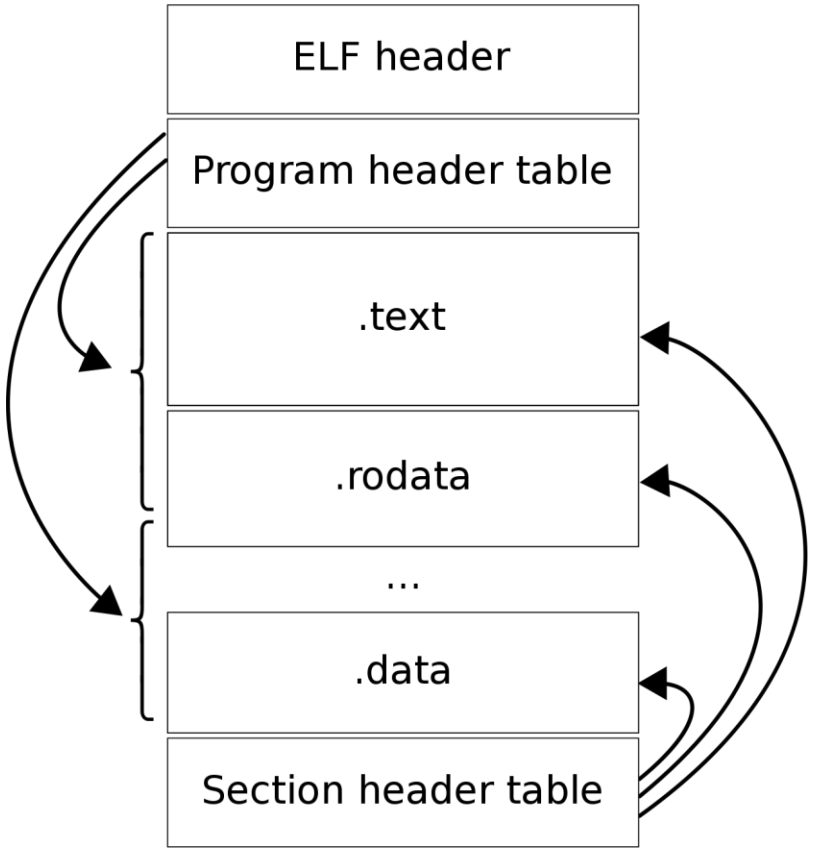
Memory layout & Makefile

C

- OS를 다루는 Low-level 프로그래밍에 좋다.
  - C와 ARM/x86/RISC-V 명령어 매핑이 쉽다. (inline)
  - C types과 하드웨어 데이터 구조와 연동이 좋다.
    - 디바이스 드라이버 개발 시
    - 하드웨어 레지스터를 위한 비트 플래그 설정이 쉽다.
- Runtime 시간이 최소화됨

ELF(Executable and Linkable Format)

- 리눅스 실행 파일 포멧
- text: code
- .rodata: read only data
- .data: global C 변수



https://en.wikipedia.org/wiki/Executable_and_Linkable_Format

프로세스 메모리 레이아웃

- text: code, read-only data
- data: global C 변수
- stack: function's local 변수
- heap: 동적 메모리 할당 malloc/free

