OS Assignment (UFCFWK-15-2)

Shancang Li

Department of Computer Science and Creative Technologies,
UWE Bristol

shancang.li@uwe.ac.uk

October 30, 2017

Contents

- All code part of Pintos Kernel
- Code compiled directly with the kernel
- From now on, run user programs on top of kernel
  - Modify the kernel to make the user program work

Example User Program in C

- In C, a user program test.c can pass argument
  - in `echo.c`

```
00001: #include <stdio.h>
00002: #include <syscall.h>
00003:
00004: int
00005: main (int argc, char **argv)
00006: {
00007:   int i;
00008:
00009:   for (i = 0; i < argc; i++)
00010:     printf ("%s ", argv[i]);
00011:   printf ("\n");
00012:
00013:   return EXIT_SUCCESS;
00014: }
```

  - `./echo arg1 arg2`

Example User Program in C

University of the West of England, BRISTOL

- vi.c can call system libraries
  - in `vi.c`

```
00001: #include <stdio.h>
00002: int main()
00003: {
00004:     file* p_file = file_open("myfile.txt","w");
00005:     if (p_file != NULL) file_write("file_open", p_file);
00006:     file_close(p_file);
00007: }
```

  - Get `file_open`, `file_write`, `file_close` by system calls
- Pintos need you to implement
  - Argument passing
  - System calls

Using File System

- May need to interact with file system
- Do not modify the file system!
- Certain limitations
  - No internal synchronization
  - Fixed file size
  - No subdirectories
  - File names limited to 14 chars
  - System crash might corrupt the file system
- Files to take a look at: `filesys.h`, `file.h`

Some commands

- Creating a simulated disk
  `$ pintos-mkdisk filesys.dsk --filesys-size=2`
- Formatting the disk
  `$ pintos -f -q`
- Copying the program into the disk
  `$ pintos -p ../../examples/echo -a echo ---q`
- Running the program
  `$ pintos -q run echo x`
- Single command:
  `$ pintos --fs-disk=2 -p ../../examples/echo -a echo ---f -q run echo x`

Various directory

- Few user programs
  - `Pintos/examples`
- Relevant files
  - `Pintos/userprog`
- Other files
  - `Pintos/threads, Pintos/filesys/`

Requirements

- Process Termination Messages (worksheet 3)
- **Argument Passing**
- **System calls**

You will be working in **group of two or three** for this assignment. You must submit

- Architecture design **document**, detailing your design and how you implemented it.
- Commented **source code**. All changes and additions to the source code must be documented

Getting Started

University of the West of England, BRISTOL

- In the default version, programs will crash
- To get simple programs to run:
  - *esp = PHYS_BASE - 12;
- This will NOT make argument passing work

Process Termination

- Process Termination Messages
  - `printf("%s : exit(%d ", thread-name, thread-exit-code);`
  - for e.g.: `args-single: exit(0)`
- Do not print any other message!

