1111 Linux Operating System Project1

Cliff Chen

2022 年 11 月 6 日

1 Hands-on

1.1 Try Compile and Run Linux Kernel

1.1.1 Compile Linux Kernel 6.0.5

```
PROJECT=/home/cliff/projects/course-linux-operation-system/project1
cd $PROJECT

# Install compile tools
sudo apt install build-essential fakeroot libncurses5-dev \
                 libssl-dev ccache flex bison libelf-dev bc

# Download
git submodule add https://github.com/torvalds/linux
cd $PROJECT/linux
git checkout 3829606fc5dffeccdf80aebeed3aa75255257f35

# Test compilation
make ARCH=x86_64 x86_64_defconfig
make -j$(nproc)
```

1.1.2 Compile Busybox 1.35.0

```
cd $PROJECT

# Download
git submodule add https://github.com/mirror/busybox.git
cd $PROJECT/busybox

# Configuration
make defconfig

make menuconfig
# Busybox Settings → Build Options → [*] Build BusyBox as a static binary (no shared libs)

make -j$(nproc)
```

Generate initramfs

```
cd $PROJECT
mkdir initramfs
cd initramfs
mkdir -p bin sbin etc proc sys usr/bin usr/sbin
cp -av ../busybox/_install/* .
cat <<EOT > init
#!/bin/sh

mount -t proc none /proc
mount -t sysfs none /sys

echo -e ”\nBoot took \$(cut -d’ ’ -f1 /proc/uptime) seconds\n”

mkdir -p /mnt/host_share
mount -t 9p -o trans=virtio host_share /mnt/host_share -oversion=9p2000.L

exec /bin/sh -c ’/mnt/host_share/main; exec sh’
EOT
chmod +x init
find . -print0 | cpio --null -ov --format=newc | gzip -9 > ../initramfs.cpio.gz
```

1.1.3 Run

```
LINUX_KERNEL_PATH=$PROJECT/linux/arch/x86_64/boot/bzImage
INITRAMFS_PATH=$PROJECT/initramfs.cpio.gz
qemu-system-x86_64 \
  -kernel $LINUX_KERNEL_PATH \
  -initrd $INITRAMFS_PATH \
  -nographic \
  -enable-kvm \
  -append ”console=ttyS0” \
  -virtfs local,path=$PROJECT,security_model=passthrough,mount_tag=host_share
```

1.2 Task1, 2: Add `sys_segment_info` System Call

1.2.1 Modify Kernel: add system call

```
cat <<EOT > $PROJECT/sys_segment_info.c
#include <linux/types.h>
#include <linux/syscalls.h>
#include <linux/kernel.h>
#include <linux/sched.h>
#include <linux/ptrace.h>
```

```
#include <linux/thread_info.h>
#include <asm/current.h>

#include <linux/segment_info.h>

SYSCALL_DEFINE1(segment_info, struct segment_info *, dsi)
{
  struct vm_area_struct *vma = current->mm->mmap;
  struct segment_info si = {
      .ma_size = 0,
      .start_code = current->mm->start_code,
      .end_code = current->mm->end_code,
      .start_data = current->mm->start_data,
      .end_data = current->mm->end_data,
      .start_brk = current->mm->start_brk,
      .brk = current->mm->brk,
      .mmap_base = current->mm->mmap_base,
      .thread_sp = current_user_stack_pointer(),
  };

  while (vma)
  {
    struct ma_struct *ma = si.ma + si.ma_size++;
    *ma = (struct ma_struct){
        .vm_start = vma->vm_start,
        .vm_end = vma->vm_end,
        .name = ” \0” ,
    };

    if (vma->vm_file)
      // fs/d_path.c
      d_path(&vma->vm_file->f_path, ma->name, sizeof(ma->name));
    else if (vma->vm_ops && vma->vm_ops->name)
      strcpy(ma->name, vma->vm_ops->name(vma));

    vma = vma->vm_next;
  }

  if (copy_to_user(dsi, &si, sizeof(si)))
    return -1;

  return current->pid;
}
EOT
```

```
cat <<EOT > $PROJECT/segment_info.h
struct ma_struct
{
^^Iunsigned long vm_start, vm_end;
^^Ichar name[24];
};

struct segment_info
{
^^Iunsigned long start_code, end_code;
^^Iunsigned long start_data, end_data;
^^Iunsigned long start_brk, brk;
^^Iunsigned long start_stack, end_stack;
^^Iunsigned long thread_sp;
^^Iunsigned long mmap_base;
^^Iunsigned long ma_size;
^^Istruct ma_struct ma[24];
};
EOT
```

