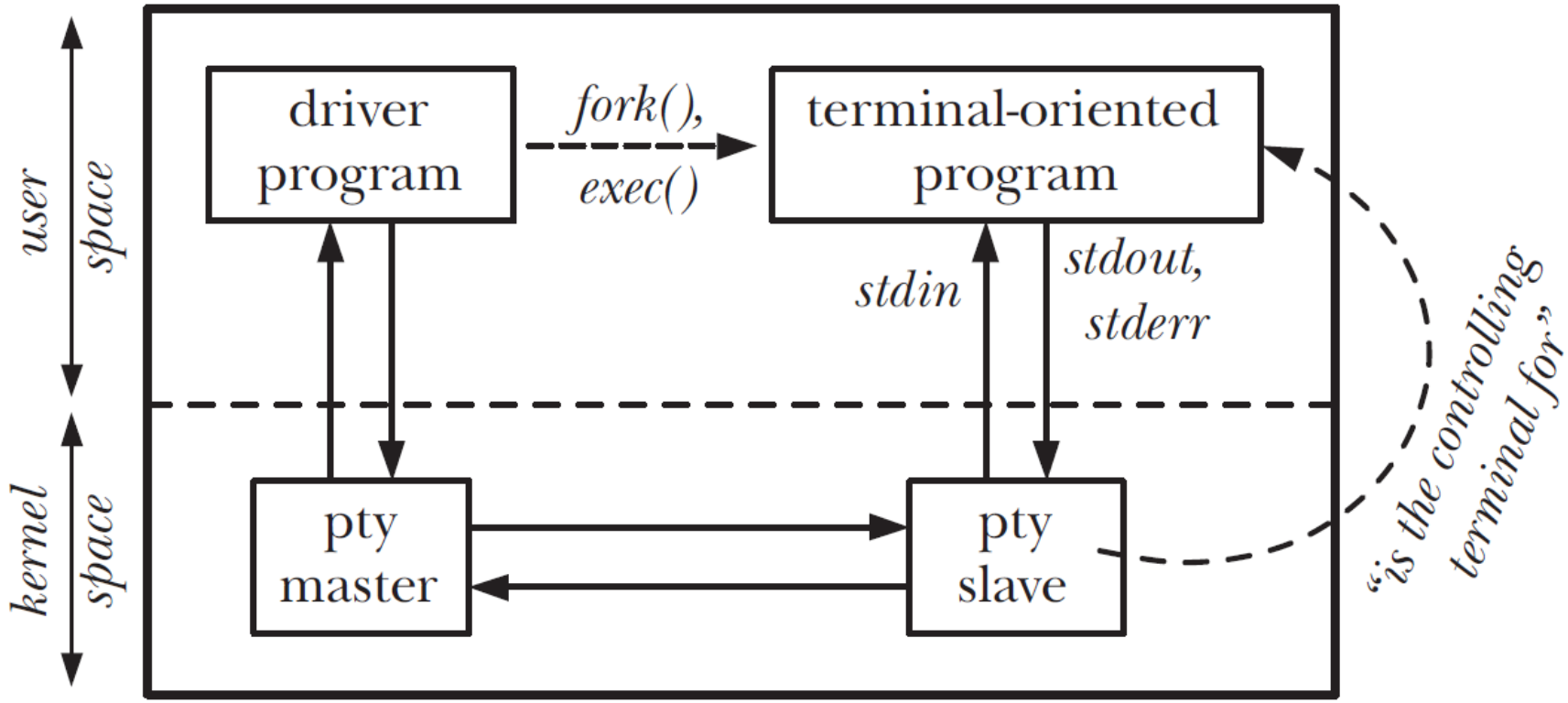
- Home directory, shell, UID & GID
- Go standard library incomplete (misses shell information)
- `/etc/passwd` (!)
- CGO: `getpwnam`(2)/`getpwuid`(2)

```c
#include <sys/types.h>
#include <pwd.h>

struct passwd {
    char *pw_name; /* username */
    char *pw_passwd; /* user password */
    uid_t pw_uid; /* user ID */
    gid_t pw_gid; /* group ID */
    char *pw_gecos; /* user information */
    char *pw_dir; /* home directory */
    char *pw_shell; /* shell program */
};

struct passwd *getpwnam(const char *name);
struct passwd *getpwuid(uid_t uid);
```