Main idea - 1

University of the West of England, BRISTOL

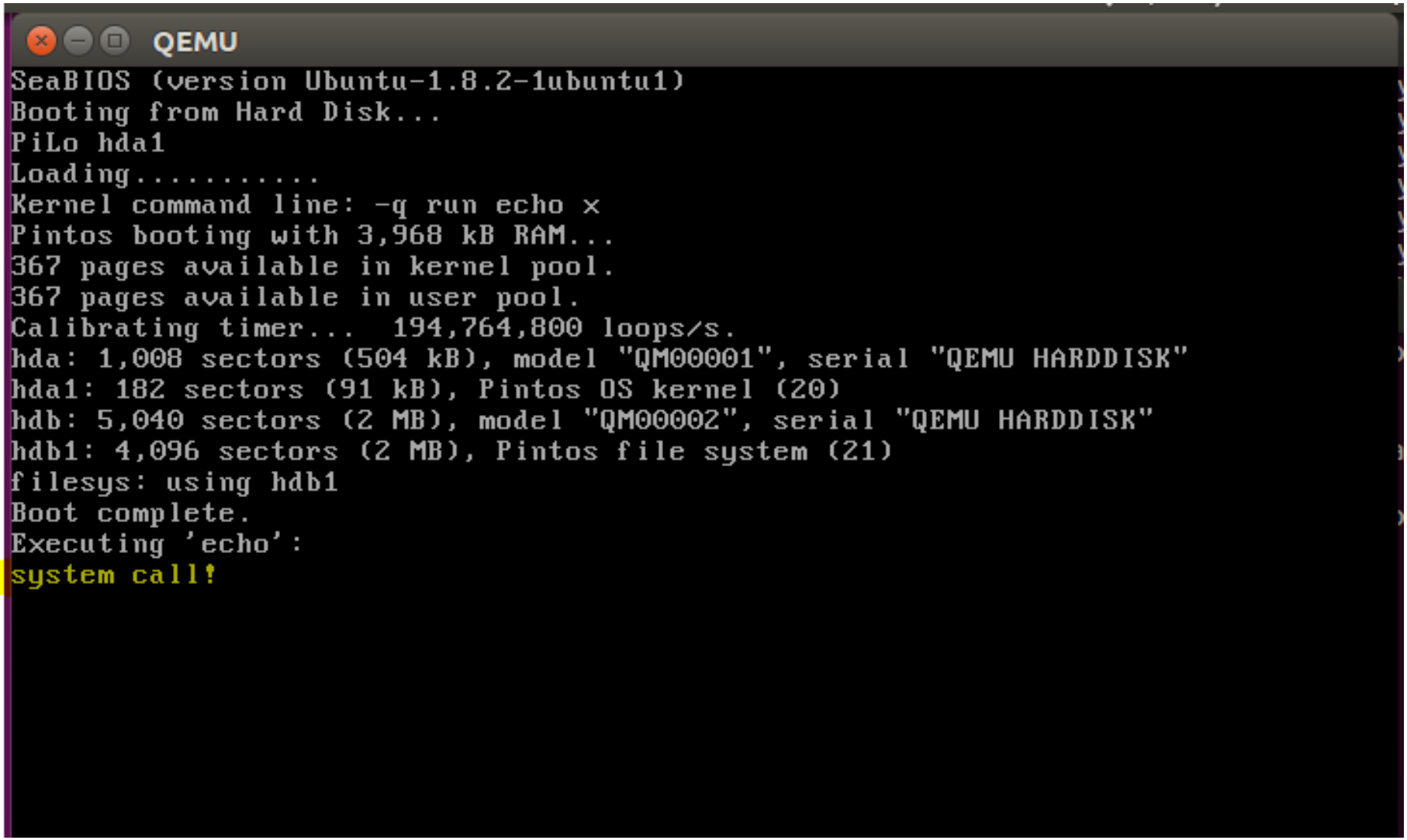
```
thread_exit (void)
{
  ASSERT (!intr_context ());

#ifdef USERPROG
  process_exit ();
#endif

  /* Remove thread from all threads list, set our status to dying,
     and schedule another process.  That process will destroy us
     when it calls thread_schedule_tail(). */
  intr_disable ();
  list_remove (&thread_current()->allelem);
  thread_current ()->status = THREAD_DYING;
  schedule ();
  NOT_REACHED ();
}
process_exit (void)
{
  struct thread *cur = thread_current ();
  uint32_t *pd;

  /* Destroy the current process's page directory and switch back
     to the kernel-only page directory. */
  pd = cur->pagedir;
  if (pd != NULL)
    {
      /* Correct ordering here is crucial.  We must set
         cur->pagedir to NULL before switching page directories,
         so that a timer interrupt can't switch back to the
         process page directory.  We must activate the base page
         directory before destroying the process's page
         directory, or our active page directory will be one
         that's been freed (and cleared). */
      cur->pagedir = NULL;
      pagedir_activate (NULL);
      pagedir_destroy (pd);
    }
} « end process_exit »
```

