Programming Tools

Compiler

- Convert source code to object modules

  .o: external references not yet resolved

```
$ nm
                 U printf
0000000000000000 t
0000000000000000 d
0000000000000000 b
0000000000000000 r
0000000000000000 ?
0000000000000000 a *ABS*
0000000000000000 T c
0000000000000000 a const.c
0000000000000048 T main
```

Linker

- Combine .o and .so into single a.out executable module

  .so/.dll: dynamically loaded at run-time

- “man ld”

```
$ ldd a.out
linux-vdso.so.1 (0x00007fffbc501000)
libc.so.6 => /lib64/libc.so.6 (0x00007fd288370000)
/lib64/ld-linux-x86-64.so.2 (0x00007fd28872f000)
```

Creating a static library

- libsomething.a

  gcc –c *.c

  ar rlv libsomething.a *.o

  ranlib libsomething.a

- use library as:

  gcc –L/your/dir –lsomething

Creating a dynamic library

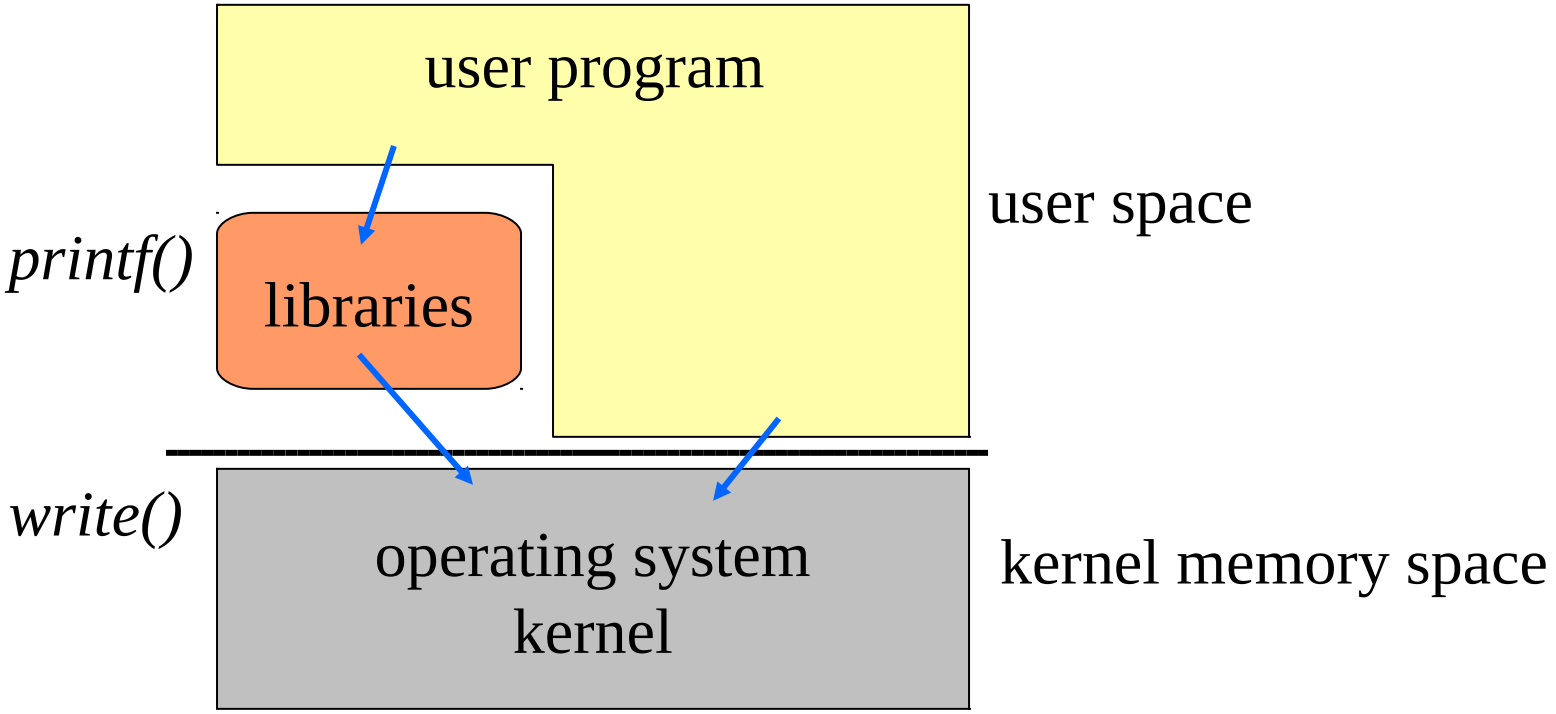
- Details differ for each platform

```
gcc –shared –fPIC –o libhelper.so *.o
```