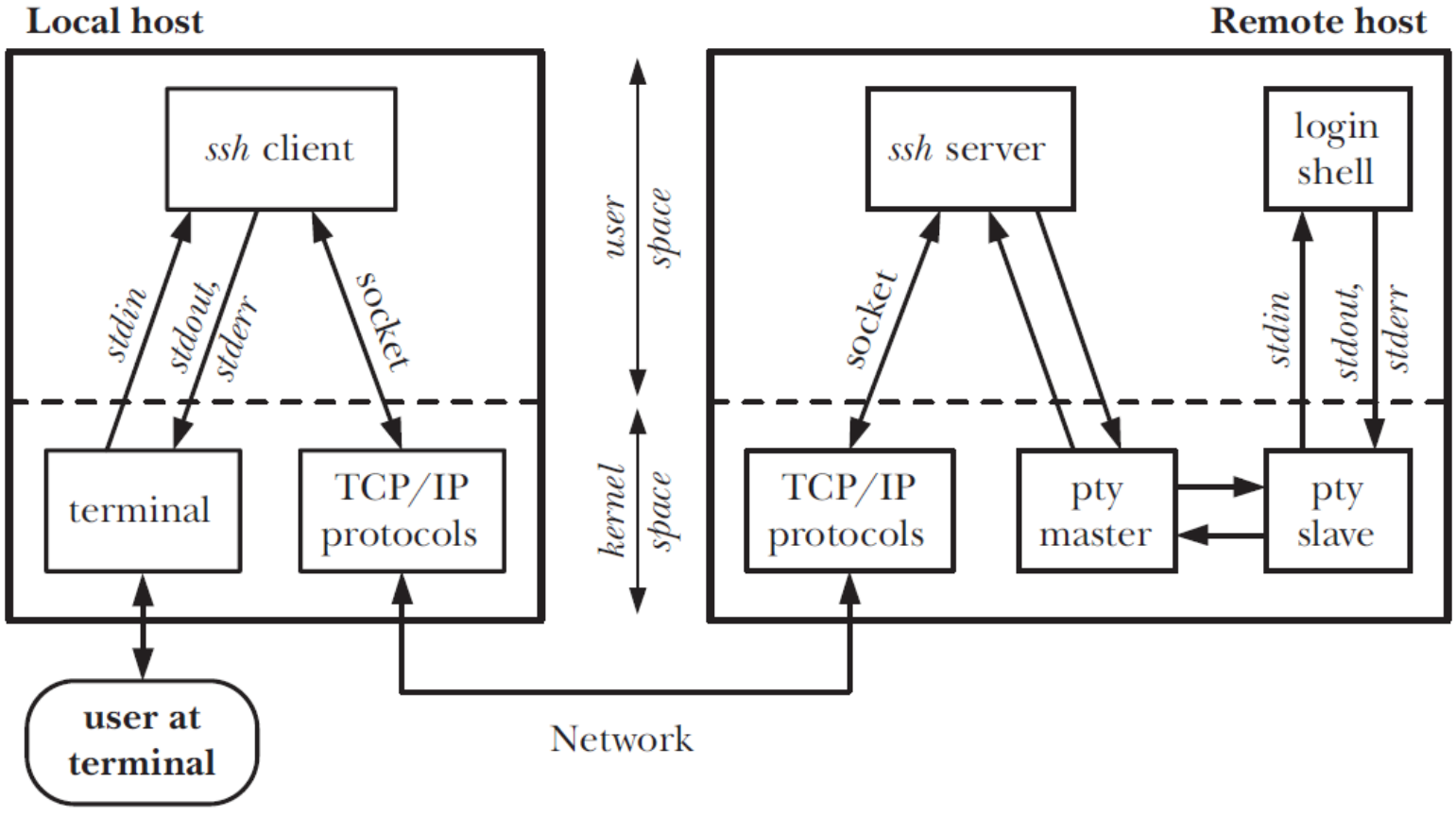
Shells expect to be connected to a TTY



PTY fakes being a TTY



Overview


Local host
Remote host
ssh client
ssh server
login shell
user space
kernel space
stdin
stdout,
stderr
socket
terminal
TCP/IP protocols
pty master
pty slave
user at terminal
Network

- `istty`(3) on the connected `fds`
- `posix_openpt("/dev/ptmx")`(3) → `grant_pt`(3) → `unlockpt`(3) → `ptsname`(3)
- Wrapper function in internal(!) package of the Go standard library `os/signal/internal/pty`

```go
// Open returns a master pty and the name of the linked slave
    tty.
func Open() (master *os.File, slave string, err error) {
    m, err := C.posix_openpt(C.O_RDWR)
    if err != nil {
        return nil, "", ptyError("posix_openpt", err)
    }
    if _, err := C.grantpt(m); err != nil {
        C.close(m)
        return nil, "", ptyError("grantpt", err)
    }
    if _, err := C.unlockpt(m); err != nil {
        C.close(m)
        return nil, "", ptyError("unlockpt", err)
    }
    slave = C.GoString(C.ptsname(m))
    return os.NewFile(uintptr(m), "pty-master"), slave, nil
}
```

- Shell requirements:
  - user (UID & GID) & host name
  - `TERM` env var (for `ncurses`(3X))
  - window resolution (including `SIGWINCH`)
  - session leader (controlling terminal)
- Transfer of env vars (client $\leftrightarrow$ server)
- Continuous transfer of `SIGWINCH` not implemented $\rightarrow$ prospects

```
cmd := exec.Command(pwd.Shell, "--login")
