Process Environment (Chap7-Chap9)

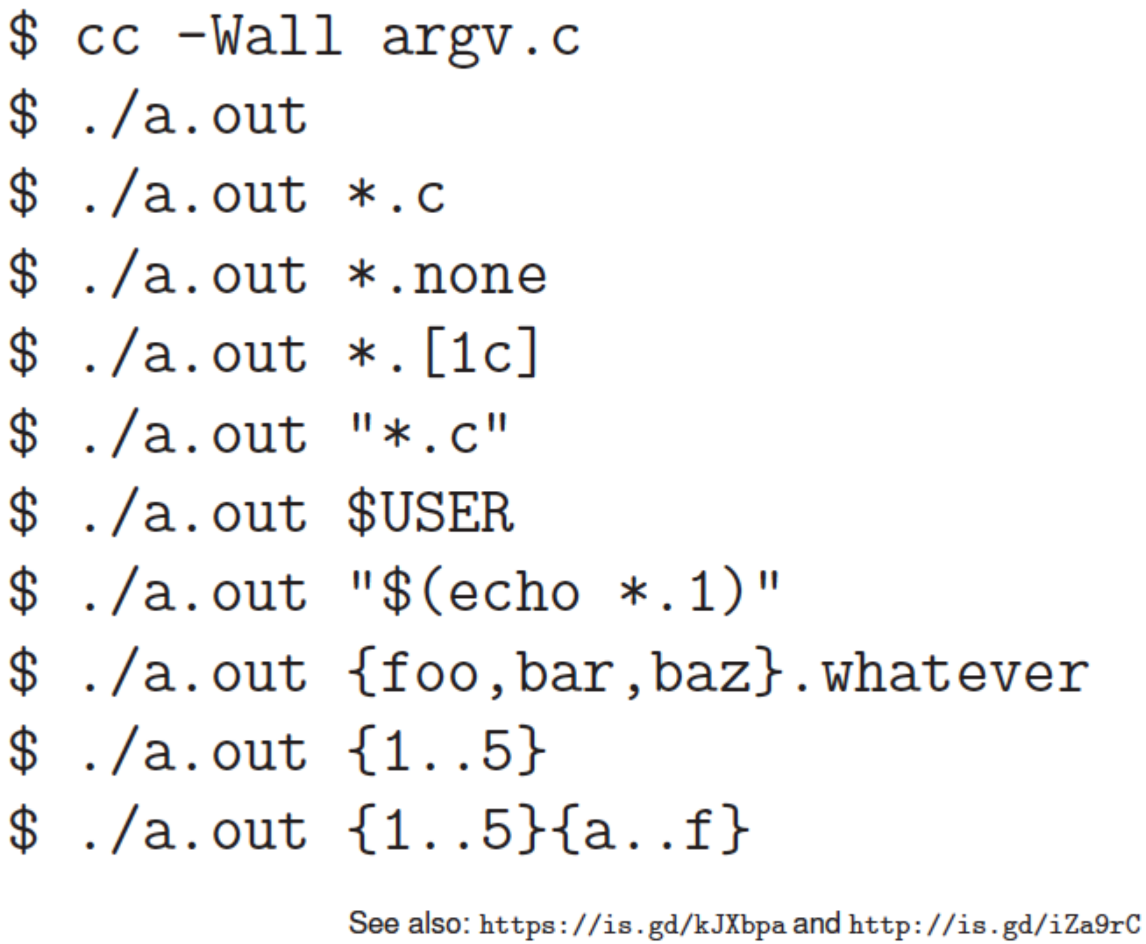
Hongik University

Eunsung Jung

Disclaimer: The slides are borrowed from many sources!

LAB: Shell Command-Line Processing

- Refer to Figure 7.4

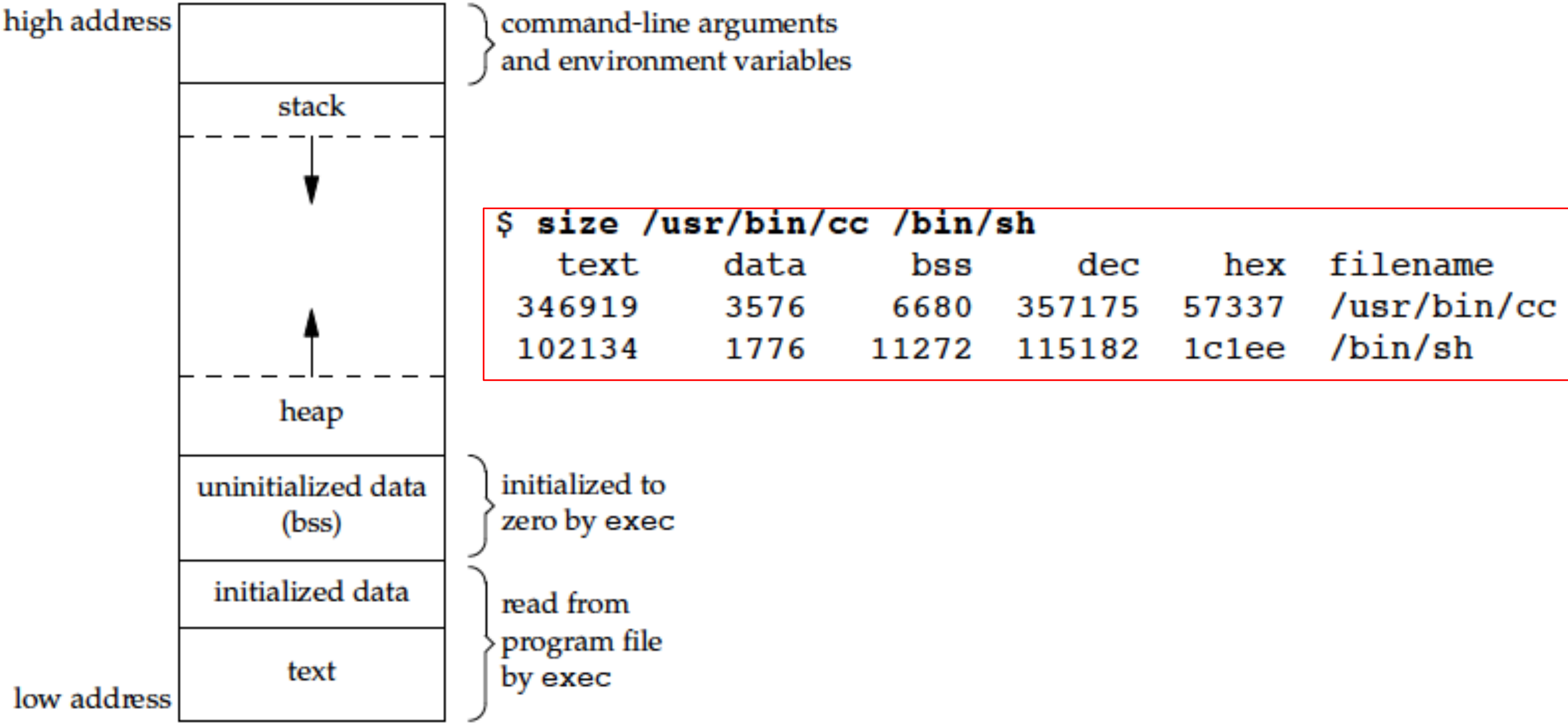


```
$ cc -Wall argv.c
$ ./a.out
$ ./a.out *.c
$ ./a.out *.none
$ ./a.out *.[1c]
$ ./a.out "*.c"
$ ./a.out $USER
$ ./a.out "$(echo *.1)"
$ ./a.out {foo,bar,baz}.whatever
$ ./a.out {1..5}
$ ./a.out {1..5}{a..f}
```

