세계 속의 GIST 이공계 대학

Operating System Project 1

Release Date: 2024-03-11

Due Date: 2024-03-25

GIST

Gwangju Institute of Science and Technology

0. Background

- In the first project, you need to configure your system for PintOS project
  - Virtual Machine: VMWare
  - OS: Ubuntu Linux
  - Emulator: QEMU

- Your PintOS runs on the emulated x86 CPU in Linux environment, and you need to know how to code in Linux

- VI editor
  - The most common editor for developing C/C++ program in Ubuntu Linux

0-1. VI editing

- Dual-mode operation
  - Input mode
    - Available to input texts by typing
  - Control mode
    - Supports every functions that manage/control texts
    - Various commands exist for text manipulation

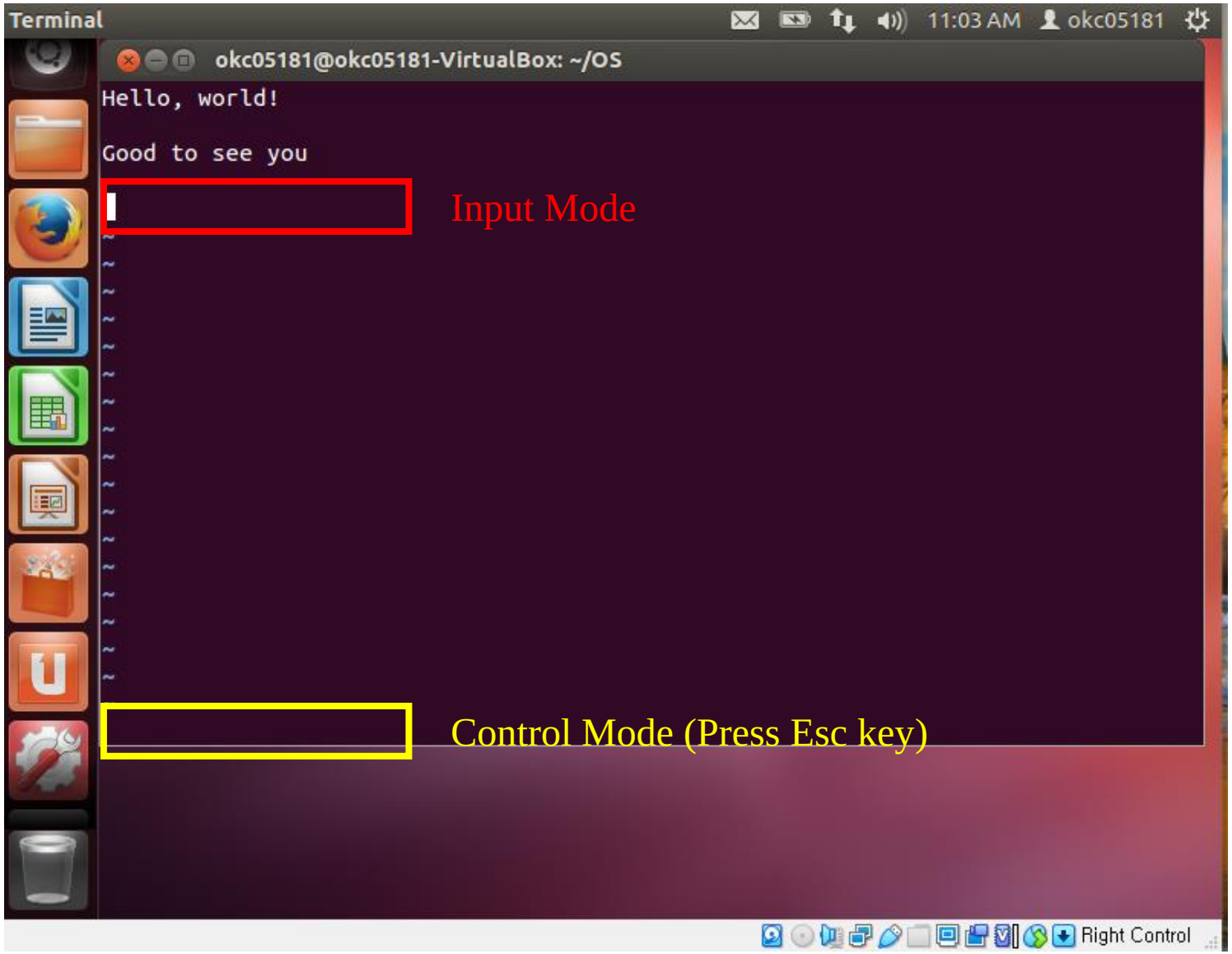
  - You can open VI editor by typing
    - vi filename

