1)

a) ('-l', 'maynard', 'srobbins', 'krobbins', 'dj')

b) 4 ✓

c) 4 ✓

d)

i) 'maynard' ✓

ii) "vip\n" ✓

iii) 'srobbins' ✓

iii) 'krobbins', 'maynard' ✓

16

e)

i) "weebone\n" ✓

ii) "vip\n", "vipo\n", "fred\n", "visual64\n", "zorkmd\n"

iii) "vip2\n", "visual64\n"

2)

```
$outfile = pop;
open(OUT, "> $outfile") or die "Unable to open $outfile $!\n";

foreach $file (@ARGV) {
  chomp $file;
  open (FILE, $file) or die "Unable to open $file $!\n";
  @lines = <FILE>;
    print OUT "$file:\n", @lines[0..4], "\n";
}
```

17

5 lines not 6 lines!

This interpolates inside a string separating with spaces!

3)
```
$regex = shift;
@dirs = <>;
chop @dirs;
@files = `find @files -print`;
chomp @files;
@tfiles = grep(-f && -T, @files);
foreach $file(@tfiles){
    open(FILE, $file) or die "unable to open $file $!\n";
    while (<FILE>){
        if (/$regex/){
            print "$file: $_";
        }
    }
}
```

20

4)
```
%grade = ();
@pdirs = @ARGV;
chop @pdirs;
foreach $dir (@pdir){
    @students = `ls -a $dir`;
    chop @students;
    foreach $stdt (@students){
        next unless $stdt eq "." or $stdt eq "..";
        open(REP, $stdt) or die "unable to open $stdt $!\n";
        @lines = <REP>;
        while(<REP>){
            if (/^Total Points:\s+(\d+).*$/){
                $grade{$stdt} += $1;
            }   }
    }
}
foreach $stdt (keys %grade){
    print "$stdt:\t $grade{$stdt}\n";
}
```

There are no newlines on the command line!

no hidden files!

17

"$dir/$stdt"

You already read to the EOF? nothing is left!

5)

```
int reccopy(int num1, char *fdb1, int num2, char *fdb2, int rsz){
    int fdin, fdout, sizz, ...
    char buf[rsz]
     int curr = 0;
    if((fdin = open(fdb1, O_RDONLY)) < 0){
        perror("open error");
        close(fdin);
        return -1;
    }
    if((fdout = open(fdb2, O_RDWR)) < 0){
        perror("open error");
        close(fdin);
        close(fdout);
        return -1;
    }
    // move records down starting from num2
    sizz = lseek(fdout, 0, SEEK_END)/rsz;
    lseek(fdout,(size-1)*rsz, SEEK_SET
    while((read(fdout, &buf, rsz) == rsz) && curr != num2)
        if(write(fdout, &buf, rsz) != rsz){
            perror("write error")
            return -1;
        }
        curr = lseek(fdout, -3*rsz, SEEK_CUR)/rsz;
    }
    // get num from fdb1 and put it on fdb2
    lseek(fdin, num1*rsz, SEEK_SET);
    if(read(fdin, &buf, rsz) != rsz){
        perror("read error");
        return -1
    }
    lseek(fdout, num2*rsz, SEEK_SET);
    if(write(fdout, &buf, rsz) != rsz){
        perror("read error");
        return -1;
    }
    close(fdin);
    close(fdout);
    return 0;
}
```

22

- Not starting from num2 - doesn't work if num2==0 / bad curr initialization!
- unnecessary
- Doesn't work if num2 is the last record!

6)

```
int main(
  int fd[2];
  if(pipe(fd)){
      perror("pipe error");
      exit(-1);
  }
  switch(fork()){
      case -1;
        perror("fork error");
        exit(-1);
      case 0;
        close(fd[0]);
        dup2(fd[1], STDOUT_FILENO);
        close(fd[1]);
        execlp("diff", "diff", "/var/log/ps.udp.1", "/var/log/ps.udp.0", NULL);
        err_sys("exec error");
      default:
        close(fd[1]);
        dup2(fd[0], STDIN_FILENO);
        close(fd[0]);
        execlp("mail", "mail", "-s", "Attack", "root", NULL);
        err_sys("exec error");
  }
  return 0;
}
```

25

7)

```
int ppopen(const char *command, const char *type){
  int fd[2]

  if(pipe(fd)){
     perror("pipe error");
     return -1;
  }

  switch(fork()){
      case -1:
        perror("pipe error");
        return -1;
      case 0:
        if(!strcmp(type,"r")){
          close(fd[0]);
          dup2(fd[1], STDOUT_FILENO);
          close(fd[1]);
          execlp(command, command, NULL);
        }
        else{
          close(fd[1]);
          dup2(fd[0], STDIN_FILENO);
          close(fd[0]);
          execlp(command, command, NULL);
        }
        perror("exec error");
        return -1;
  }

  return fd;
```

22

No, Return a file descriptor and close the other!

Having the address of an invalid array is of no help!

8)

```
int main () {
int i,n, fd[35][2], ndx,
pid_t pid;
struct message {
    pid_t pid;
    int ndx;
} msg


for(i=0; i<35; i++){
    if(pipe(fd[i])){
        perror("pipe error");
        exit(-1);
    }
}

ndx = 0

for(i=1; i<35; i++)
    switch(parent = fork())
        case -1:
            perror("fork error");
            exit(-1);
        case 0:
            ndx = i;
    }
    if(parent){
        break
    }
}

for(i=0; i<35; i++)
    if(ndx != i){
        close(fd[i][0]);
    }
}
close(fd[ndx][1]);

pid = getpid();
```

23

2.)

```
if (pid != 2) {
    msg.pid = pid;
    msg.ndx = ndx;
    for (i=0; i<35; i++) {
        if (ndx != i) {
            if (write (fd[i][1], &msg, sizeof(msg)) != sizeof(msg))
                perror("write error");
                exit(-1);
            }
        close(fd[i][1]);
    }
}
while( read (fd[ndx][0], &msg, sizeof(msg)) != sizeof(msg)) {
    printf("Process %d index %d heard from process %d index %d",
            pid, ndx, msg.pid, msg.ndx);
}
return 0;
}
```

didn't close write ends of even processes!

hangs

9)

```
int main ( ) {
    struct message {
        char prog[128];
        char rfifo[128];
        char data[128];
    } msg;
    int FifoIN , fd[2], fdout

    if ((FifoIN = open ("/tmp/dispatch", O_RDWR)) < 0 ) {
        perror ("Fifo error");
        exit (-1);
    }
    while ( read (fifoIN, &msg , sizeof(msg) ) == sizeof(msg) ) {
        switch ( fork() ) {
            case -1:
                perror ("fork error");
                exit (-1);
            case 0:
                close (fifoIN);
                switch ( fork() ) {
                    case -1:
                        perror ("fork error");
                        exit (-1);
                    case 0:
                        if ( pipe (fd) ) {
                            perror ("pipe error");
                        } exit -1;
                        if (write ( fd[1], msg.data, 128 ) != 128 ) {
                            perror ("write error");
                            exit (-1);
                        }
                        close (fd[1]);
                        if ((fdout = open (msg.rfifo, O_WRONLY )) < 0 ) {
                            perror ("open error");
                            exit (-1);
                        }
```

20

Read the problem!

9)

```
        dup2 (fdout, STDOUT_FILENO);
        dup2 (fd[0], STDIN_FILENO);
        close(fdout);
        close(fd[0]);
        execlp (msg.prog, msg.prog, NULL);
        perror("exec error");
        exit(-1);
    }
    default:
        exit(0);
default:
    wait(NULL);
}

return 0;
}
```