See also: `https://is.gd/kJXbpa` and `http://is.gd/iZa9rC`

Memory Layout of a C Program

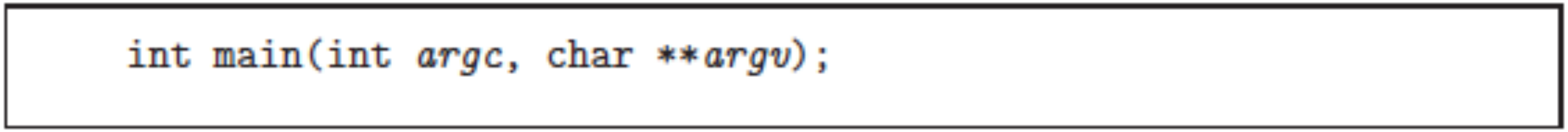
- The size (1) command reports the sizes (in bytes) of the text, data, and bss segments.


high address
stack
heap
uninitialized data (bss)
initialized data
text
low address
command-line arguments and environment variables
initialized to zero by exec
read from program file by exec

```
$ size /usr/bin/cc /bin/sh
   text     data      bss      dec     hex  filename
 346919     3576     6680   357175   57337  /usr/bin/cc
 102134     1776    11272   115182   1c1ee  /bin/sh
```

Figure 7.6 Typical memory arrangement

The main function

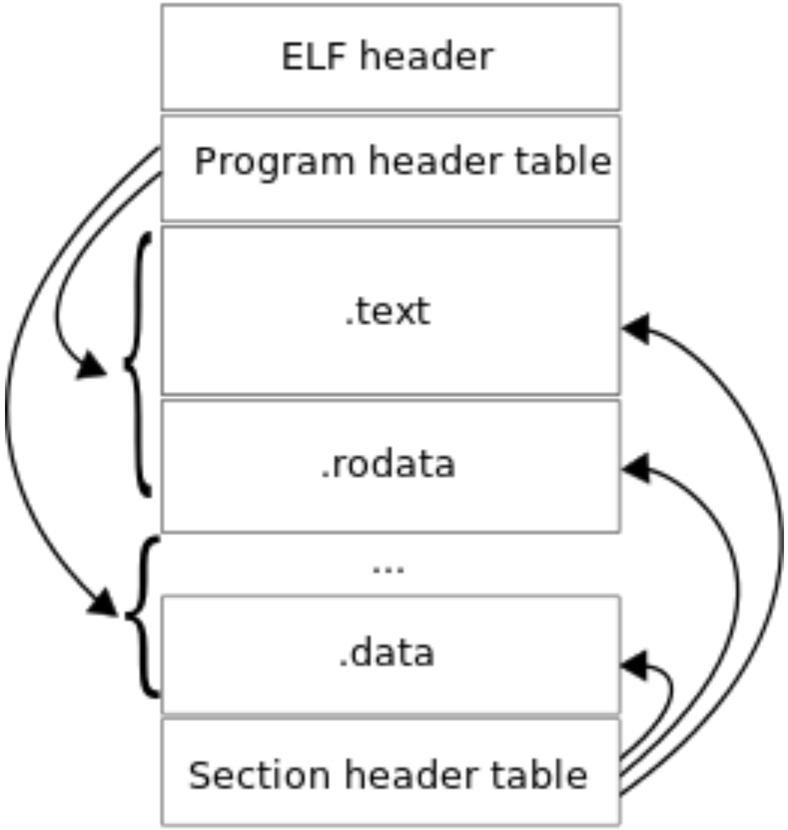


```
int main(int argc, char **argv);
```

- C program started by kernel (by one of the exec functions)
- Special startup routine called by kernel which sets up things for main (or whatever entry point is defined)
- argc is a count of the number of command line arguments (including the command itself)
- argv is an array of pointers to the arguments
- it is guaranteed by both ANSI C and POSIX.1 that argv[argc] == NULL