C 프로그래밍 메모리 레이아웃

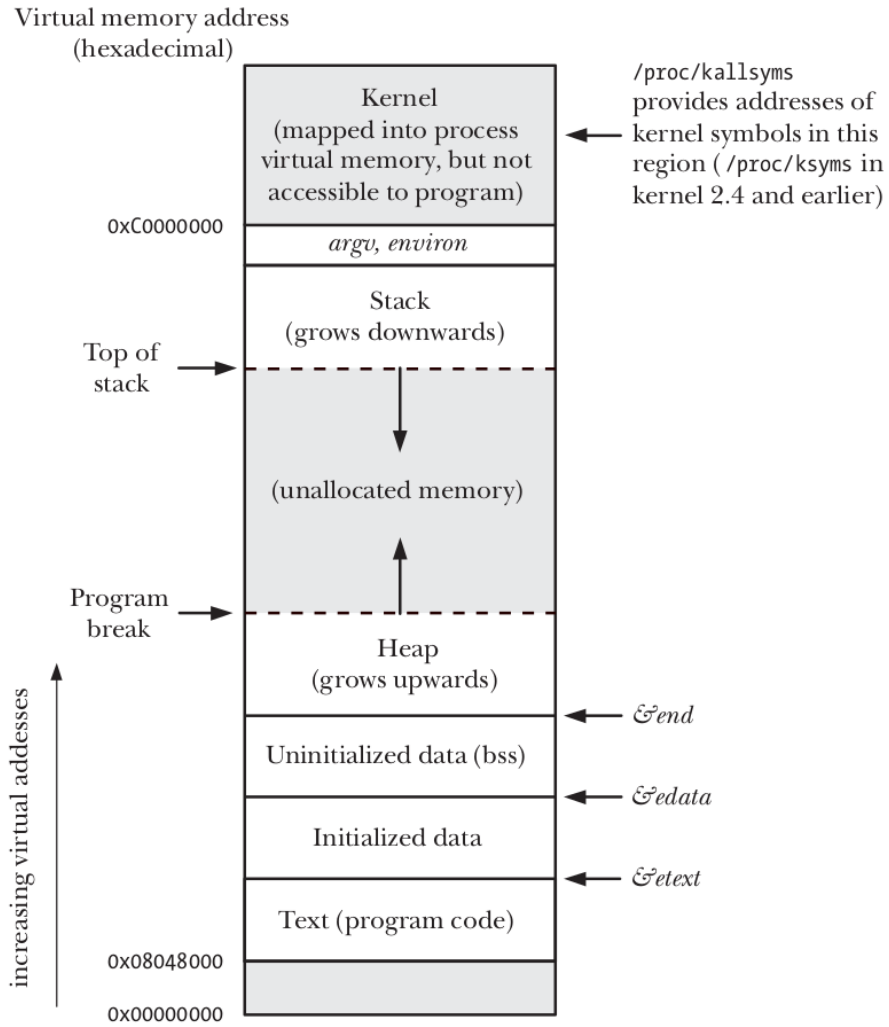


Figure 6-1: Typical memory layout of a process on Linux/x86-32

C 프로그래밍 메모리 레이아웃

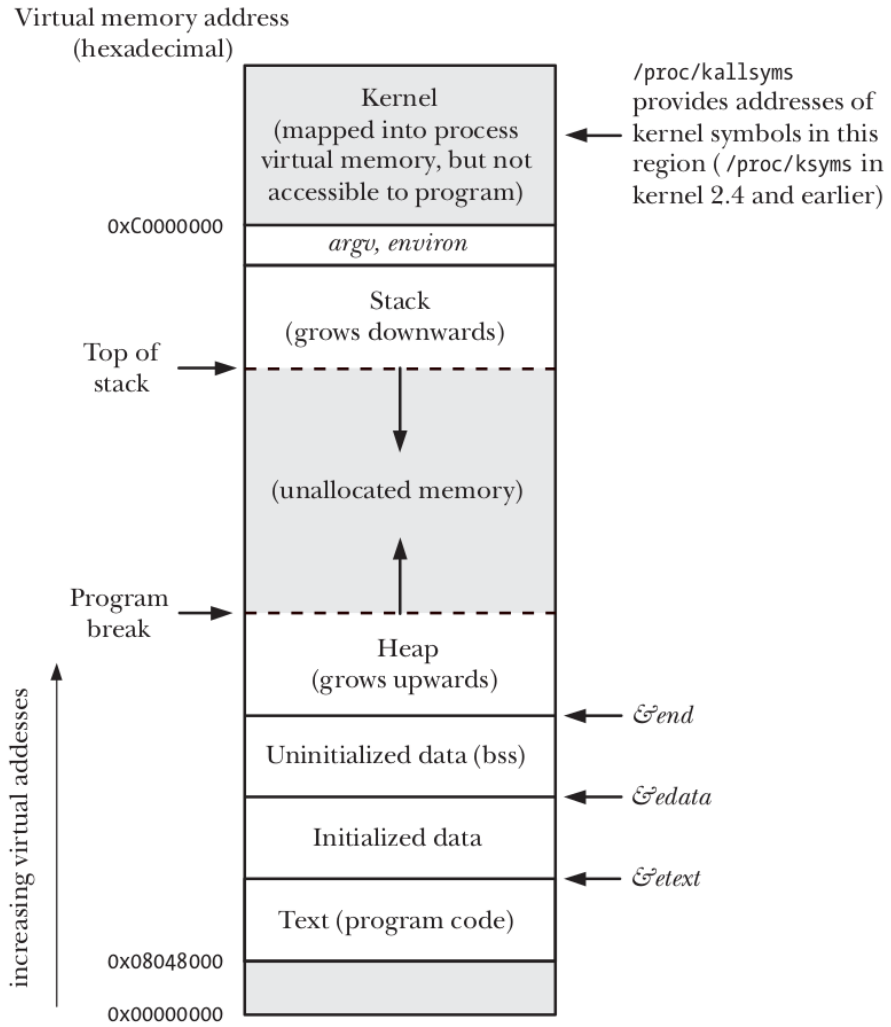


Figure 6-1: Typical memory layout of a process on Linux/x86-32

Listing 6-1: Locations of program variables in process memory segments

proc/mem_segments.c

```
#include <stdio.h>
#include <stdlib.h>

char globBuf[65536];            /* Uninitialized data segment */
int primes[] = { 2, 3, 5, 7 };  /* Initialized data segment */

static int
square(int x)                   /* Allocated in frame for square() */
{
    int result;                 /* Allocated in frame for square() */

    result = x * x;
    return result;              /* Return value passed via register */
}

static void
doCalc(int val)                 /* Allocated in frame for doCalc() */
{
    printf("The square of %d is %d\n", val, square(val));

    if (val < 1000) {
        int t;                  /* Allocated in frame for doCalc() */

        t = val * val * val;
        printf("The cube of %d is %d\n", val, t);
    }
}

int
main(int argc, char *argv[])    /* Allocated in frame for main() */
{
    static int key = 9973;      /* Initialized data segment */
    static char mbuf[10240000]; /* Uninitialized data segment */
    char *p;                    /* Allocated in frame for main() */

    p = malloc(1024);           /* Points to memory in heap segment */

    doCalc(key);

    exit(EXIT_SUCCESS);
}
```

proc/mem_segments.c

ELF 분석 도구(Binutils)

- readelf -a copy
  - 실행 파일(elf) 분석 도구
  - glibc는 C library
  - text (code), initialized data, symbol table, debug info...
- ojdump -S copy
  - 기계어로 번역
- nm copy
  - 심볼 출력

Makefile

- 대규모 프로그램 관리하는 툴

  예) 다른 툴들 maven, npm, yarn

- 아래 처럼 빌드하는 방법은? makefile

  gcc -o myprogram file1.c file2.c file3.c

make

- make : 소스 코드로 부터 실행 파일 또는 라이브러리를 생성하는 유틸리티

- Makefile : make 명령어가 사용하는 스크립트 파일
  - 어떻게 컴파일 하는지
  - 어떤 파일을 컴파일 하는지

Makefile 규칙

- target : source1 source2 ... sourceN
  command
  command

Example:

```
myprogram : file1.c file2.c file3.c
        gcc -o myprogram file1.c file2.c file3.c
```

예

```
program : foo.o bar.o
        gcc -o program foo.o bar.o