cmd.SysProcAttr = &syscall.SysProcAttr{
    Setsid: true,
    //Setctty: true,
    Credential: &syscall.Credential{
        Uid: pwd.Uid,
        Gid: pwd.Gid,
    },
}
cmd.Env = userEnvs
```

Setting `CTTY` flag (for controlling terminal) fails → prospects

- Forward all keystrokes without interpretation (client-sside)
- `cooked` mode $\rightarrow$ `raw` mode
- x-package (!) `x/crypto/ssh/terminal`

```go
import "golang.org/x/crypto/ssh/terminal"
//...
oldState, err := terminal.MakeRaw(0)
if err != nil {
    panic(err)
}
defer terminal.Restore(0, oldState)
```

- client $\leftrightarrow$ server $\leftrightarrow$ ptm $\leftrightarrow$ pts $\leftrightarrow$ shell
- `/dev/zero` $\rightarrow$ connection (client-side) $\rightarrow$ server $\rightarrow$ `pv -rabtW` $\rightarrow$ `/dev/null`
- TLS vs no TLS

Throughput with:	TLS (size) MiB/s (GiB)	no TLS (size) MiB/s (GiB)
Arch Linux (loopback)	427 (25.1)	1177.6 (69.0)
WSL (loopback)	69.7 (4.09)	116 (6.82)
Arch Linux to WSL (ethernet*)	85.1 (4.99)	83.7 (4.91)

*: Netgear Switch & Cat 5 ethernet cable

- TLS vs plain text
- Key auth vs only password auth

Evaluation

Comparison to Berkeley `r`-commands

- Only `rlogin`(1) is considered (`rsh`(1))
- TLS vs plain text
- Key auth vs only password auth
- Password auth: Both use `login`(1)

- TLS vs own protocol
- Privilege separation
- Many additional features

Conclusion

- Many problems encountered
- Many new concepts learned
- Mixed feelings

End

Thank you for your attention!