```
-- front
6.828 Shells Lecture

Hello.

-- intro
Bourne shell

Simplest shell: run cmd arg arg ...
        fork
                exec in child
                wait in parent

More functionality:
        file redirection: cmd >file
                open file as fd 1 in child before exec

Still more functionality:
        pipes: cmd | cmd | cmd ...
                create pipe,
                run first cmd with pipe on fd 1,
                run second cmd with other end of pipe on fd 0

More Bourne arcana:
        $* - command args
        "$@" - unexpanded command args
        environment variables
        macro substitution
        if, while, for
        ||
        &&
        "foo $x"
        'foo $x'
        `cat foo`

-- rc
Rc Shell


No reparsing of input (except explicit eval).

Variables as explicit lists.

Explicit concatenation.

Multiple input pipes <{cmd} - pass /dev/fd/4 as file name.

Syntax more like C, less like Algol.

diff <{echo hi} <{echo bye}

-- es
Es shell


rc++
```

Goal is to override functionality cleanly.

Rewrite input like cmd | cmd2 as %pipe {cmd} {cmd2}.

Users can redefine %pipe, etc.

Need lexical scoping and let to allow new %pipe refer to old %pipe.

Need garbage collection to collect unreachable code.

Design principle:
	minimal functionality + good defaults
	allow users to customize implementations

	emacs, exokernel

-- apps
Applications

Shell scripts are only as good as the programs they use.
	(What good are pipes without cat, grep, sort, wc, etc.?)

The more the scripts can access, the more powerful they become.

-- acme
Acme, Plan 9 text editor

Make window system control files available to
everything, including shell.

Can write shell scripts to script interactions.

/home/rsc/bin/Slide
/home/rsc/bin/Slide-
/home/rsc/bin/Slide+

/usr/local/plan9/bin/adict

win

-- javascript
JavaScript

Very powerful
	- not because it's a great language
	- because it has a great data set
	- Google Maps
	- Gmail
	- Ymail
	- etc.