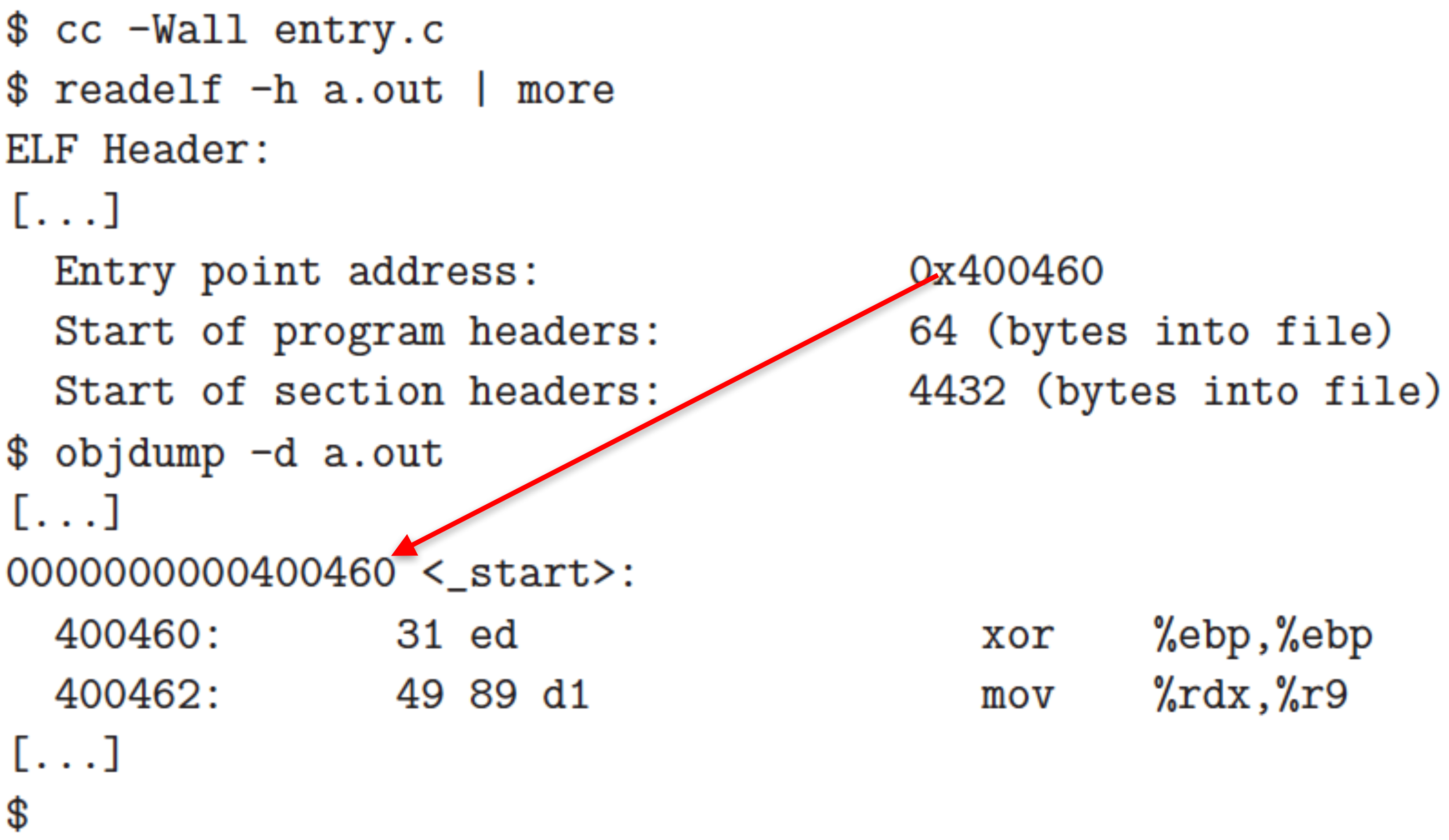
Process Creation

- **Executable and Linkable Format** (**ELF**, formerly named **Extensible Linking Format**)



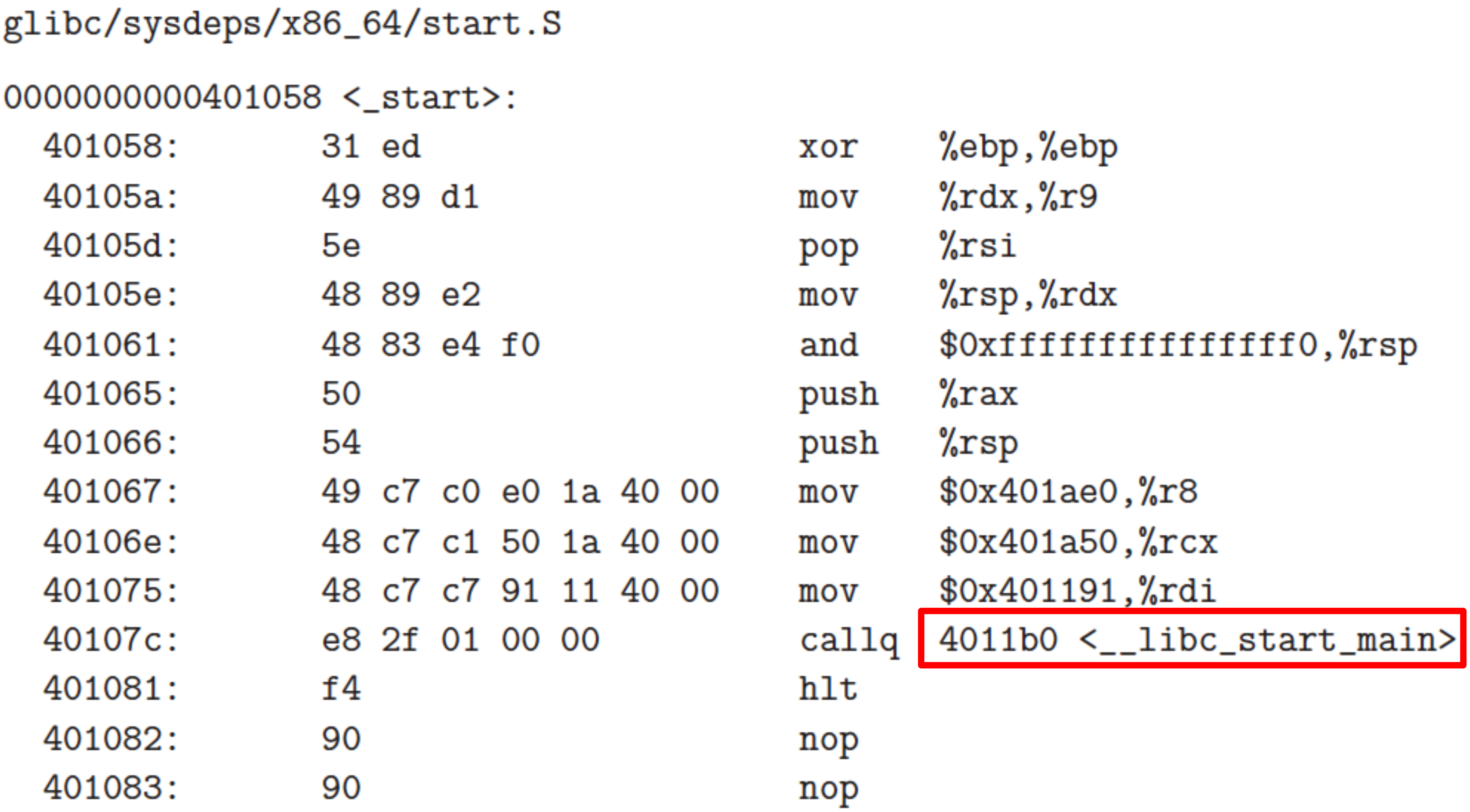
Process Creation

- On Linux:



```
$ cc -Wall entry.c
$ readelf -h a.out | more
ELF Header:
[...]
  Entry point address:                 0x400460
  Start of program headers:            64 (bytes into file)
  Start of section headers:            4432 (bytes into file)
$ objdump -d a.out
[...]
0000000000400460 <_start>:
  400460:       31 ed                     xor    %ebp,%ebp
  400462:       49 89 d1                  mov    %rdx,%r9
[...]
$
```

