```
int execv(const char *path, char *const argv[]);pid_t getpid(void); pid_t getppid(void);
pid_t fork(void); // rets 0 in child, pid of child in parent
void exit(int status); // tells parent dying with SIGCHLD, tell child dying with SIGHUP
pid_t wait(int *status);pid_t waitpid(pid_t pid, int *status, int options); // -1, NULL, 0
int dup(int oldfd); int dup2(int oldfd, int newfd); int pipe(int fds[2]);
int stat(const char *path, struct stat *buf); // also fstat, fails if file not there
Struct stat {
  mode_t st_mode;     /* file type  & mode & permissions */  // File types (st_mod)
  ino_t     st_ino;   /* file inode number */                // Reg (S_ISREG(buf.st_mode))
  dev_t    st_dev;    /* device number (file system) */      // Directory (S_ISDIR(..))
  dev_t   st_rdev;    /* device number for special files */  // Character dev (S_ISCHR(..))
  nlink_t st_nlinks;  /* number of links */                  // Block dev (S_ISBLK(..))
  uid_t    st_uid;    /* owner user ID */                    // FIFO (S_ISFIFO(..))
  gid_t   st_gid;     /* owner group ID */                   // Symbolic link (S_ISLNK(..))
  off_t   st_size;    /* size in bytes, for regular files */ // Socket (S_ISSOCK(..))
  time_t st_atime;    /* time of the last access */          //
  time_t st_mtime;    /* time of the last modification */    //
  time_t st_ctime;    /* time of the last status change */ }
mode_t umask(mode_t mask); // bits that are 1 are turned off.
int open(const char *pathname, int flags);
int creat(const char *pathname, mode_t mode);
int remove(const char *path); /* C function, same as link */
int rename(const char *oldname, const char *newname)
int link(existingpath, newpath) //create new ent in dir with same inode #
// hard link not allowed for dirs, creates loops, same inode #, ls field++
int unlink(const char *path); // rm only symlink removed, open follows symlink by default
int symlink(const char *actualpath, const char *sympath);
int readlink(const char *pathname, char *buf, int bufsize);
off_t lssek(int filedes, off_t offset, int whence) //whence = SEEK_SET, SEEK_CUR, SEEK_END
// Only root can write dirs, others read/use syscalls to write
DIR *opendir(const char *name)
struct dirent *readdir(DIR *dp); // returns NULL at end of dir
void rewinddir (DIR *dp); // resets dp to start of dir
int closedir(DIR *dp)
struct dirent {ino_t d_ino; char *d_name[NAME_MAX+1]; }
int chdir(const char *pathname); also fchdir // only affects cwd for current process
char *getcwd(char *buf, size_t size);
// Signals: proc can: block for a time; ignore; default; catch and run user-defined func
signal(int sig, void (*disp)(int)))(int); // catch sig, run disp fun (or SIG_IGN/SIG_DFL)
Int sigemptyset(sigset_t *set);
Int sigfillset(sigset_t *set);
Int sigaddset(sigset_t *set, int signo);
Int sigdelset(sigset_t *set, int signo);
Int sigismember(const sigset_t *set, int signo);
Int sigprocmask(int how, const sigset_t *set, sigset_t  *oset);
sigprocmask(SIG_BLOCK, &newmask, &oldmask);
/* critical region */
sigprocmask(SIG_SETMASK, &oldmask, NULL);
Int sigaction(int signo, const struct sigaction *act, struct sigaction *oact)
    Struct sigaction {
      void (*sa_handler)(); /* signal handler */
      sigset_t sa_mask;     /*additional signal to be block */
      int sa_flags;         /* various options for handling  signal */
    };
Int kill(pid_t pid, int signo);
Unsigned int alarm(unsigned int seconds) // sends SIGALRM in x seconds
// Syscalls may return prematurely b/c of signal, check syscall ret val
Struct termios {
    tcflag_t c_iflag;       /* input flag */
    tcflag_t c_oflag;       /* output flag */
    tcflag_t c_cflag;       /* control flags */
```

```
    tcflag_t c_lflag;       /* local flags */
    cc_t     c_cc[NCCS];    /* control characters */
}
int tcgetattr(int fildes, struct termios *termios_p)
int tcsetattr(int fildes, int optional_actions, const  struct termios *termios_p)
// Process group has leader whose pid = pgid
pid_t getpgrp(void)
int setpgid(pid_t, pid, pid_t, pgid) // proc can set gid of self and children
// Session, 1+ process grp, ps -j, ex. Login shell, leader est. conn to control term, only
one 1/o dev/session, one fg grp, many bg grps, input only to fg, output shared
pid setsid(void); // become session leader, new grp leader of new grp, fails if grp leader
pid_t  tcgetpgrp(int filedes);
int tcsetpgrp(int filedes, pid_t pgrpid);
// Job control, control mult jbs, fg accesses terminal, when bg job tries to read: SIGTTIN
// Orphan proc grp: parent of each proc is orphan or not member of grp's session
// Term I/o, signals: create new grp/job (p/chld setpgid), fg jobs tcsetgrp to give term
// Shell w/o job ctrl: all prcs in same sess/grp, every proc can open /dev/tty
int msgget(key_t key, int  msgflag); // Create message queue
key_t ftok(const char *path, int id);
Int msgget(key_t, int flag)
Int msgctl(int msgid, int cmd, struct msgid_ds *buf)
Int msgsnd(int msgid, const void *ptr, size nbytes, int flag);
Int msgrcv(int msgid, void *ptr, size_t nbytes, long type, int flag);
int shmget(key_t key, size_t size, int shmflg);
void *shmat(int shmid, void *shmaddr, int shmflg);
int shmdt(void *shmaddr);
int shmctl(int shmid, int cmd, struct shmid_ds *buf);
 semget( … )
Incr/Decr/Test-and-set : semop(…)
Deletion: semctl(semid, 0, IPC_RMID, 0);
for all pipes //pipes funct
  pipe(fds[i]);
if (fork()== 0) { // last
  close(0); dup(fds[numPipes - 1][0]);
  close(fds[numPipes - 1][1]);
  for all pipes
    close(fds[i][0]);
    close(fds[i][1]);
    execv(argvs[numPipes][0], argvs[numPipes]);
for all pipes going down // middle
  if (fork()== 0) {
    close(1); dup(fds[i][1]);
    close(0); dup(fds[i - 1][0]);
    for all pipes
      close(fds[i][0]);
      close(fds[i][1]);
      execv(argvs[i][0], argvs[i]);
if (fork() == 0) { // first pipe
  close(1); dup(fds[0][1]);
  for all pipes
    close(fds[i][0]);
    close(fds[i][1]);
    execv(argvs[0][0], argvs[0]);
for all pipes
    close(fds[i][0]);
    close(fds[i][1]);
for all pipes
    waitpid(-1, NULL, 0);
```