-- greasemonkey
GreaseMonkey

```
// ==UserScript==
// @name            Google Ring
// @namespace       http://swtch.com/greasemonkey/
```

```
// @description     Changes Google Logo
// @include         http://*.google.*/*
// ==/UserScript==

(function() {
        for(var i=0; i<document.images.length; i++){
                if(document.images[i].src ==
"http://www.google.com/intl/en/images/logo.gif")
                        document.images[i].src =
"http://swtch.com/googlering.png";
        }
})();

-- webscript0
Webscript

Why can't I script my web interactions?

/home/rsc/plan9/bin/rc/fedex

webscript /home/rsc/src/webscript/a3
        /home/rsc/src/webscript/a2

-- acid
Acid, a programmable (scriptable) debugger

defn stopped(pid)
{
        pfixstop(pid);
        pstop(pid);
}

defn pfixstop(pid)
{
        if *fmt(*PC-1, 'b') == 0xCC then {
                // Linux stops us after the breakpoint, not at it
                *PC = *PC-1;
        }
}

/usr/local/plan9/acid/port:/^defn.bpset
/usr/local/plan9/acid/port:/^defn.step

defn checkpdb(pdb)
{
        loop 1,768 do {
                if *pdb != 0 then { print(pdb\X, " ", *pdb\X, "\n"); }
                pdb = pdb +4;
        }
}

-- guis
GUIs

Can we script guis?  Not as clear.

Acme examples show one way:
```

```
        turn events into file (pipe) to read.

Tcl/tk is close too.

Eventually everyone turns to C.

-- others
Honorable Mentions

Scheme

Lisp

AutoCAD

Viaweb RTML

-- c
"Real" programming languages vs. Scripts

Why does everyone eventually rewrite scripts in C?
        (aka C++, C#, any compiled language)

What do you need C for now?

How could you make it accessible to a script language?

-- /home/rsc/bin/Slide
#!/usr/local/plan9/bin/rc

echo name `{pwd}^/$1 | 9p write acme/$winid/ctl
echo clean | 9p write acme/$winid/ctl
echo get | 9p write acme/$winid/ctl

-- /home/rsc/bin/Slide-
#!/usr/local/plan9/bin/rc

name=$%
current=`{basename $name}
currentx=`{9 grep -n '^'$current'([   ]|$)' index | sed 's/:.*//'}

pagex=`{echo $currentx - 1 | hoc}
if(~ $pagex 0){
        echo no such page
        exit 0
}
page=`{sed -n $pagex^p index | awk '{print $1}'}
if(~ $#page 0){
        echo no such page
        exit 0
}

Slide $page
-- /home/rsc/bin/Slide+
#!/usr/local/plan9/bin/rc

name=$%
```

```
current=`{basename $name}
currentx=`{9 grep -n '^'$current'([   ]|$)' index | sed 's/:.*//'}

pagex=`{echo $currentx + 1 | hoc}
page=`{sed -n $pagex^p index | awk '{print $1}'}
if(~ $#page 0){
        echo no such page
        exit 0
}

Slide $page
-- /usr/local/plan9/bin/adict
#!/usr/local/plan9/bin/rc

. 9.rc
. $PLAN9/lib/acme.rc

fn event {
        # $1 - c1 origin of event
        # $2 - c2 type of action
        # $3 - q0 beginning of selection
        # $4 - q1 end of selection
        # $5 - eq0 beginning of expanded selection
        # $6 - eq1 end of expanded selection
        # $7 - flag
        # $8 - nr number of runes in $7
        # $9 - text
        # $10 - chorded argument
        # $11 - origin of chorded argument

        switch($1$2){
        case E* # write to body or tag
        case F* # generated by ourselves; ignore
        case K* # type away we do not care
        case Mi # mouse: text inserted in tag
        case MI # mouse: text inserted in body
        case Md # mouse: text deleted from tag
        case MD # mouse: text deleted from body

        case Mx MX      # button 2 in tag or body
                winwriteevent $*

        case Ml ML      # button 3 in tag or body
                {
                        if(~ $dict NONE)
                                dictwin /adict/$9/ $9
                        if not
                                dictwin /adict/$dict/$9 $dict $9
                } &
        }
}

fn dictwin {
        newwindow
        winname $1
        switch($#*){
        case 1
```

```
                dict -d '?' >[2=1] | sed 1d | winwrite body