1.2.2 Add source file to options and compile

```
ln -s $PROJECT/sys_segment_info.c \
      $PROJECT/linux/arch/x86/kernel/sys_segment_info.c
ln -s $PROJECT/segment_info.h \
      $PROJECT/linux/include/linux/segment_info.h
echo " 451^^Icommon^^Isegment_info^^I^^Isys_segment_info"  \
      >> $PROJECT/linux/arch/x86/entry/syscalls/syscall_64.tbl
echo "obj-y                    += sys_segment_info.o"  \
      >> $PROJECT/linux/arch/x86/kernel/Makefile
cd $PROJECT/linux
make -j$(nproc)
```

% echo ” struct segment_info;\nasmlinkage long sys_segment_info(pid_t tid, struct segment_info* si);” » $PROJECT

1.2.3 Modify Guest User Space Code: add multi-thread

```
cat <<EOT > $PROJECT/main.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <sys/mman.h>
#include <sys/types.h>
#include <sys/syscall.h>
```

```
#include <pthread.h>

#include ”segment_info.h”

char bss[4];                    // bss
char data[5] = ”data”;          // data
char *heap;                     // heap
char *mmmap;                    // mmap
char *code() { return ”code”; } // code

char intersect(unsigned long a1, unsigned long a2, unsigned long b1, unsigned long b2)
{
  return (b2 > a1 && b2 <= a2) || (b1 >= a1 && b1 < a2) || (b1 <= a1 && b2 > a2);
}

void print_segment_info(char *thread_name)
{
  setvbuf(stdout, NULL, _IOFBF, 16384);
  char stack[6] = ”stack”; // stack

  struct segment_info si;
  printf(”☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒ [User Mode] %s thread id: %d ☒☒☒☒☒☒☒☒☒☒☒☒☒☒\n”, thread_name, sysca
  printf(” >>>>>> [%s]:\t\t’ %p’ \n”, code(), &code);
  printf(” >>>>>> [%s]:\t\t’ %p’ \n”, data, &data);
  printf(” >>>>>> [%s]:\t\t’ %p’ \n”, bss, &bss);
  printf(” >>>>>> [%s]:\t\t’ %p’ \n”, heap, heap);
  printf(” >>>>>> [%s]:\t\t’ %p’ \n”, mmmap, mmmap);
  printf(” >>>>>> [%s]:\t\t’ %p’ \n”, stack, &stack);
  printf(” >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>><<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<\n”);
  printf(” >>>>>> <start_code>:\t’ %p’ \n”, si.start_code);
  printf(” >>>>>> <end_code>:\t’ %p’ \n”, si.end_code);
  printf(” >>>>>> <start_data>:\t’ %p’ \n”, si.start_data);
  printf(” >>>>>> <end_data>:\t’ %p’ \n”, si.end_data);
  printf(” >>>>>> <start_brk>:\t’ %p’ \n”, si.start_brk);
  printf(” >>>>>> <brk>:\t\t’ %p’ \n”, si.brk);
  printf(” >>>>>> <mmap_base>:\t’ %p’ \n”, si.mmap_base);
  printf(” >>>>>> <thread_sp>:\t’ %p’ \n”, si.thread_sp);
  printf(” >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>><<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<\n”);

  for (int w = 0; w < si.ma_size; ++w)
  {
    char msg[100] = ”unknown ”;
    if (intersect(si.ma[w].vm_start, si.ma[w].vm_end, si.start_code, si.end_code))
      strcpy(msg, ”code/text segment ”);
    else if (intersect(si.ma[w].vm_start, si.ma[w].vm_end, si.start_data, si.end_data))
```

```
      strcpy(msg, "data segment ");
    else if (intersect(si.ma[w].vm_start, si.ma[w].vm_end, si.end_data, si.start_brk))
      strcpy(msg, "bss segment ");
    else if (intersect(si.ma[w].vm_start, si.ma[w].vm_end, si.start_brk, si.brk))
      strcpy(msg, "heap segment ");
    else if (intersect(si.ma[w].vm_start, si.ma[w].vm_end, si.thread_sp, si.thread_sp))
      sprintf(msg, "%s stack segment ", thread_name);
    else if (intersect(si.ma[w].vm_start, si.ma[w].vm_end, si.brk, si.mmap_base))
      strcpy(msg, "mmap segment(shared library, thread stack...) ");

    if (si.ma[w].name)
      sprintf(msg, "%s%s", msg, si.ma[w].name);

    printf(">>>>>> '%p' - '%p' %s\n", si.ma[w].vm_start, si.ma[w].vm_end, msg);
  }

  printf("^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^\n");
  fflush(stdout);
}

int main()
{
  heap = (char *)malloc(sizeof(char) * 5);
  mmmap = mmap(NULL, 5 * sizeof(char), PROT_READ | PROT_WRITE, MAP_PRIVATE | MAP_ANONYMOUS,
  strcpy(bss, "bss");
  strcpy(heap, "heap");
  strcpy(mmmap, "mmap");

  print_segment_info("main");

  pthread_t t1, t2;
  pthread_create(&t1, NULL, print_segment_info, "t1");
  pthread_create(&t2, NULL, print_segment_info, "t2");
  pthread_join(t1, NULL);
  pthread_join(t2, NULL);
  free(heap);
}
EOT
```