foo.o: foo.c
        gcc -c foo.c

bar.o: bar.c
        gcc -c bar.c
```

변수들

```
OBJFILES = file1.o file2.o file3.o
PROGRAM = myprog

$(PROGRAM): $(OBJFILES)
	gcc -o $(PROGRAM) $(OBJFILES)

clean:
	rm $(OBJFILES) $(PROGRAM)
```

make 실행

- make target

- make -f <makefilename>
- $ make -f <makefilename> target

• Makefile 분석(TLSP)

```
DIRS =  lib \
    acl altio \
    cap cgroups \
    daemons dirs_links \
    filebuff fileio filelock files filesys getopt \
    inotify \
    loginacct \
    memalloc \
    mmap \
    pgsjc pipes pmsg \
    proc proccred procexec procpri procres \
    progconc \
    psem pshm pty \
    shlibs \
    signals sockets \
    svipc svmsg svsem svshm \
    sysinfo \
    syslim \
    threads time timers tty \
    users_groups \
    vdso \
    vmem \
    xattr

# The "namespaces" and "seccomp" directories are deliberately excluded from
# the above list because much of the code in those directories requires a
# relatively recent kernel and userspace to build. Nevertheless, each of
# those directories contains a Makefile.

BUILD_DIRS = ${DIRS}


# Dummy targets for building and clobbering everything in all subdirectories

all:
    @ echo ${BUILD_DIRS}
    @ for dir in ${BUILD_DIRS}; do (cd $${dir}; ${MAKE}) ; \
        if test $$? -ne 0; then break; fi; done

allgen:
    @ for dir in ${BUILD_DIRS}; do (cd $${dir}; ${MAKE} allgen) ; done

clean:
    @ for dir in ${BUILD_DIRS}; do (cd $${dir}; ${MAKE} clean) ; done
```

1:**Makefile.inc**

```
TLPI_DIR = ..
TLPI_LIB = ${TLPI_DIR}/libtlpi.a
TLPI_INCL_DIR = ${TLPI_DIR}/lib

LINUX_LIBRT = -lrt
LINUX_LIBDL = -ldl
LINUX_LIBACL = -lacl
LINUX_LIBCRYPT = -lcrypt
LINUX_LIBCAP = -lcap

IMPL_CFLAGS = -std=c99 -D_XOPEN_SOURCE=600 \
        -D_DEFAULT_SOURCE \
        -g -I${TLPI_INCL_DIR} \
        -pedantic \
        -Wall \
        -W \
        -Wmissing-prototypes \
        -Wno-sign-compare \
        -Wimplicit-fallthrough \
        -Wno-unused-parameter

ifeq ($(CC),clang)
    IMPL_CFLAGS += -Wno-uninitialized -Wno-infinite-recursion \
            -Wno-format-pedantic
endif

CFLAGS = ${IMPL_CFLAGS}

IMPL_THREAD_FLAGS = -pthread

IMPL_LDLIBS = ${TLPI_LIB}

LDLIBS =

RM = rm -f
```