Main idea - 2

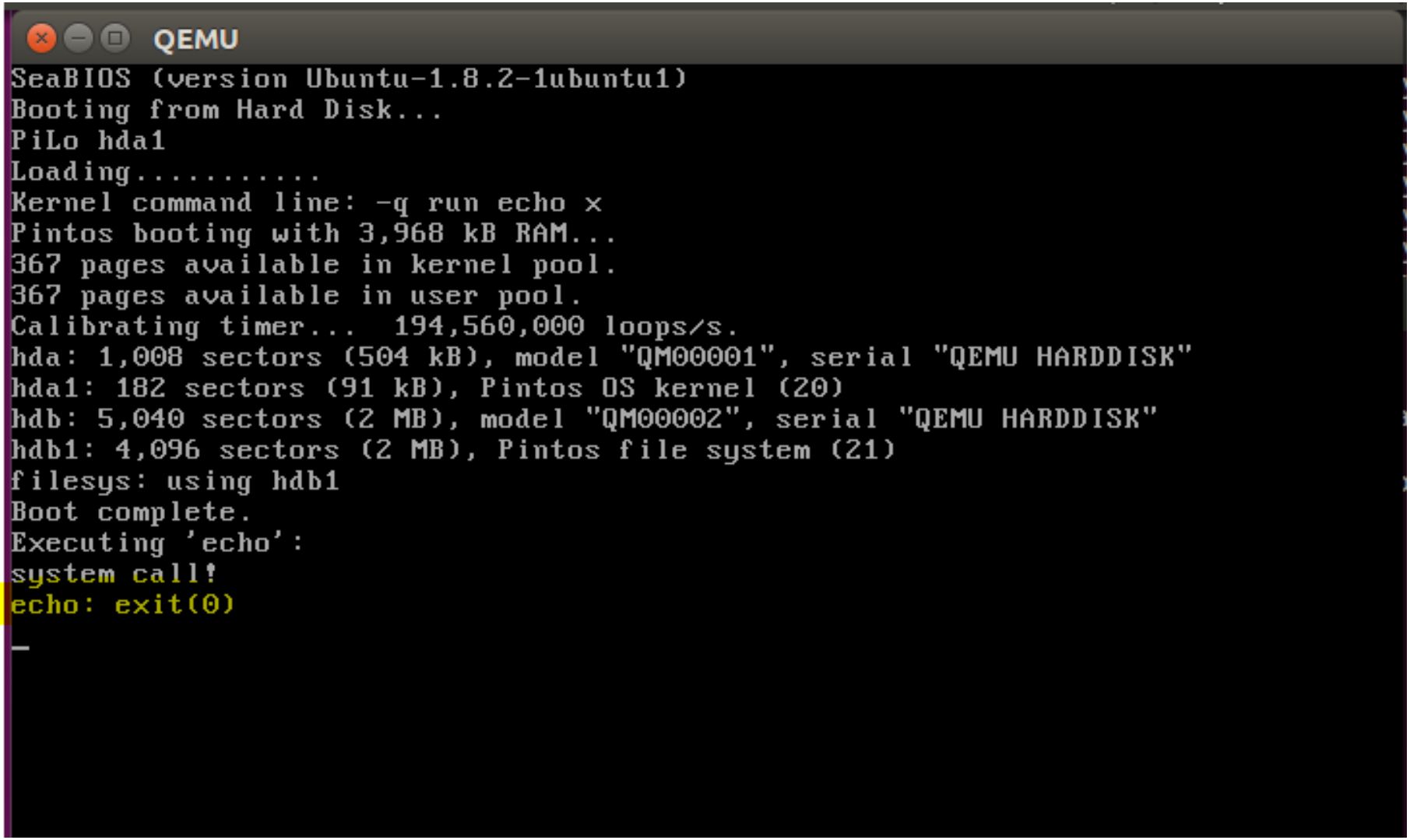
University of the West of England, BRISTOL

```
struct thread
  {
    /* Owned by thread.c. */
    tid_t tid;                          /* Thread identifier. */
    enum thread_status status;          /* Thread state. */
    char name[16];                      /* Name (for debugging purposes). */
    uint8_t *stack;                     /* Saved stack pointer. */
    int priority;                       /* Priority. */
    struct list_elem allelem;           /* List element for all threads list. */

    /* Shared between thread.c and synch.c. */
    struct list_elem elem;              /* List element. */

    /* int64_t ticks ç”¨äºŽè®¾ç½®timer_sleep()çš„å”¤é†’æ—¶é—´ */
    int64_t ticks;

#ifdef USERPROG
    /* Owned by userprog/process.c. */
    uint32_t *pagedir;                  /* Page directory. */
#endif

    /* Owned by thread.c. */
    unsigned magic;                     /* Detects stack overflow. */
  } « end thread » ;
```