Process Creation



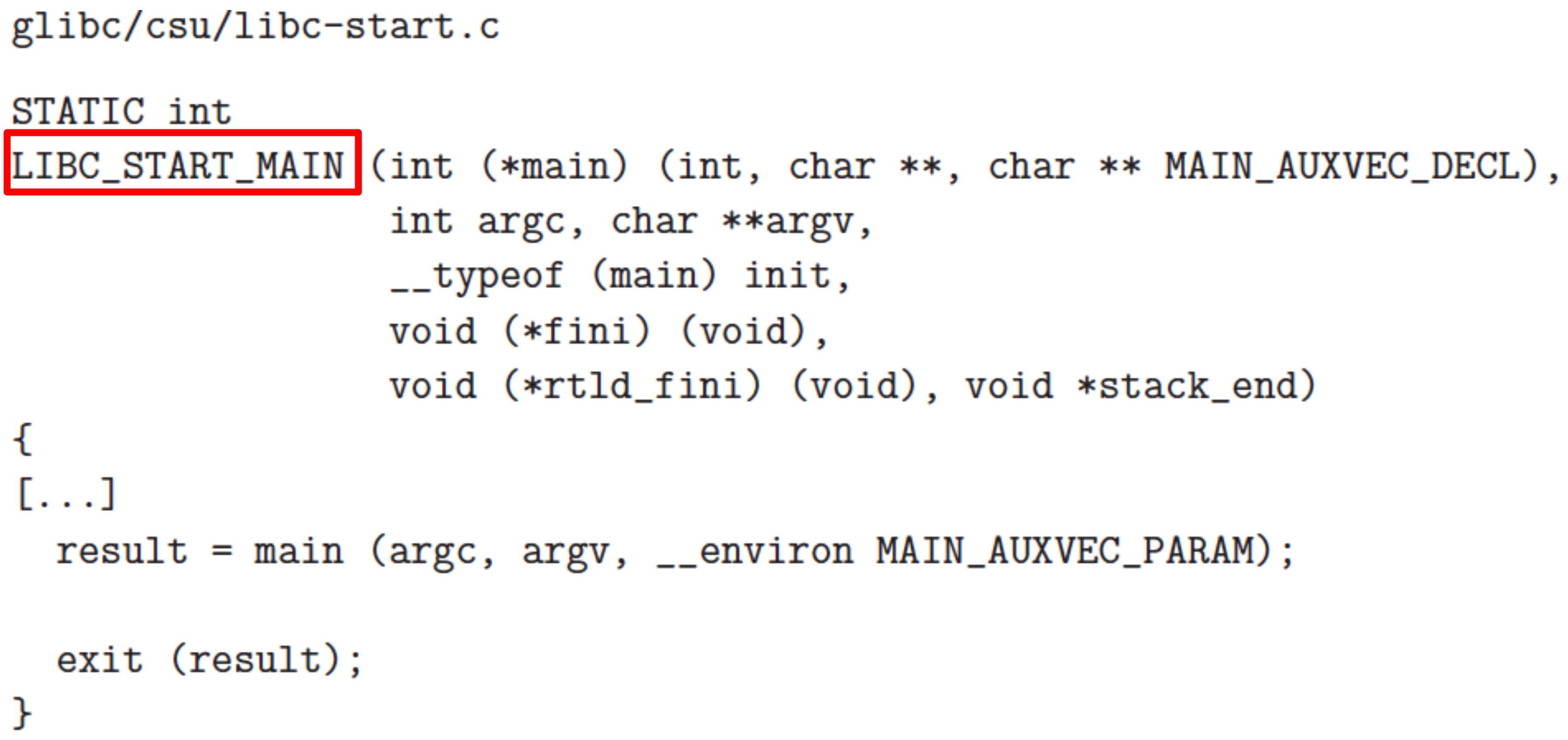
```
glibc/sysdeps/x86_64/start.S

0000000000401058 <_start>:
  401058:       31 ed                   xor    %ebp,%ebp
  40105a:       49 89 d1                mov    %rdx,%r9
  40105d:       5e                      pop    %rsi
  40105e:       48 89 e2                mov    %rsp,%rdx
  401061:       48 83 e4 f0             and    $0xfffffffffffffff0,%rsp
  401065:       50                      push   %rax
  401066:       54                      push   %rsp
  401067:       49 c7 c0 e0 1a 40 00    mov    $0x401ae0,%r8
  40106e:       48 c7 c1 50 1a 40 00    mov    $0x401a50,%rcx
  401075:       48 c7 c7 91 11 40 00    mov    $0x401191,%rdi
  40107c:       e8 2f 01 00 00          callq  4011b0 <__libc_start_main>
  401081:       f4                      hlt
  401082:       90                      nop
  401083:       90                      nop
```

Process Creating

- git clone git://sourceware.org/git/glibc.git
- cd glibc
- Find start.S!

- FYI, Kernel codes
  - wget https://www.kernel.org/pub/linux/kernel/v4.x/linux-4.12.12.tar.gz
  - tar xvf linux-4.12.12.tar.gz

Process Creation



```
glibc/csu/libc-start.c

STATIC int
LIBC_START_MAIN (int (*main) (int, char **, char ** MAIN_AUXVEC_DECL),
                 int argc, char **argv,
                 __typeof (main) init,
                 void (*fini) (void),
                 void (*rtld_fini) (void), void *stack_end)
{
[...]
  result = main (argc, argv, __environ MAIN_AUXVEC_PARAM);

  exit (result);
}
```

Process Termination

- There are 8 ways for a process to terminate.

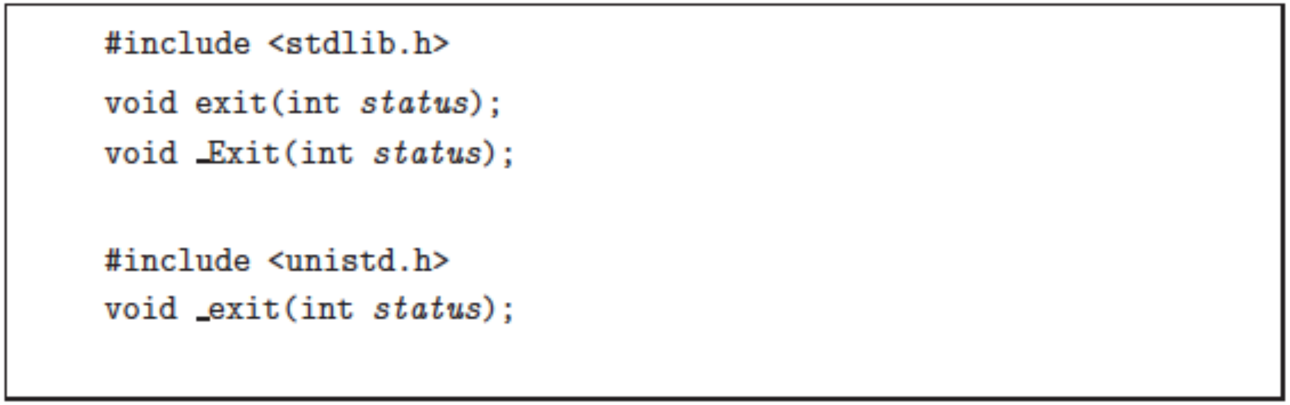
- Normal termination (5 ways)
  - return from main
  - calling exit
  - calling _exit (or _Exit)
  - return of last thread from its start routine
  - calling pthread_exit from last thread

Process Termination

- There are 8 ways for a process to terminate.