        case 2
                dict=$2
        case 3
                dict=$2
                dict -d $dict $3 >[2=1] | winwrite body
        }
        winctl clean
        wineventloop
}

dict=NONE
if(~ $1 -d){
        shift
        dict=$2
        shift
}
if(~ $1 -d*){
        dict=`{echo $1 | sed 's/-d//'}
        shift
}
if(~ $1 -*){
        echo 'usage: adict [-d dict] [word...]' >[1=2]
        exit usage
}

switch($#*){
case 0
        if(~ $dict NONE)
                dictwin /adict/
        if not
                dictwin /adict/$dict/ $dict
case *
        if(~ $dict NONE){
                dict=`{dict -d'?' | 9 sed -n 's/^   ([^\[     ]+).*/\1/p' |
sed 1q}
                if(~ $#dict 0){
                        echo 'no dictionaries present on this system' >[1=2]
                        exit nodict
                }
        }
        for(i)
                dictwin /adict/$dict/$i $dict $i
}

-- /usr/local/plan9/lib/acme.rc
fn newwindow {
        winctl=`{9p read acme/new/ctl}
        winid=$winctl(1)
        winctl noscroll
}

fn winctl {
        echo $* | 9p write acme/acme/$winid/ctl
}

fn winread {
```

```
        9p read acme/acme/$winid/$1
}

fn winwrite {
        9p write acme/acme/$winid/$1
}

fn windump {
        if(! ~ $1 - '')
                winctl dumpdir $1
        if(! ~ $2 - '')
                winctl dump $2
}

fn winname {
        winctl name $1
}

fn winwriteevent {
        echo $1$2$3 $4 | winwrite event
}

fn windel {
        if(~ $1 sure)
                winctl delete
        if not
                winctl del
}

fn wineventloop {
        . <{winread event >[2]/dev/null | acmeevent}
}
-- /home/rsc/plan9/rc/bin/fedex
#!/bin/rc

if(! ~ $#* 1) {
        echo usage: fedex 123456789012 >[1=2]
        exit usage
}

rfork e

fn bgrep{
pattern=`{echo $1 | sed 's;/;\\&;'}
shift

@{ echo 'X {
$
a

.
}
X ,x/(.+\n)+\n/ g/'$pattern'/p' |
sam -d $* >[2]/dev/null
}
}
```

```
fn awk2 {
	awk 'NR%2==1 { a=$0; }
		NR%2==0 { b=$0; printf("%-30s %s\n", a, b); }
	' $*
}

fn awk3 {
	awk '{line[NR] = $0}
	END{
		i = 4;
		while(i < NR){
			what=line[i++];
			when=line[i];
			comment="";
			if(!(when ~ /..\/..\/.... ..:../)){
				# out of sync
				printf("%s\n", what);
				continue;
			}
			i++;
			if(!(line[i+1] ~ /..\/..\/.... ..:../) &&
				(i+2 > NR || line[i+2] ~ /..\/..\/....
..:../)){
				what = what ", " line[i++];
			}
			printf("%s  %s\n", when, what);
		}
	}' $*
}

# hget 'http://www.fedex.com/cgi-
bin/track_it?airbill_list='$1'&kurrent_airbill='$1'&language=english&cntry_co
de=us&state=0' |
hget 'http://www.fedex.com/cgi-
bin/tracking?action=track&language=english&cntry_code=us&initial=x&mps=y&trac
knumbers='$1 |
	htmlfmt >/tmp/fedex.$pid
sed -n '/Tracking number/,/^$/p' /tmp/fedex.$pid | awk2
echo
sed -n '/Reference number/,/^$/p' /tmp/fedex.$pid | awk2
echo
sed -n '/Date.time/,/^$/p' /tmp/fedex.$pid | sed 1,4d | fmt -l 4000 | sed 's/
[A-Z][A-Z] /&\n/g'
rm /tmp/fedex.$pid
-- /home/rsc/src/webscript/a3
#!./o.webscript

load "http://www.ups.com/WebTracking/track?loc=en_US"
find textbox "InquiryNumber1"
input "1z30557w0340175623"
find next checkbox
input "yes"
find prev form
submit
if(find "Delivery Information"){
	find outer table
	print
```

```
}else if(find "One or more"){
        print
}else{
        print "Unexpected results."
        find page
        print
}
-- /home/rsc/src/webscript/a2
#load "http://apc-reset/outlets.htm"
load "apc.html"
print
print "\n=============\n"
find "yoshimi"
find outer row
find next select
input "Immediate Reboot"
submit
print
-- /usr/local/plan9/acid/port
// portable acid for all architectures

defn pfl(addr)
{
        print(pcfile(addr), ":", pcline(addr), "\n");
}

defn
notestk(addr)
{
        local pc, sp;
        complex Ureg addr;

        pc = addr.pc\X;
        sp = addr.sp\X;

        print("Note pc:", pc, " sp:", sp, " ", fmt(pc, 'a'), " ");
        pfl(pc);
        _stk({"PC", pc, "SP", sp, linkreg(addr)}, 1);