1.2.4 Compile and Run Virtual Machine

```
cd $PROJECT
gcc -Wno-format -Wno-incompatible-pointer-types \
    -Wno-implicit-function-declaration -Wno-error=unused-result \
    -o main main.c -static -g
```

```
LINUX_KERNEL_PATH=$PROJECT/linux/arch/x86_64/boot/bzImage
INITRAMFS_PATH=$PROJECT/initramfs.cpio.gz
qemu-system-x86_64 \
  -kernel $LINUX_KERNEL_PATH \
  -initrd $INITRAMFS_PATH \
  -nographic \
  -enable-kvm \
  -append ” console=ttyS0” \
  -virtfs local,path=$PROJECT,security_model=passthrough,mount_tag=host_share
```

Output

```
☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒ [User Mode] main thread id: 77 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒
>>>>>> [code]:          ’ 0x401845’
>>>>>> [data]:          ’ 0x4e1110’
>>>>>> [bss]:           ’ 0x4e33d0’
>>>>>> [heap]:          ’ 0x1d16770’
>>>>>> [mmap]:          ’ 0x7fa3955d4000’
>>>>>> [stack]:         ’ 0x7ffc10a9eb5a’
>>>>>>>>>>>>>>>>>>>>>>>>>>>><<<<<<<<<<<<<<<<<<<<<<<<<<<<
>>>>>> <start_code>:    ’ 0x401000’
>>>>>> <end_code>:      ’ 0x4b099d’
>>>>>> <start_data>:    ’ 0x4dd768’
>>>>>> <end_data>:      ’ 0x4e3370’
>>>>>> <start_brk>:     ’ 0x1d15000’
>>>>>> <brk>:           ’ 0x1d37000’
>>>>>> <mmap_base>:     ’ 0x7fa3955d5000’
>>>>>> <thread_sp>:     ’ 0x7ffc10a9e718’
>>>>>>>>>>>>>>>>>>>>>>>>>>>><<<<<<<<<<<<<<<<<<<<<<<<<<<<
>>>>>> ’ 0x400000’ –’ 0x401000’  unknown
>>>>>> ’ 0x401000’ –’ 0x4b1000’  code/text segment
>>>>>> ’ 0x4b1000’ –’ 0x4dd000’  unknown
>>>>>> ’ 0x4dd000’ –’ 0x4e1000’  data segment
>>>>>> ’ 0x4e1000’ –’ 0x4e4000’  data segment
>>>>>> ’ 0x4e4000’ –’ 0x4ea000’  bss segment
>>>>>> ’ 0x1d15000’ –’ 0x1d37000’  heap segment
>>>>>> ’ 0x7fa3955d4000’ –’ 0x7fa3955d5000’  mmap segment(shared library, thread stack...)
>>>>>> ’ 0x7ffc10a7f000’ –’ 0x7ffc10aa0000’  main stack segment
>>>>>> ’ 0x7ffc10bc1000’ –’ 0x7ffc10bc5000’  unknown [vvar]
>>>>>> ’ 0x7ffc10bc5000’ –’ 0x7ffc10bc7000’  unknown [vdso]
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒ [User Mode] t2 thread id: 79 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒
>>>>>> [code]:          ’ 0x401845’
>>>>>> [data]:          ’ 0x4e1110’
```

```
>>>>>> [bss]:            ’ 0x4e33d0’
>>>>>> [heap]:           ’ 0x1d16770’
>>>>>> [mmap]:           ’ 0x7fa3955d4000’
>>>>>> [stack]:          ’ 0x7fa394dd215a’
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>><<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
>>>>>> <start_code>:     ’ 0x401000’
>>>>>> <end_code>:       ’ 0x4b099d’
>>>>>> <start_data>:     ’ 0x4dd768’
>>>>>> <end_data>:       ’ 0x4e3370’