- Abnormal termination (3 ways)
  - calling abort
  - terminated by a signal
  - response of the last thread to a cancellation request

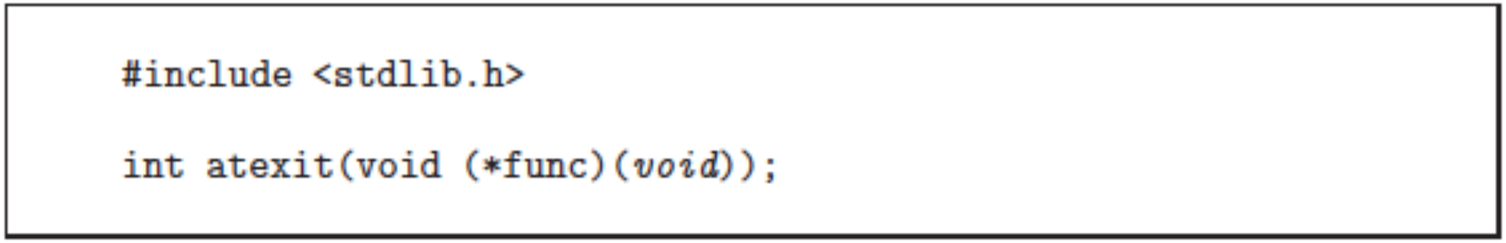
exit(3) and _exit(2)



```
#include <stdlib.h>

void exit(int status);
void _Exit(int status);


#include <unistd.h>
void _exit(int status);
```

- _exit and _Exit
  - return to the kernel immediately
  - _exit required by POSIX.1
  - _Exit required by ISO C99
  - synonymous on Unix
- exit does some cleanup and then returns
- both take integer argument, aka exit status

atexit(3)



```
#include <stdlib.h>

int atexit(void (*func)(void));
```

- Registers a function with a signature of void funcname(void) to be called at exit
- Functions invoked in reverse order of registration
- Same function can be registered more than once
- Extremely useful for cleaning up open files, freeing certain resources, etc.

Lifetime of a UNIX Process

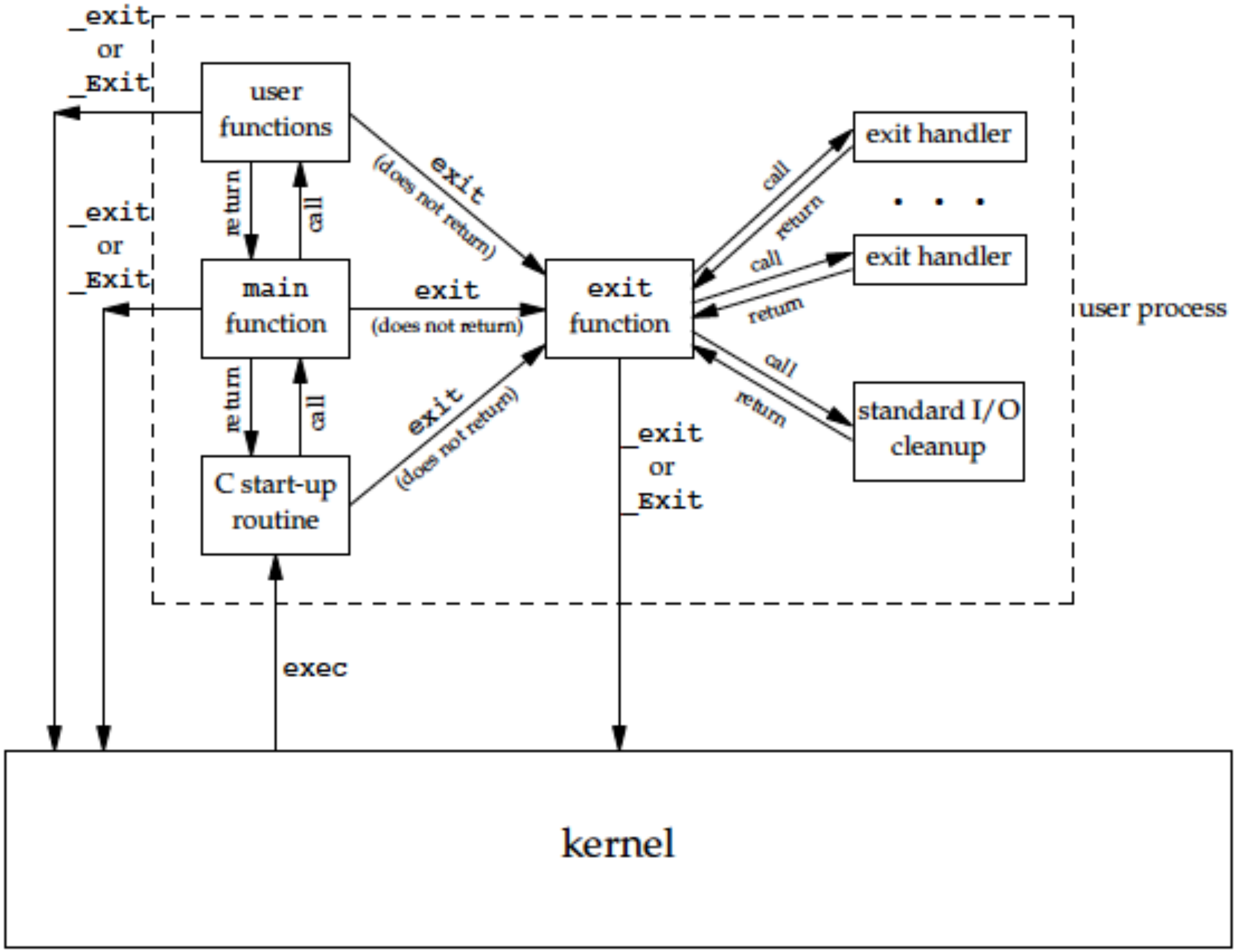
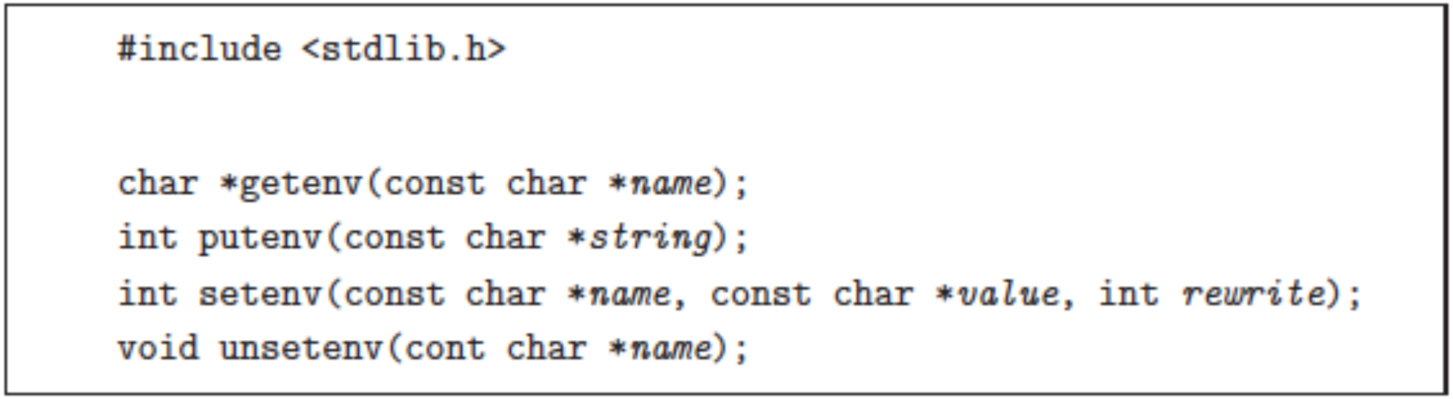