fileio/Makefile

1:Makefile

```
include ../Makefile.inc

GEN_EXE = atomic_append bad_exclusive_open copy \
    multi_descriptors seek_io t_readv t_truncate

LINUX_EXE = large_file

EXE = ${GEN_EXE} ${LINUX_EXE}

all : ${EXE}

allgen : ${GEN_EXE}


clean :
    ${RM} ${EXE} *.o

showall :
    @ echo ${EXE}

${EXE} : ${TLPI_LIB}        # True as a rough approximation
~
```

lib/Makefile.std

```
1:Makefile.std
# Makefile to build library used by all programs
#
# This make file relies on the assumption that each C file in this
# directory belongs in the library
#
# This makefile is very simple so that every version of make
# should be able to handle it
#
include ../Makefile.inc

# The library build is "brute force" -- we don't bother with
# dependency checking.

allgen : ${TLPI_LIB}

${TLPI_LIB} : *.c ename.c.inc
    ${CC} -c -g ${CFLAGS} *.c
    ${RM} ${TLPI_LIB}
    ${AR} rs ${TLPI_LIB} *.o

ename.c.inc :
    sh Build_ename.sh > ename.c.inc
    echo 1>&2 "ename.c.inc built"

clean :
    ${RM} *.o ename.c.inc ${TLPI_LIB}
~
~
~
~
'Makefile.std" 27L, 606C
```

strace

- 시스템 콜 트레이스
- starce <프로그램 명>

```
kesl@kesl-ThinkPad-T550:~/grepp/1.system_call/tlpi-dist/fileio$ strace ./copy seek_io seek_io_new
execve("./copy", ["./copy", "seek_io", "seek_io_new"], 0x7ffd4deff990 /* 63 vars */) = 0
brk(NULL)                               = 0x564640448000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffcbcd01f60) = -1 EINVAL (부적절한 인수)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (그런 파일이나 디렉터리가 없습니다)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=114642, ...}) = 0
mmap(NULL, 114642, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fd396d83000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\300A\2\0\0\0\0\0"..., 832) = 832
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0", 32, 848) = 32
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\30x\346\264ur\f|Q\226\236i\253-'o"..., 68, 880) = 68
fstat(3, {st_mode=S_IFREG|0755, st_size=2029592, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fd396d81000
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0", 32, 848) = 32
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\30x\346\264ur\f|Q\226\236i\253-'o"..., 68, 880) = 68
mmap(NULL, 2037344, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fd396b8f000
mmap(0x7fd396bb1000, 1540096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x22000) = 0x7fd396bb1000
mmap(0x7fd396d29000, 319488, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x19a000) = 0x7fd396d29000
mmap(0x7fd396d77000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7000) = 0x7fd396d77000
mmap(0x7fd396d7d000, 13920, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fd396d7d000
close(3)                                = 0
arch_prctl(ARCH_SET_FS, 0x7fd396d82540) = 0
mprotect(0x7fd396d77000, 16384, PROT_READ) = 0
mprotect(0x56463ea28000, 4096, PROT_READ) = 0
mprotect(0x7fd396dcc000, 4096, PROT_READ) = 0
munmap(0x7fd396d83000, 114642)          = 0
openat(AT_FDCWD, "seek_io", O_RDONLY)   = 3
```

실습

- 4가지 실습

1. strace 실습(5분)

- strace를 통해 시스템 콜 호출 내용을 본다.

2. TLPI Makefile 분석(10분)

- TLPI 소스 코드 중 Makefile 분석.

- 예)
  - Makefile
  - Makefile.inc
  - <폴더명>/Makefile

3. ELF 분석 실습(10분)

- 아래 유틸리티를 통해 ELF 파일 분석
- readelf
- nm
- objdump -S

4. TOY 프로젝트 Makefile 작성 실습 과제

- 샘플 소스를 기반으로 TOY 프로젝트 빌드를 위한 make 파일을 작성한다.
  - 현재 Makefile에 내용 없음. 수강생이 직접 작성
  - 샘플 소스는 수업 시간 전 제공

- 파일 내용
  - Makefile
  - ui/gui.c, input.c
  - web/webserver.c
  - system/system_server.c

- 빌드
  - make

- 최종 결과물(실행 파일)
  - toy_system