>>>>>> <start_brk>:      ’ 0x1d15000’
>>>>>> <brk>:            ’ 0x1d37000’
>>>>>> <mmap_base>:      ’ 0x7fa3955d5000’
>>>>>> <thread_sp>:      ’ 0x7fa394dd1d18’
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>><<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
>>>>>> ’ 0x400000’ –’ 0x401000’  unknown
>>>>>> ’ 0x401000’ –’ 0x4b1000’  code/text segment
>>>>>> ’ 0x4b1000’ –’ 0x4dd000’  unknown
>>>>>> ’ 0x4dd000’ –’ 0x4e1000’  data segment
>>>>>> ’ 0x4e1000’ –’ 0x4e4000’  data segment
>>>>>> ’ 0x4e4000’ –’ 0x4ea000’  bss segment
>>>>>> ’ 0x1d15000’ –’ 0x1d37000’  heap segment
>>>>>> ’ 0x7fa3945d2000’ –’ 0x7fa3945d3000’  mmap segment(shared library, thread stack...)
>>>>>> ’ 0x7fa3945d3000’ –’ 0x7fa394dd3000’  t2 stack segment
>>>>>> ’ 0x7fa394dd3000’ –’ 0x7fa394dd4000’  mmap segment(shared library, thread stack...)
>>>>>> ’ 0x7fa394dd4000’ –’ 0x7fa3955d5000’  mmap segment(shared library, thread stack...)
>>>>>> ’ 0x7ffc10a7f000’ –’ 0x7ffc10aa0000’  unknown
>>>>>> ’ 0x7ffc10bc1000’ –’ 0x7ffc10bc5000’  unknown [vvar]
>>>>>> ’ 0x7ffc10bc5000’ –’ 0x7ffc10bc7000’  unknown [vdso]
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
☒☒☒☒☒☒☒☒☒☒☒☒☒ [User Mode] t1 thread id: 78 ☒☒☒☒☒☒☒☒☒☒☒☒
>>>>>> [code]:           ’ 0x401845’
>>>>>> [data]:           ’ 0x4e1110’
>>>>>> [bss]:            ’ 0x4e33d0’
>>>>>> [heap]:           ’ 0x1d16770’
>>>>>> [mmap]:           ’ 0x7fa3955d4000’
>>>>>> [stack]:          ’ 0x7fa3955d315a’
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>><<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
>>>>>> <start_code>:     ’ 0x401000’
>>>>>> <end_code>:       ’ 0x4b099d’
>>>>>> <start_data>:     ’ 0x4dd768’
>>>>>> <end_data>:       ’ 0x4e3370’
>>>>>> <start_brk>:      ’ 0x1d15000’
>>>>>> <brk>:            ’ 0x1d37000’
>>>>>> <mmap_base>:      ’ 0x7fa3955d5000’
>>>>>> <thread_sp>:      ’ 0x7fa3955d2d18’
```

```
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>><<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
>>>>>> ' 0x400000' -' 0x401000'  unknown
>>>>>> ' 0x401000' -' 0x4b1000'  code/text segment
>>>>>> ' 0x4b1000' -' 0x4dd000'  unknown
>>>>>> ' 0x4dd000' -' 0x4e1000'  data segment
>>>>>> ' 0x4e1000' -' 0x4e4000'  data segment
>>>>>> ' 0x4e4000' -' 0x4ea000'  bss segment
>>>>>> ' 0x1d15000' -' 0x1d37000'  heap segment
>>>>>> ' 0x7fa3945d2000' -' 0x7fa3945d3000'  mmap segment(shared library, thread stack...)
>>>>>> ' 0x7fa3945d3000' -' 0x7fa394dd3000'  mmap segment(shared library, thread stack...)
>>>>>> ' 0x7fa394dd3000' -' 0x7fa394dd4000'  mmap segment(shared library, thread stack...)