}

defn
notelstk(addr)
{
        local pc, sp;
        complex Ureg addr;

        pc = addr.pc\X;
        sp = addr.sp\X;

        print("Note pc:", pc, " sp:", sp, " ", fmt(pc, 'a'), " ");
        pfl(pc);
        _stk({"PC", pc, "SP", sp, linkreg(addr)}, 1);
}

defn params(param)
{
        while param do {
```

```
                sym = head param;
                print(sym[0], "=", itoa(sym[1], "%#ux"));
                param = tail param;
                if param then
                        print (",");
        }
}

stkprefix = "";
stkignore = {};
stkend = 0;

defn locals(l)
{
        local sym;

        while l do {
                sym = head l;
                print(stkprefix, "\t", sym[0], "=", itoa(sym[1], "%#ux"),
"\n");
                l = tail l;
        }
}

defn _stkign(frame)
{
        local file;

        file = pcfile(frame[0]);
        s = stkignore;
        while s do {
                if regexp(head s, file) then
                        return 1;
                s = tail s;
        }
        return 0;
}

// print a stack trace
//
// in a run of leading frames in files matched by regexps in stkignore,
// only print the last one.
defn _stk(regs, dolocals)
{
        local stk, frame, pc, fn, done, callerpc, paramlist, locallist;

        stk = strace(regs);
        if stkignore then {
                while stk && tail stk && _stkign(head tail stk) do
                        stk = tail stk;
        }

        callerpc = 0;
        done = 0;
        while stk && !done do {
                frame = head stk;
                stk = tail stk;
```

```
                fn = frame[0];
                pc = frame[1];
                callerpc = frame[2];
                paramlist = frame[3];
                locallist = frame[4];

                print(stkprefix, fmt(fn, 'a'), "(");
                params(paramlist);
                print(")");
                if pc != fn then
                        print("+", itoa(pc-fn, "%#ux"));
                print(" ");
                pfl(pc);
                if dolocals then
                        locals(locallist);
                if fn == var("threadmain") || fn == var("p9main") then
                        done=1;
                if fn == var("threadstart") || fn == var("scheduler") then
                        done=1;
                if callerpc == 0 then
                        done=1;
        }
        if callerpc && !done then {
                print(stkprefix, fmt(callerpc, 'a'), " ");
                pfl(callerpc);
        }
}

defn findsrc(file)
{
        local lst, src;

        if file[0] == '/' then {
                src = file(file);
                if src != {} then {
                        srcfiles = append srcfiles, file;
                        srctext = append srctext, src;
                        return src;
                }
                return {};
        }

        lst = srcpath;
        while head lst do {
                src = file(head lst+file);
                if src != {} then {
                        srcfiles = append srcfiles, file;
                        srctext = append srctext, src;
                        return src;
                }
                lst = tail lst;
        }
}

defn line(addr)
{
        local src, file;
```

```
	file = pcfile(addr);
	src = match(file, srcfiles);

	if src >= 0 then
		src = srctext[src];
	else
		src = findsrc(file);

	if src == {} then {
		print("no source for ", file, "\n");
		return {};
	}
	line = pcline(addr)-1;
	print(file, ":", src[line], "\n");
}

defn addsrcdir(dir)
{
	dir = dir+"/";

	if match(dir, srcpath) >= 0 then {
		print("already in srcpath\n");
		return {};
	}

	srcpath = {dir}+srcpath;
}

defn source()
{
	local l;

	l = srcpath;
	while l do {
		print(head l, "\n");
		l = tail l;
	}
	l = srcfiles;

	while l do {
		print("\t", head l, "\n");
		l = tail l;
	}
}

defn Bsrc(addr)
{
	local lst;