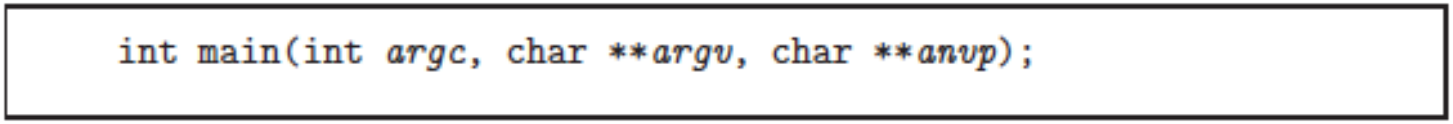
Figure 7.2 How a C program is started and how it terminates

Environment List

- Environment variables are stored in a global array of pointers:
  - extern char **environ;
  - The list is null terminated.
- These can also be accessed by:

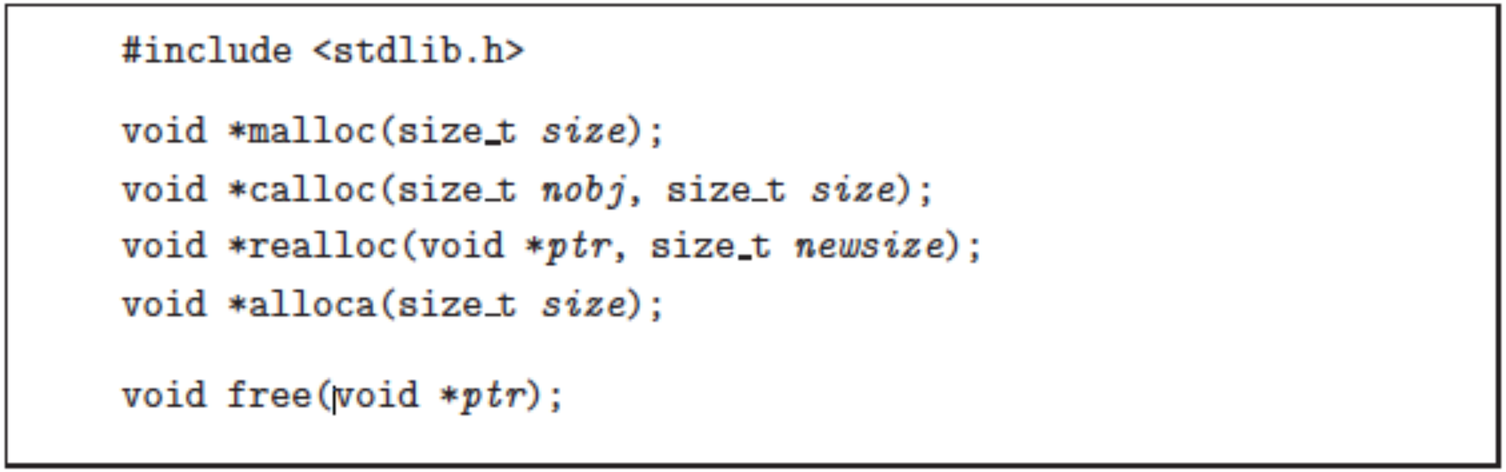


```
#include <stdlib.h>


char *getenv(const char *name);
int putenv(const char *string);
int setenv(const char *name, const char *value, int rewrite);
void unsetenv(cont char *name);
```



```
int main(int argc, char **argv, char **anvp);
```