Example


QEMU
SeaBIOS (version Ubuntu-1.8.2-1ubuntu1)
Booting from Hard Disk...
PiLo hda1
Loading..........
Kernel command line: -q run echo x
Pintos booting with 3,968 kB RAM...
367 pages available in kernel pool.
367 pages available in user pool.
Calibrating timer... 194,764,800 loops/s.
hda: 1,008 sectors (504 kB), model "QM00001", serial "QEMU HARDDISK"
hda1: 182 sectors (91 kB), Pintos OS kernel (20)
hdb: 5,040 sectors (2 MB), model "QM00002", serial "QEMU HARDDISK"
hdb1: 4,096 sectors (2 MB), Pintos file system (21)
filesys: using hdb1
Boot complete.
Executing 'echo':
system call!

Example

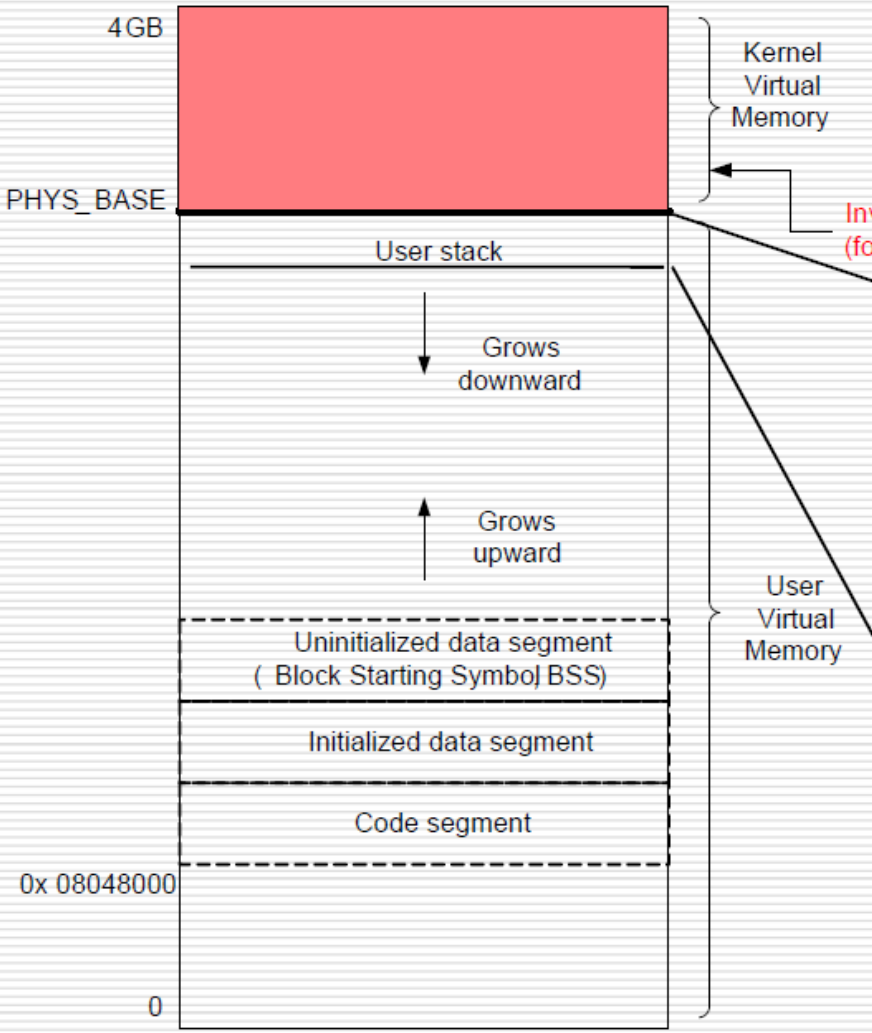

QEMU
SeaBIOS (version Ubuntu-1.8.2-1ubuntu1)
Booting from Hard Disk...
PiLo hda1
Loading.....
Kernel command line: -q run echo x
Pintos booting with 3,968 kB RAM...
367 pages available in kernel pool.
367 pages available in user pool.
Calibrating timer... 194,560,000 loops/s.
hda: 1,008 sectors (504 kB), model "QM00001", serial "QEMU HARDDISK"
hda1: 182 sectors (91 kB), Pintos OS kernel (20)
hdb: 5,040 sectors (2 MB), model "QM00002", serial "QEMU HARDDISK"
hdb1: 4,096 sectors (2 MB), Pintos file system (21)
filesys: using hdb1
Boot complete.
Executing 'echo':
system call!
echo: exit(0)

Argument Passing

- Pintos currently lacks argument passing. You need too implement it.
- Change `*esp=PHYS_BASE` to `*esp=PHYS_BASE-12` in `setup_stack()` to get started.
- Change `process_execute()` in process.c to process multiple arguments
- String parsing: `strtok_r()` in `lib/string.h`

Argument Passing

Set up the stack for program: `ls -l foo bar`



PHYS_BASE = 3GB

Invalid Pointer Area (for User Programs)

Address	Name	Data	Type
0xbffffffc	argv[3][...]	'bar\0'	char[4]
0xbffffff8	argv[2][...]	'foo\0'	char[4]
0xbffffff5	argv[1][...]	'-l\0'	char[3]
0xbfffffed	argv[0][...]	'/bin/ls\0'	char[8]
0xbfffffec	word-align	0	uint8_t
0xbfffffe8	argv[4]	0	char *
0xbfffffe4	argv[3]	0xbffffffc	char *
0xbfffffe0	argv[2]	0xbffffff8	char *
0xbfffffdc	argv[1]	0xbffffff5	char *
0xbfffffd8	argv[0]	0xbfffffed	char *
0xbfffffd4	argv	0xbfffffd8	char **
0xbfffffd0	argc	4	int
0xbfffffcc	return address	0	void (*) ()