	lst = srcpath;
	file = pcfile(addr);
	if file[0] == '/' && access(file) then {
		rc("B "+file+":"+itoa(pcline(addr)));
		return {};
	}
	while head lst do {
```

```
                name = head lst+file;
                if access(name) then {
                        rc("B "+name+":"+itoa(pcline(addr)));
                        return {};
                }
                lst = tail lst;
        }
        print("no source for ", file, "\n");
}

defn srcline(addr)
{
        local text, cline, line, file, src;
        file = pcfile(addr);
        src = match(file,srcfiles);
        if (src>=0) then
                src = srctext[src];
        else
                src = findsrc(file);
        if (src=={}) then
        {
                return "(no source)";
        }
        return src[pcline(addr)-1];
}

defn src(addr)
{
        local src, file, line, cline, text;

        file = pcfile(addr);
        src = match(file, srcfiles);

        if src >= 0 then
                src = srctext[src];
        else
                src = findsrc(file);

        if src == {} then {
                print("no source for ", file, "\n");
                return {};
        }

        cline = pcline(addr)-1;
        print(file, ":", cline+1, "\n");
        line = cline-5;
        loop 0,10 do {
                if line >= 0 then {
                        if line == cline then
                                print(">");
                        else
                                print(" ");
                        text = src[line];
                        if text == {} then
                                return {};
                        print(line+1, "\t", text, "\n");
                }
```

```
                line = line+1;
        }
}

defn step()                                     // single step the process
{
        local lst, lpl, addr, bput;

        bput = 0;
        if match(*PC, bplist) >= 0 then {       // Sitting on a breakpoint
                bput = fmt(*PC, bpfmt);
                *bput = @bput;
        }

        lst = follow(*PC);

        lpl = lst;
        while lpl do {                          // place break points
                *(head lpl) = bpinst;
                lpl = tail lpl;
        }

        startstop(pid);                         // do the step

        while lst do {                          // remove the breakpoints
                addr = fmt(head lst, bpfmt);
                *addr = @addr;
                lst = tail lst;
        }
        if bput != 0 then
                *bput = bpinst;
}

defn bpset(addr)                                // set a breakpoint
{
        if status(pid) != "Stopped" then {
                print("Waiting...\n");
                stop(pid);
        }
        if match(addr, bplist) >= 0 then
                print("breakpoint already set at ", fmt(addr, 'a'), "\n");
        else {
                *fmt(addr, bpfmt) = bpinst;
                bplist = append bplist, addr;
        }
}

defn bptab()                                    // print a table of breakpoints
{
        local lst, addr;

        lst = bplist;
        while lst do {
                addr = head lst;
                print("\t", fmt(addr, 'X'), " ", fmt(addr, 'a'), "  ",
fmt(addr, 'i'), "\n");
                lst = tail lst;
```

```
        }
}

defn bpdel(addr)                                // delete a breakpoint
{
        local n, pc, nbplist;

        if addr == 0 then {
                while bplist do {
                        pc = head bplist;
                        pc = fmt(pc, bpfmt);
                        *pc = @pc;
                        bplist = tail bplist;
                }
                return {};
        }

        n = match(addr, bplist);
        if n < 0  then {
                print("no breakpoint at ", fmt(addr, 'a'), "\n");
                return {};
        }

        addr = fmt(addr, bpfmt);
        *addr = @addr;