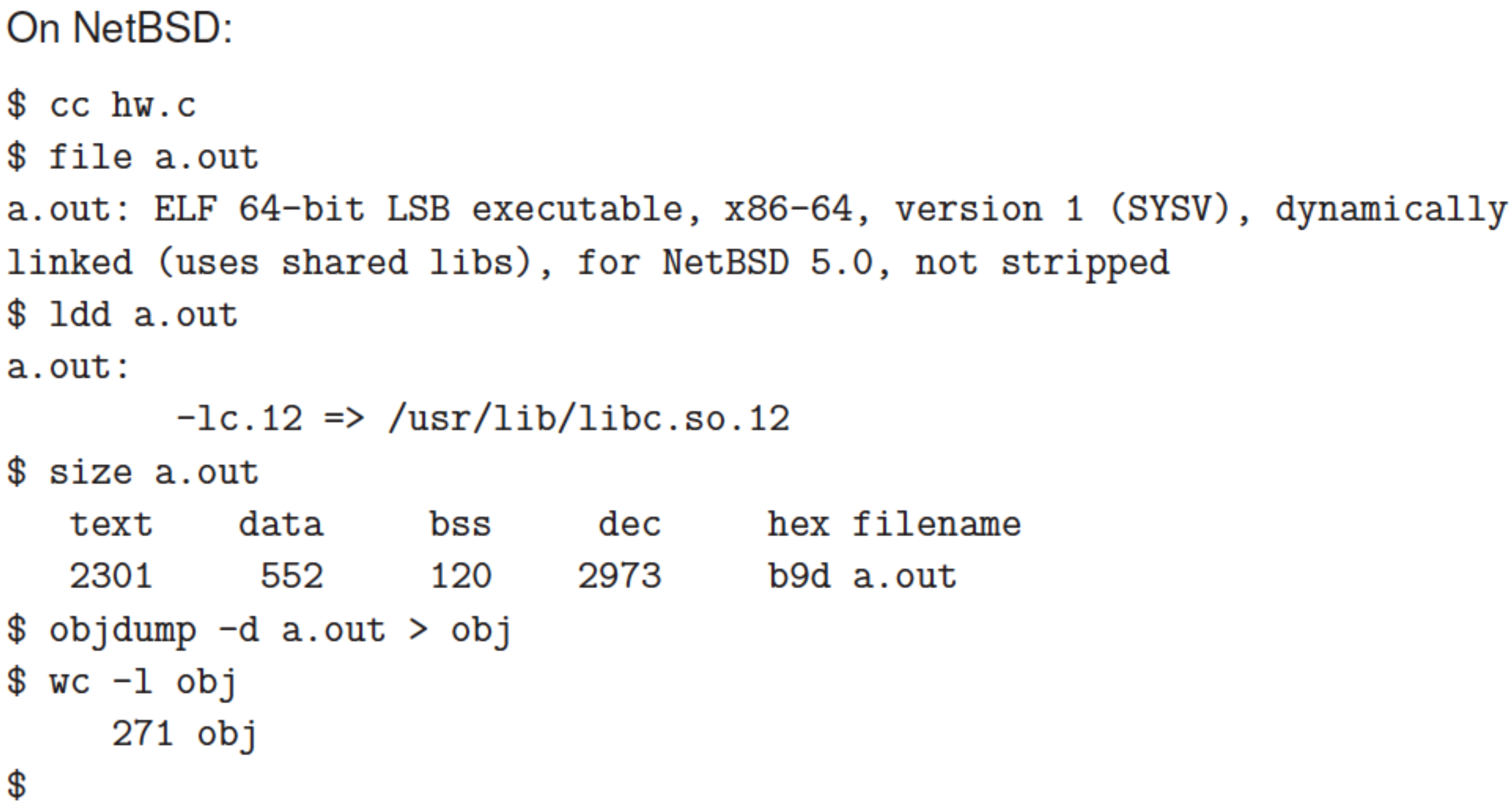
Memory Allocation



```
#include <stdlib.h>

void *malloc(size_t size);
void *calloc(size_t nobj, size_t size);
void *realloc(void *ptr, size_t newsize);
void *alloca(size_t size);

void free(void *ptr);
```

- malloc – initial value is indeterminate.
- calloc – initial value set to all zeros.
- realloc – changes size of previously allocated area. Initial value of any additional space is indeterminate.
- alloca – allocates memory on stack

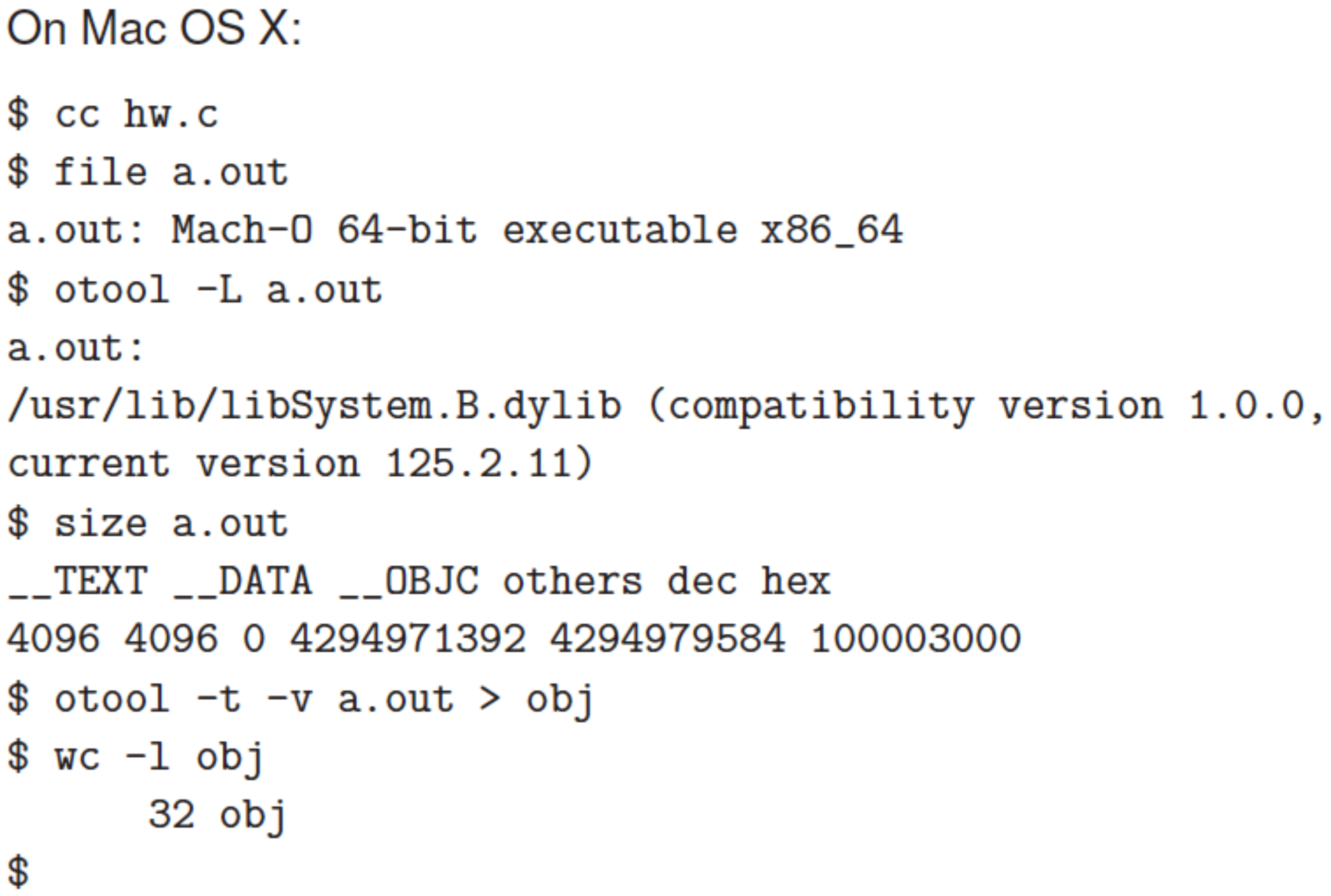
Memory Layout of a C Program



On NetBSD:

```
$ cc hw.c
$ file a.out
a.out: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically
linked (uses shared libs), for NetBSD 5.0, not stripped
$ ldd a.out
a.out:
        -lc.12 => /usr/lib/libc.so.12
$ size a.out
   text    data     bss     dec     hex filename
   2301     552     120    2973     b9d a.out
$ objdump -d a.out > obj
$ wc -l obj
     271 obj
$
```