>>>>>> ' 0x7fa394dd4000' -' 0x7fa3955d5000'  t1 stack segment
>>>>>> ' 0x7ffc10a7f000' -' 0x7ffc10aa0000'  unknown
>>>>>> ' 0x7ffc10bc1000' -' 0x7ffc10bc5000'  unknown [vvar]
>>>>>> ' 0x7ffc10bc5000' -' 0x7ffc10bc7000'  unknown [vdso]
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

2 Prior Knowledge

2.1 Task, Thread, Process in Linux

在 Linux 作業系統中，執行的最小單位稱為 Task，資料結構是由 include/linux/sched.h#L727 (v6.0.5) 下的 `task_struct` 定義，可以看作是一個 process descriptor。

2.2 SYSCALL_DEFINEx

2.3 Page Table

2.4 VMA

2.5 initrd, initramfs

2.6 copy_to_user, copy_from_user

2.7 Linux Copy On Write Memory

2.8 task_struct

2.8.1 mm_struct

2.9 thread_info

3 Reference

3.1 Build Linux Kernel

- Building a Custom Linux Kernel & Debugging via QEMU + GDB

- Prepare the environment for developing Linux kernel with qemu.
- How to Build A Custom Linux Kernel For Qemu
- Build the Linux kernel and Busybox and run them on QEMU

3.2 Build BusyBox with `host_share` storage

- How to qemu-arm with busybox linux and shared folder

3.3 Add System Call

- Adding a New System Call
- How to pass parameters to Linux system call?
- System Call (系統呼叫)
- System calls in the Linux kernel. Part 1.

3.4 Misc

- Which Linux syscall is used to get a thread’s ID?
- How can we get the starting address of task_struct of a process

3.4.1 To Be Organized

Tier 1

- Address Space
- Chapter 3 Page Table Management
- How The Kernel Manages Your Memory
- Page Tables
- Process Address Space
- Virtual Memory (虛擬記憶體)
- OS Process & Thread (user/kernel) 筆記
- Linux 核心 Copy On Write - Memory Region
- Linux 核心 Copy On Write 實作機制
- Linux 核心設計: Memory
- Linux 核心設計: 記憶體管理
- Linux 作業系統學習筆記（三）核心初始化
- Linux 的程序地址空間 [三]
- Linux 的程序地址空間 [二] - VMA
- Linux 程序描述符 task_struct 結構體詳解–Linux 程序的管理與排程（一）
- Linux 程序核心棧與 thread_info 結構詳解–Linux 程序的管理與排程（九）
- Linux 程序棧空間大小
- Linux 程序棧空間大小
- Linux 記憶體管理第三章 – 頁表管理 (Page Table Management)
- 分享一個關於 pthread 執行緒棧在 mm_struct 裡面的分佈問題

Tier 2

- ”current” in Linux kernel code

- How to get the physical address from the logical one in a Linux kernel module?
- Is stack memory contiguous?
- Is stack memory contiguous physically in Linux?
- Linux Kernel —get page global directory and analyze the result
- NCTU OSDI Dicussion - Memory Management III
- Where are the stacks for the other threads located in a process virtual address space?
- （三）程序各種 id：pid、pgid、sid、全域性 pid、區域性 pid
- 【原創】（十三）Linux 記憶體管理之 vma/malloc/mmap
- /proc//maps 簡要分析
- linux 記憶體管理 (8) —記憶體描述符 (mm_struct)
- Linux 程序地址管理之 mm_struct
- mm_struct 簡介