        nbplist = {};                           // delete from list
        while bplist do {
                pc = head bplist;
                if pc != addr then
                        nbplist = append nbplist, pc;
                bplist = tail bplist;
        }
        bplist = nbplist;                       // delete from memory
}

defn cont()                                     // continue execution
{
        local addr;

        addr = fmt(*PC, bpfmt);
        if match(addr, bplist) >= 0 then {      // Sitting on a breakpoint
                *addr = @addr;
                step();                         // Step over
                *addr = bpinst;
        }
        startstop(pid);                         // Run
}

defn stopped(pid)               // called from acid when a process changes
state
{
        pfixstop(pid);
        pstop(pid);             // stub so this is easy to replace
}

defn procs()                    // print status of processes
```

```
{
	local c, lst, cpid;

	cpid = pid;
	lst = proclist;
	while lst do {
		np = head lst;
		setproc(np);
		if np == cpid then
			c = '>';
		else
			c = ' ';
		print(fmt(c, 'c'), np, ": ", status(np), " at ", fmt(*PC,
'a'), " setproc(", np, ")\n");
		lst = tail lst;
	}
	pid = cpid;
	if pid != 0 then
		setproc(pid);
}

_asmlines = 30;

defn asm(addr)
{
	local bound;

	bound = fnbound(addr);

	addr = fmt(addr, 'i');
	loop 1,_asmlines do {
		print(fmt(addr, 'a'), " ", fmt(addr, 'X'));
		print("\t", @addr++, "\n");
		if bound != {} && addr > bound[1] then {
			lasmaddr = addr;
			return {};
		}
	}
	lasmaddr = addr;
}

defn casm()
{
	asm(lasmaddr);
}

defn xasm(addr)
{
	local bound;

	bound = fnbound(addr);

	addr = fmt(addr, 'i');
	loop 1,_asmlines do {
		print(fmt(addr, 'a'), " ", fmt(addr, 'X'));
		print("\t", *addr++, "\n");
		if bound != {} && addr > bound[1] then {
```

```
                        lasmaddr = addr;
                        return {};
                }
        }
        lasmaddr = addr;
}

defn xcasm()
{
        xasm(lasmaddr);
}

defn win()
{
        local npid, estr;

        bplist = {};
        notes = {};

        estr = "/sys/lib/acid/window '0 0 600 400' "+textfile;
        if progargs != "" then
                estr = estr+" "+progargs;

        npid = rc(estr);
        npid = atoi(npid);
        if npid == 0 then
                error("win failed to create process");

        setproc(npid);
        stopped(npid);
}

defn win2()
{
        local npid, estr;

        bplist = {};
        notes = {};

        estr = "/sys/lib/acid/transcript '0 0 600 400' '100 100 700 500'
"+textfile;
        if progargs != "" then
                estr = estr+" "+progargs;

        npid = rc(estr);
        npid = atoi(npid);
        if npid == 0 then
                error("win failed to create process");

        setproc(npid);
        stopped(npid);
}

printstopped = 1;
defn new()
{
        local a;
```

```
	bplist = {};
	newproc(progargs);
	a = var("p9main");
	if a == {} then
		a = var("main");
	if a == {} then
		return {};
	bpset(a);
	while *PC != a do
		cont();
	bpdel(a);
}

defn stmnt()			// step one statement
{
	local line;

	line = pcline(*PC);
	while 1 do {
		step();
		if line != pcline(*PC) then {
			src(*PC);
			return {};
		}
	}
}

defn func()			// step until we leave the current function
{
	local bound, end, start, pc;

	bound = fnbound(*PC);
	if bound == {} then {
		print("cannot locate text symbol\n");
		return {};
	}

	pc = *PC;
	start = bound[0];
	end = bound[1];
	while pc >= start && pc < end do {
		step();
		pc = *PC;
	}
}

defn next()
{
	local sp, bound, pc;

	sp = *SP;
	bound = fnbound(*PC);
	if bound == {} then {
		print("cannot locate text symbol\n");
		return {};
	}
```

```
        stmnt();
        pc = *PC;
        if pc >= bound[0] && pc < bound[1] then
                return {};

        while (pc < bound[0] || pc > bound[1]) && sp >= *SP do {
                step();
