# 1 System Calls

## 1.1 Pipe

returns: 0 success, -1 error

puts FDs of pipe in the argument array

```
int pipe(int pipeFD[2]);
```

## 1.2 Dup

```
int newFD = dup(int oldFD);
int newFD = dup2(int oldFD, int newFD);
```

dup2 copies oldFD onto newFD, so the fileD at newFD becomes oldFD

## 1.3 Fork

`pid_t fork()`

returns pid to parent, and 0 to child.

## 1.4 Exec

All exec functions replace the call stack. The first element of argv must be the filename to execute.

```
int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);
int execvpe(const char *file, char *const argv[],
char *const envp[]);
```

### 1.4.1 Example

```
dup2(hubToPlayer[0], 0);
dup2(playerToHub[1], 1);
dup2(devNull, 2);

char playerIDArg[ARG_SIZE];
sprintf(playerIDArg, "%d", i);

execlp(playerExecutables[i],
    playerExecutables[i],
    numPlayersArg, playerIDArg,
    thresholdArg, handArg, (char*) 0);
```

## 1.5 Wait

```
int exitStatus;
pid_t wait(int *stat_loc);
// possible options: WNOHANG
int options = 0;
//pid_t waitpid(pid_t pid, &exitStatus, int options);
pid_t waitpid(451, &exitStatus, 0);
int status = WEXITSTATUS(exitStatus);
bool WIFEEXITED(exitStatus);
    // true if exit() was called by child
int WEXITSTATUS(exitStatus);
    // The value given to exit() by child
bool WIFSIGNALED(exitStatus);
    //true if exit without exit()
int WTERMSIG(exitStatus);
    // the signal that killed child
```

## 1.6 Signals

1 HUP, 2 INT, 9 KILL, 11 SEGV, 13 PIPE

```
static void sighup_handler(int signum);
int main() {
    struct sigaction saHup;
    saHup.sa_handler = sighup_handler;
    saHup.sa_flags = SA_RESTART;
    sigaction(SIGHUP, &saHup, NULL);
}
int main() {
    sigset_t signalMask;
    sigemptyset(&signalMask);
    sigaddset(&signalMask, SIGPIPE);
    pthread_sigmask(SIG_BLOCK, &signalMask, 0);
}
void *handler(void *) {
    sigset_t waiton; // setup sigset
    while (!sigwait(&waiton)) {
        // do things when receiving signal
    }
}
```

## 1.7 PThread

```
pthread_mutex_init(pthread_mutex_t);
int pthread_create(&threadID, attr,
        void*(*func)(void*), void*arg);

void pthread_exit(void *retval);
int pthread_join(threadID, void**retval);

sem_init(sem_t *sem, 0, initialVal);
sem_post(sem_t); sem_wait(sem_t*);
sem_trywait(sem_t*);
```

## 1.8 stdio

```
FILE *fdopen(int FD, char *mode)
int sscanf(string, format, ...)
char *fgets(char *retstr, int maxchars, FILE*)
```

Reads until eof or newline, terminating newline is stored.

# 2 Networks

## 2.1 DNS

Over UDP

## 2.2 UDP

Discrete *Datagrams*, no handshake and verification. Messages have a mx size, no delivery acknowledgement messages (unless you implement it on top).

## 2.3 TCP

Bi-Directional Connection oriented. Provides reliability (keep sending until you get an ACK).

Segments

ACK and NAK Messages.

## 2.4 Notation

CIDR: Set host bits to 0 `/numnetbits`. Netmask: Set all network bits to 1, all host bits to 0.

## 2.5 Special Addresses

Gateway: All host bits 0. Broadcast: All host bits 1.

### 2.5.1 Non-routable ips

- 10.0.0.0 / 8
- 192.168.0.0 / 16
- 127.0.0.0 / 8 Loopback
- 169.254.0.0 / 16 ONLY used for fallback when DHCP failed

## 2.6 NAT = Network Address Translation

Rewrites source IP for TCP requests, and keeps track using incoming port-outgoing port on each side of the network.

## 2.7 Layers

1. Physical
   - Wires
2. DataLink
   - Ethernet
   - MAC Addressing (48bit)
3. Network
   - IP: Internet Protocol
   - IPv4/v6 Addressing (v4 32 bit)
4. Transport
   - UDP/TCP
   - Port numbers
5. Application
   - HTTP + HTML
   - GET/POST

# 3 Memory

```
void *malloc(size_t size);
void free(void *ptr);
void *calloc(size_t nmemb, size_t size);
void *realloc(void *ptr, size_t size);
```

### 3.1 TLB = Translation Look-aside Buffer

Hardware memory page→frame cache, global.

### 3.2 Disk fragmentation

Internal: There is unused space in allocated blocks (because files are smaller than blocksize). External: There is no large contiguous free space; small files are created and deleted everywhere, bad indexing for expanding files.

### 3.3 C Types

- int: $\geq$ 16bits
- INT modifiers: signed,unsigned,short,long
- `long` alone implies int $\geq$ 32bits
- long long: $\geq$ 64 bit int.
- float: single precision float 32bits
- double: double precision float 64bits
- long double: extended precision float 96/128bit

## 4 Bash

Wildcards: `*` and `?`

`uniq`: comp consecutive lines [-d (print duplicates) –c (prefix line with count of occurences) –i (ignore case) –u (print unique lines) –sN (skip first N chars) –wN (only count first N chars)

`sort`: [-r] reverse order [-k] key

`kill`: -s SIGNAME

- `head / tail` -n number of lines

`grep`: -v invert, regex: ^ beginning of line, $ end of line, . any character, * match 0 or more times, [] match any one character inside [].

`tr`: -s 'c', combine all ocurences of 'c' in line into one, -d 'c' delete all occurences of c

`pgrep` : -x exact match, -c count, -u $USER, user

```
cut -f$fieldNum -d$delimiterchar
chmod $mode $filename
```

```
ln -s $linkname $targetname

$ ps -ef
UID        PID  PPID   C STIME TTY          TIME CMD
user    procID parenID                      age  name

$ ls -li | tr -s ' '
4719745 -rwxrw-r-- 1 root root 2048 Jan 13 07:11 bob
$ ls -d
.
$ ls -1
alice
bob
```

- inode number
- `[-/d/l]uuugggooo` File, symlink, or directory, then permissions
- Number of hardlinks
- owner name
- owner group
- file size (bytes)
- modification time and file name
- symlink info

```
 ----------------
< GNU can do it! >
 ----------------
  \
   \
    \     (____)
           (oo)------___
           (__)\        \/\
               ||----\ |
               ||     ||
```