Memory Layout of a C Program



On Mac OS X:

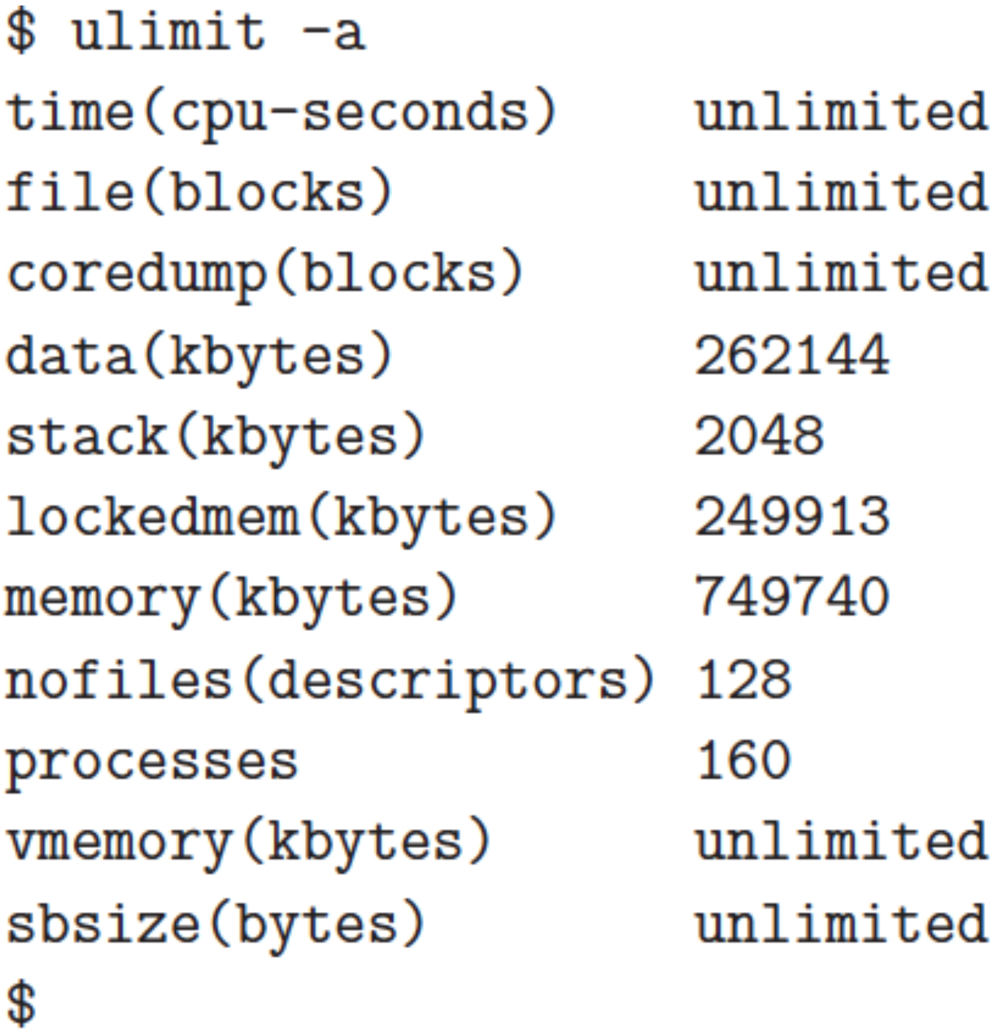
```
$ cc hw.c
$ file a.out
a.out: Mach-O 64-bit executable x86_64
$ otool -L a.out
a.out:
/usr/lib/libSystem.B.dylib (compatibility version 1.0.0,
current version 125.2.11)
$ size a.out
__TEXT __DATA __OBJC others dec hex
4096 4096 0 4294971392 4294979584 100003000
$ otool -t -v a.out > obj
$ wc -l obj
      32 obj
$
```

Memory Layout of a C Program

On Linux:

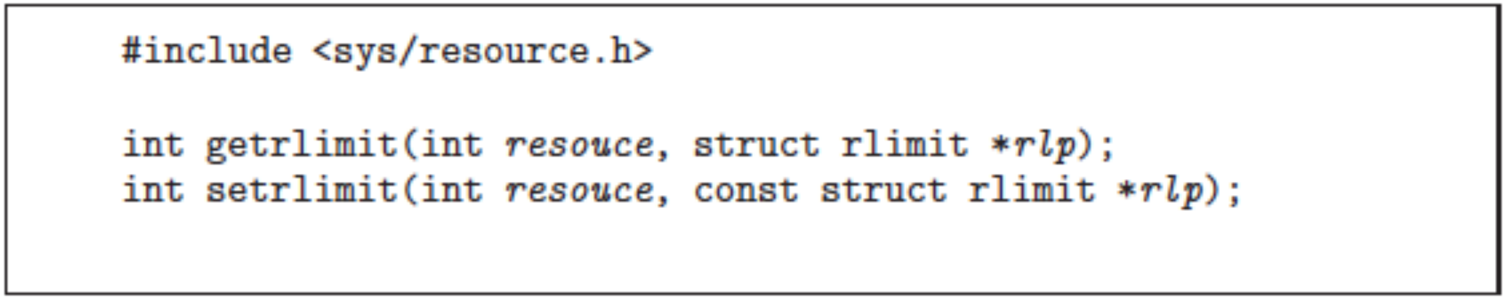
```
$ cc hw.c
$ file a.out
a.out: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV),
dynamically linked (uses shared libs), for GNU/Linux 2.6.15, not stripped
$ ldd a.out
linux-gate.so.1 =>  (0x00c66000)
libc.so.6 => /lib/tls/i686/cmov/libc.so.6 (0x006b4000)
/lib/ld-linux.so.2 (0x005fe000)
$ size a.out
   text    data     bss     dec     hex filename
    918     264       8    1190     4a6 a.out
$ objdump -d a.out >obj
$ wc -l obj
225 obj
$
```

Process limits



```
$ ulimit -a
time(cpu-seconds)    unlimited
file(blocks)         unlimited
coredump(blocks)     unlimited
data(kbytes)         262144
stack(kbytes)        2048
lockedmem(kbytes)    249913
memory(kbytes)       749740
nofiles(descriptors) 128
processes            160
vmemory(kbytes)      unlimited
sbsize(bytes)        unlimited
$
```

getrlimit(2) and setrlimit(2)



```
#include <sys/resource.h>

int getrlimit(int resouce, struct rlimit *rlp);
int setrlimit(int resouce, const struct rlimit *rlp);
```

- Changing resource limits follows these rules:
  - a soft limit can be changed by any process to a value less than or equal to its hard limit
  - any process can lower its hard limit greater than or equal to its soft limit
  - only superuser can raise hard limits
  - changes are per process only