                pc = *PC;
        }
        src(*PC);
}

defn maps()
{
        local m, mm;

        m = map();
        while m != {} do {
                mm = head m;
                m = tail m;
                print(mm[2]\X, " ", mm[3]\X, " ", mm[4]\X, " ", mm[0], " ",
mm[1], "\n");
        }
}

defn dump(addr, n, fmt)
{
        loop 0, n do {
                print(fmt(addr, 'X'), ": ");
                addr = mem(addr, fmt);
        }
}

defn mem(addr, fmt)
{

        local i, c, n;

        i = 0;
        while fmt[i] != 0 do {
                c = fmt[i];
                n = 0;
                while '0' <= fmt[i] && fmt[i] <= '9' do {
                        n = 10*n + fmt[i]-'0';
                        i = i+1;
                }
                if n <= 0 then n = 1;
                addr = fmt(addr, fmt[i]);
                while n > 0 do {
                        print(*addr++, " ");
                        n = n-1;
                }
                i = i+1;
        }
        print("\n");
        return addr;
}
```

```
defn symbols(pattern)
{
	local l, s;

	l = symbols;
	while l do {
		s = head l;
		if regexp(pattern, s[0]) then
			print(s[0], "\t", s[1], "\t", s[2], "\t", s[3], "\n");
		l = tail l;
	}
}

defn havesymbol(name)
{
	local l, s;

	l = symbols;
	while l do {
		s = head l;
		l = tail l;
		if s[0] == name then
			return 1;
	}
	return 0;
}

defn spsrch(len)
{
	local addr, a, s, e;

	addr = *SP;
	s = origin & 0x7fffffff;
	e = etext & 0x7fffffff;
	loop 1, len do {
		a = *addr++;
		c = a & 0x7fffffff;
		if c > s && c < e then {
			print("src(", a, ")\n");
			pfl(a);
		}
	}
}

defn acidtypes()
{
	local syms;
	local l;

	l = textfile();
	if l != {} then {
		syms = "acidtypes";
		while l != {} do {
			syms = syms + " " + ((head l)[0]);
			l = tail l;
		}
```

```
                includepipe(syms);
        }
}

defn getregs()
{
        local regs, l;

        regs = {};
        l = registers;
        while l != {} do {
                regs = append regs, var(l[0]);
                l = tail l;
        }
        return regs;
}

defn setregs(regs)
{
        local l;

        l = registers;
        while l != {} do {
                var(l[0]) = regs[0];
                l = tail l;
                regs = tail regs;
        }
        return regs;
}

defn resetregs()
{
        local l;

        l = registers;
        while l != {} do {
                var(l[0]) = register(l[0]);
                l = tail l;
        }
}

defn clearregs()
{
        local l;

        l = registers;
        while l != {} do {
                var(l[0]) = refconst(~0);
                l = tail l;
        }
}

progargs="";
print(acidfile);

-- /usr/local/plan9/acid/386
// 386 support
```

```
defn acidinit()               // Called after all the init modules are loaded
{
        bplist = {};
        bpfmt = 'b';

        srcpath = {
                "./",
                "/sys/src/libc/port/",
                "/sys/src/libc/9sys/",
                "/sys/src/libc/386/"
        };

        srcfiles = {};                // list of loaded files
        srctext = {};                 // the text of the files
}

defn linkreg(addr)
{
        return {};
}

defn stk()                            // trace
{
        _stk({"PC", *PC, "SP", *SP}, 0);
}

defn lstk()                           // trace with locals
{
        _stk({"PC", *PC, "SP", *SP}, 1);
}

defn gpr()             // print general(hah hah!) purpose registers
{
        print("AX\t", *AX, " BX\t", *BX, " CX\t", *CX, " DX\t", *DX, "\n");
        print("DI\t", *DI, " SI\t", *SI, " BP\t", *BP, "\n");
}

defn spr()                            // print special processor registers
{
        local pc;
        local cause;

        pc = *PC;
        print("PC\t", pc, " ", fmt(pc, 'a'), "  ");
        pfl(pc);
        print("SP\t", *SP, " ECODE ", *ECODE, " EFLAG ", *EFLAGS, "\n");
        print("CS\t", *CS, " DS\t ", *DS, " SS\t", *SS, "\n");