- same usage as (-l*library*)

  `LD_LIBRARY_PATH`

Program tracing



Program tracing

- Simple debugging: find out what system calls a program is using
- `strace` on Linux
  - does not require access to source code

  -f: follow children

  -p: attach to existing process

  -T: include timestamps

strace

```
$ strace –t –T cat foo
14:26:59 open("foo", O_RDONLY|O_LARGEFILE) = 3 <0.000712>
14:26:59 fstat(3, {st_mode=S_IFREG|0644, st_size=6, ...}) = 0
   <0.000005>
14:26:59 brk(0x8057000)                 = 0x8057000 <0.000011>
14:26:59 read(3, "hello\n", 32768)      = 6 <0.000010>
14:26:59 write(1, "hello\n", 6hello
)         = 6 <0.000015>
14:26:59 read(3, "", 32768)             = 0 <0.000005>
14:26:59 close(3)                       = 0 <0.000010>
14:26:59 _exit(0)                       = ?
```

Memory utilization: top

- Show consumers of CPU and memory

```
load averages:  0.42,  0.22,  0.16                                    14:17:35
274 processes: 269 sleeping, 1 zombie, 3 stopped, 1 on cpu
CPU states: 81.3% idle,  5.2% user, 13.4% kernel,  0.1% iowait,  0.0% swap
Memory: 512M real, 98M free, 345M swap in use, 318M swap free

   PID USERNAME THR PRI NICE  SIZE   RES STATE    TIME    CPU COMMAND
   144 root       1  53    0 3384K 1728K sleep   33.3H  3.67% ypserv
 11011 hgs        1  48    0 2776K 2248K sleep    0:00  0.57% tcsh
 11040 hgs        1  55    0 1800K 1352K cpu/0    0:00  0.39% top
   281 root       1  58    0 4240K 2720K sleep  313:03  0.38% amd
 10933 kbutler    1  58    0   11M 8376K sleep    0:00  0.17% lisp
  1817 yjh9       1  58    0 8968K 7528K sleep    0:39  0.10% emacs
 13955 yjh9       1  58    0 8496K 7200K sleep    2:47  0.09% emacs
```

Debugging

- Interact with program while running
  - step-by-step execution
    - instruction
    - source line
    - procedure
  - inspect current state
    - call stack
    - global variables
    - local variables

Debugging

- Requires compiler support:
  - generate mapping from program counter to source line
  - needs symbol table for variable names

Debugging

```
$ gcc –g –o loop loop.c
$ gdb loop
(gdb) break main
(gdb) run foo
Starting program: src/test/loop

Breakpoint 1, main (argc=2, argv=0xffbef6ac) at
   loop.c:5
5          for (i = 0; i < 10; i++) {
```

gdb

```
(gdb) n
6 printf("i=%d\n", i);
(gdb) where
#0  loop (i=1) at loop.c:4
#1  0x105ec in main (argc=2, argv=0xffbef6a4) at loop.c:11
(gdb) p i
$1 = 0
(gdb) break 9
Breakpoint 2 at 0x105e4: file loop.c, line 9.
(gdb) cont
Continuing.
i=0
i=1
...
Breakpoint 2, main (argc=1, argv=0xffbef6ac) at loop.c:9
9  return 0;
```

gdb hints

- Make sure your source file is around (ar
  it doesn’t get changed)
- Does not work (well) across threads
- Can be used to debug *core dumps*:

```
$ gdb a.out core
#0  0x10604 in main (argc=1, argv=0xffbef6fc) at
    loop.c:14
*s = '\0';
(gdb) print i
$1 = 10
```

gdb - execution

run *arg*	run program
call *f(a,b)*	call function in program
step *N*	step *N* times into functions
next *N*	step *N* times over functions
up *N*	select stack frame that called current one
down *N*	select stack frame called by current one

gdb – break points

break *main.c:12*	set break point
break *foo*	set break at function
clear *main.c:12*	delete breakpoint
info break	show breakpoints
delete *1*	delete break point 1
display *x*	display variable at each step