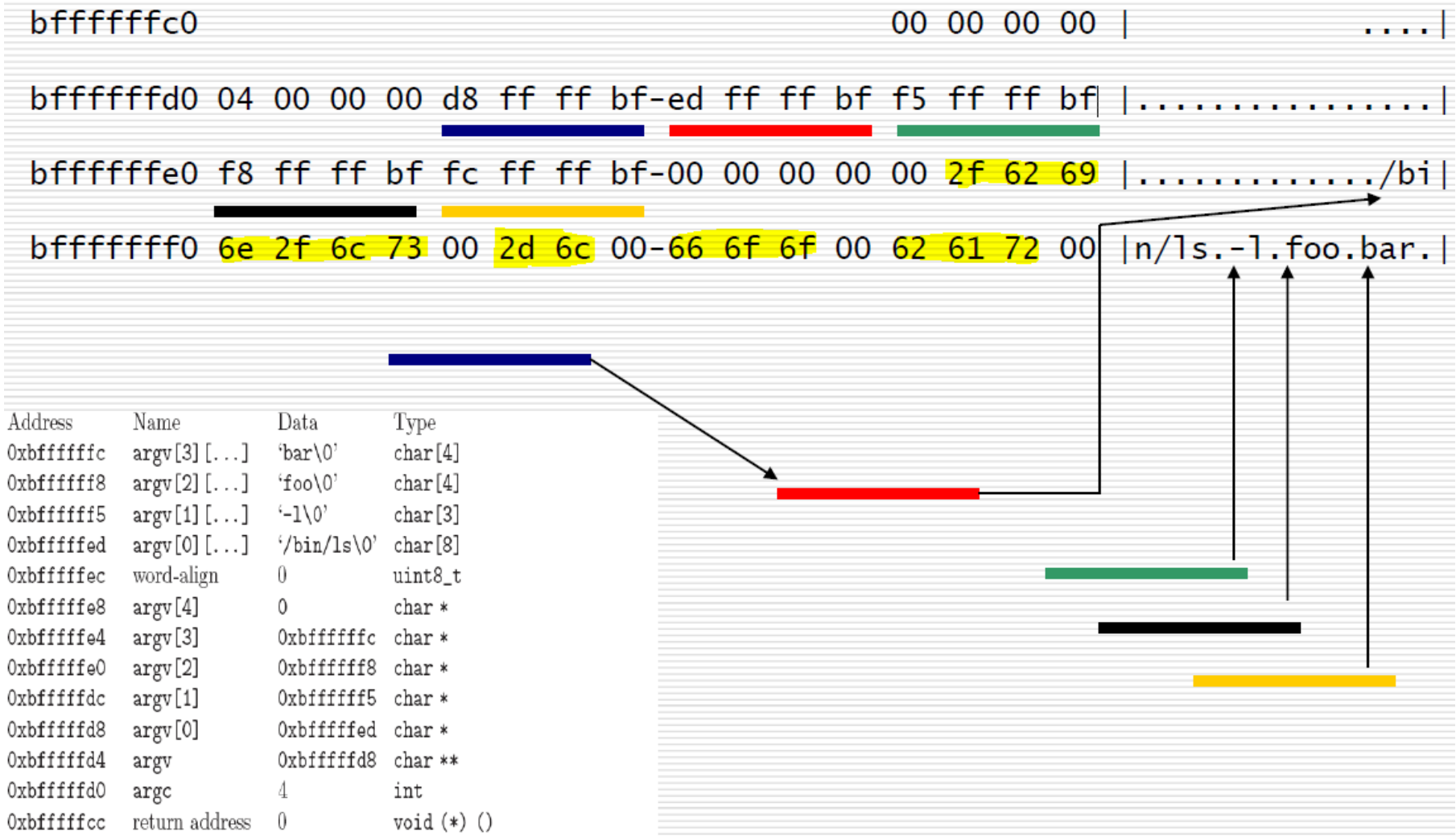
Argument Passing

Set up the stack for program: `ls -l foo bar`

Address	Name	Data	Type
0xbffffffc	argv[3][...]	'bar\0'	char[4]
0xbffffff8	argv[2][...]	'foo\0'	char[4]
0xbffffff5	argv[1][...]	'-l\0'	char[3]
0xbfffffed	argv[0][...]	'/bin/ls\0'	char[8]
0xbfffffec	word-align	0	uint8_t
0xbfffffe8	argv[4]	0	char *
0xbfffffe4	argv[3]	0xbffffffc	char *
0xbfffffe0	argv[2]	0xbffffff8	char *
0xbfffffdc	argv[1]	0xbffffff5	char *
0xbfffffd8	argv[0]	0xbfffffed	char *
0xbfffffd4	argv	0xbfffffd8	char **
0xbfffffd0	argc	4	int
0xbfffffcc	return address	0	void (*) ()

Argument Passing

Set up the stack for program: `ls -l foo bar`



Address	Name	Data	Type
0xbffffffc	argv[3][...]	'bar\0'	char[4]
0xbffffff8	argv[2][...]	'foo\0'	char[4]
0xbffffff5	argv[1][...]	'-l\0'	char[3]
0xbfffffed	argv[0][...]	'/bin/ls\0'	char[8]
0xbfffffec	word-align	0	uint8_t
0xbfffffe8	argv[4]	0	char *
0xbfffffe4	argv[3]	0xbffffffc	char *
0xbfffffe0	argv[2]	0xbffffff8	char *
0xbfffffdc	argv[1]	0xbffffff5	char *
0xbfffffd8	argv[0]	0xbfffffed	char *
0xbfffffd4	argv	0xbfffffd8	char **
0xbfffffd0	argc	4	int
0xbffffffcc	return address	0	void (*) ()

System Calls – Next week

- Pintos currently lacks system call. You need too implement it.
- Implement the system call handler in userprog/syscall.c
- System call number defined in lib/syscall-nr.h
- Process Control: exit, exec, wait.
- File system: create, remove, open, filesize, read, write, seek, tell, close
- Others: halt

Thanks!