Memory Layout  (Virtual address space of a C process)

STACK

System

env
argv
arge

auto variables for
main()

auto variables for
func()

available for
stack growth

malloc.o (lib*.so)
printf.o (lib*.so)

available for
heap growth

Heap
(malloc arena)

global variables

"...%d...

malloc.o (lib*.a)
printf.o (lib*.a)

malloc.o (lib*.so)
printf.o (lib*.so)

compiled code (.o, a.out)

main.o
func(72,73)

crt0.o (startup routine)

High memory

mfp – frame pointer (for main)

stack pointer
(grows downward if func() calls another function)

library functions if
dynamically linked
(usual case)

frame pointer
points here

brk point

stack pointer
(top of stack)
points here

Offset from current
frame pointer (for func())

Contents

main() auto variables

73 y
72 x
ra return address
caller’s frame pointer

mfp

-4 garbage
-8 garbage
-12 garbage
-16 garbage

Low memory

ra (return address)

uninitialized data (bss)

initialized data

library functions if
statically linked
(not usual case)

All auto variables and parameters
are referenced via offsets from the
frame pointer.

The frame pointer and stack pointer
are in registers (for fast access).

When funct returns, the return value
is stored in a register. The stack pointer
is move to the y location, the code
is jumped to the return address (ra),
and the frame pointer is set to mfp
(the stored value of the caller’s frame
pointer). The caller moves the return
value to the right place.

Expanded view of the stack

Assumes int = long = char * of
size 4 and assumes stack at high
address and descending down.