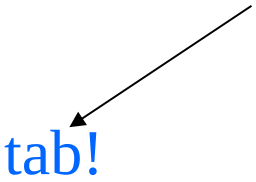
Building programs

- Programs consist of many modules
- Dependencies:
  - if one file changes, one or more others need to change
  - `.c` depends on `.h` -> re-compile
  - `.o` depends on `.c` -> re-compile
  - executable depends on `.o`'s -> link
  - library depends on .o -> archive
  - recursive!

make

- make maintains dependency graphs, based on modification times
- `Makefile` as default name
- `make [–f makefile] [target]`
- if node newer than child, remake child

```
target ...: dependency
    command
    command
```



make

```
all: hello clean
clean:
  rm –f *.o
helper.o: helper.c
  cc –c helper.c –I/home/include/mydir
hello.o: helper.c
  cc –c hello.c –O3 -DFLAG
OBJ = helper.o hello.o
hello: $(OBJ)
  $(CC) $(CFLAGS) $(LDFLAGS) –o $@ $(OBJ)
```

make variables

$@	name of current target
$?	list of dependencies newer than target
$<	name of dependency file
$*	base name of current target
$%	for libraries, the name of member

- implicit rules, e.g., a .c file into .o

```
.c.o:
$(CC) $(CFLAGS) $<
```

make environment

- Environment variables (`PATH`, `HOME`, `USER`, etc.) are available as `$(PATH)`, etc.
- Also passed to commands invoked
- Can create new variables (gmake):

```
export FOOBAR = foobar
```

profiling

- Execution profile:

```
int inner(int x) {
  static int sum;
  sum += x;
  return sum;
}

double outer(int y) {
  int i;
  double x = 1;
  double sum = 0;
  for (i = 0; i < 10000; i++) {
    x *= 2; sum += inner(i + y);
  }
  return sum;
}

int main(int argc, char *argv[])
{
  int i;
  for (i = 0; i < 1000; i++) {outer(i);}
  exit(0);
}
```

profiling

```
gcc –pg nested.c –o nested
```

- change function invocation to do logging (call `_mcount`)
- also, PC sampling (e.g., 100 times/second)
- generate a *call graph*

```
gprof nested gmon.out
```

gprof flat profile

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 59.50      2.88     2.88                             internal_mcount
 21.69      3.93     1.05     1000     1.05     1.92  outer
 17.15      4.76     0.83 10000000     0.00     0.00  inner
  0.83      4.80     0.04     1000     0.04     0.04  _libc_write
  0.83      4.84     0.04                             _mcount
  0.00      4.84     0.00     2000     0.00     0.00  _realbufend
  0.00      4.84     0.00     2000     0.00     0.00  ferror_unlocked
  0.00      4.84     0.00     1890     0.00     0.00  .mul
  0.00      4.84     0.00     1000     0.00     0.04  _doprnt
  0.00      4.84     0.00     1000     0.00     0.04  _xflsbuf
  0.00      4.84     0.00     1000     0.00     0.00  memchr
  0.00      4.84     0.00     1000     0.00     0.04  printf
```

gprof call graph

- Time spent in function and its children

```
index % time    self  children    called     name
                                                 <spontaneous>
[1]     60.0    2.88    0.00                 internal_mcount [1]
                0.00    0.00       1/3           atexit [15]
-----------------------------------------------
                1.05    0.87    1000/1000        main [3]
[2]     40.0    1.05    0.87    1000         outer [2]
                0.83    0.00 10000000/10000000     inner [5]
                0.00    0.04    1000/1000        printf [6]
-----------------------------------------------
                0.00    1.92       1/1           _start [4]
[3]     40.0    0.00    1.92       1         main [3]
                1.05    0.87    1000/1000        outer [2]
                0.00    0.00       1/1           exit [19]
-----------------------------------------------
                                                 <spontaneous>
[4]     40.0    0.00    1.92                 _start [4]
                0.00    1.92       1/1           main [3]
                0.00    0.00       2/3           atexit [15]
-----------------------------------------------
                0.83    0.00 10000000/10000000     outer [2]
[5]     17.3    0.83    0.00 10000000         inner [5]
```