        print("GS\t", *GS, " FS\t ", *FS, " ES\t", *ES, "\n");

        cause = *TRAP;
        print("TRAP\t", cause, " ", reason(cause), "\n");
}

defn regs()                           // print all registers
{
        spr();
```

```
	gpr();
}

defn mmregs()
{
	print("MM0\t", *MM0, " MM1\t", *MM1, "\n");
	print("MM2\t", *MM2, " MM3\t", *MM3, "\n");
	print("MM4\t", *MM4, " MM5\t", *MM5, "\n");
	print("MM6\t", *MM6, " MM7\t", *MM7, "\n");
}

defn pfixstop(pid)
{
	if *fmt(*PC-1, 'b') == 0xCC then {
		// Linux stops us after the breakpoint, not at it
		*PC = *PC-1;
	}
}


defn pstop(pid)
{
	local l;
	local pc;
	local why;

	pc = *PC;

	// FIgure out why we stopped.
	if *fmt(pc, 'b') == 0xCC then {
		why = "breakpoint";

		// fix up instruction for print; will put back later
		*pc = @pc;
	} else if *(pc-2\x) == 0x80CD then {
		pc = pc-2;
		why = "system call";
	} else
		why = "stopped";

	if printstopped then {
		print(pid,": ", why, "\t");
		print(fmt(pc, 'a'), "\t", *fmt(pc, 'i'), "\n");
	}

	if why == "breakpoint" then
		*fmt(pc, bpfmt) = bpinst;

	if printstopped && notes then {
		if notes[0] != "sys: breakpoint" then {
			print("Notes pending:\n");
			l = notes;
			while l do {
				print("\t", head l, "\n");
				l = tail l;
			}
		}
```

```
        }
}

aggr Ureg
{
        'U' 0 di;
        'U' 4 si;
        'U' 8 bp;
        'U' 12 nsp;
        'U' 16 bx;
        'U' 20 dx;
        'U' 24 cx;
        'U' 28 ax;
        'U' 32 gs;
        'U' 36 fs;
        'U' 40 es;
        'U' 44 ds;
        'U' 48 trap;
        'U' 52 ecode;
        'U' 56 pc;
        'U' 60 cs;
        'U' 64 flags;
        {
        'U' 68 usp;
        'U' 68 sp;
        };
        'U' 72 ss;
};

defn
Ureg(addr) {
        complex Ureg addr;
        print(" di      ", addr.di, "\n");
        print(" si      ", addr.si, "\n");
        print(" bp      ", addr.bp, "\n");
        print(" nsp     ", addr.nsp, "\n");
        print(" bx      ", addr.bx, "\n");
        print(" dx      ", addr.dx, "\n");
        print(" cx      ", addr.cx, "\n");
        print(" ax      ", addr.ax, "\n");
        print(" gs      ", addr.gs, "\n");
        print(" fs      ", addr.fs, "\n");
        print(" es      ", addr.es, "\n");
        print(" ds      ", addr.ds, "\n");
        print(" trap    ", addr.trap, "\n");
        print(" ecode   ", addr.ecode, "\n");
        print(" pc      ", addr.pc, "\n");
        print(" cs      ", addr.cs, "\n");
        print(" flags   ", addr.flags, "\n");
        print(" sp      ", addr.sp, "\n");
        print(" ss      ", addr.ss, "\n");
};
sizeofUreg = 76;

aggr Linkdebug
{
        'X' 0 version;
```

```
        'X' 4 map;
};

aggr Linkmap
{
        'X' 0 addr;
        'X' 4 name;
        'X' 8 dynsect;
        'X' 12 next;
        'X' 16 prev;
};

defn
linkdebug()
{
        local a;

        if !havesymbol("_DYNAMIC") then
                return 0;

        a = _DYNAMIC;
        while *a != 0 do {
                if *a == 21 then // 21 == DT_DEBUG
                        return *(a+4);
                a = a+8;
        }
        return 0;
}

defn
dynamicmap()
{
        if systype == "linux"  || systype == "freebsd" then {
                local r, m, n;

                r = linkdebug();
                if r then {
                        complex Linkdebug r;
                        m = r.map;
                        n = 0;
                        while m != 0 && n < 100 do {
                                complex Linkmap m;
                                if m.name && *(m.name\b) && access(*(m.name\s))
then
                                        print("textfile({\"", *(m.name\s), "\",
", m.addr\X, "});\n");
                                m = m.next;
                                n = n+1;
                        }
                }
        }
}

defn
acidmap()
{
//      dynamicmap();
```

```
        acidtypes();
}

print(acidfile);
```