- Default Control
  - When you open a file, control mode is default
  - You can freely control your cursor location by inputting arrow, page up and down, home and end keys, but **in control mode only**

0-1. VI editing

- How to use VI editor
  - In control mode, you can type ‘i’, ‘a’, A’, … commands to enter input mode
    - i: input starts at the current cursor position
    - a: input starts at the next position to the current cursor
    - A: input starts at the end of the current word
  - Other tips
    - Show the line number where the cursor locates at: Press Esc key (change mode from input to control), then type *:num*
    - Move the cursor to the line number: Press Esc key, then type *:# (Ex> :127)*
    - Save and close file: Press Esc key, then type *:wq (:w for saving only without exit)*
    - Find a specific text after the current cursor position: Press Esc key, then type */text* (Ex: */name*, searches the text ‘name’ in the opened file starting from the current cursor point. If you type / and Enter after you searched once, vi searches the next ‘name’ and put a cursor at the start of ‘name’. Or you can click on ‘N’ key to find next ‘name’.)
    - Find a specific text before the current cursor position: Press Esc key, then type *?text*

0-1. VI editing



0-2. Project #1 Introduction

- **Objectives**
  1) Install a local Linux system by using a VMware virtual machine
  2) Make a repository for PintOS and share it among your teammates and a TA
  3) Configure your local system to develop the PintOS project (2 pts)
  4) Implement Alarm Clock in the PintOS project (2 pts)
  5) Write down a report for explaining policy and mechanism of implemented alarm clock (1 pt)

- **Caution**
  - Do not copy any of codes. If you submit incomplete project with a report, you will get base points.
  - If you plagiarize any of codes, you will get **F** regardless of your scores.

- **What to submit?**
  - Please capture the result of 3-e (test result in your virtual machine) and submit the image to TA (minwook-lee@gm.gist.ac.kr) for **Objective 3**, **Objective 4**, and **Objective 5**.
  - Due date: 2024-03-24 11:59 PM
  - Late policy: 10% decrement for each day

1. VM ware install

A) Download VM

- https://my.vmware.com/en/web/vmware/downloads/info/slug/desktop_end_user_computing/vmware_workstation_player/16_0
- Choose a platform that matches your OS.
- Download the file

B) Install Ubuntu 14.04

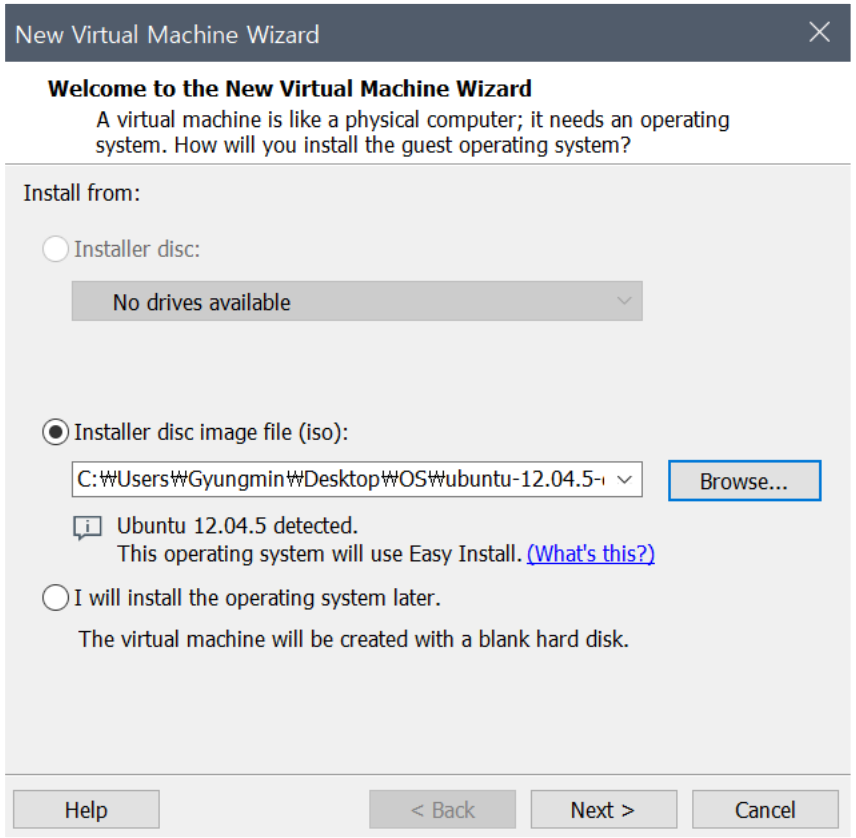
- http://releases.Ubuntu.com/14.04/

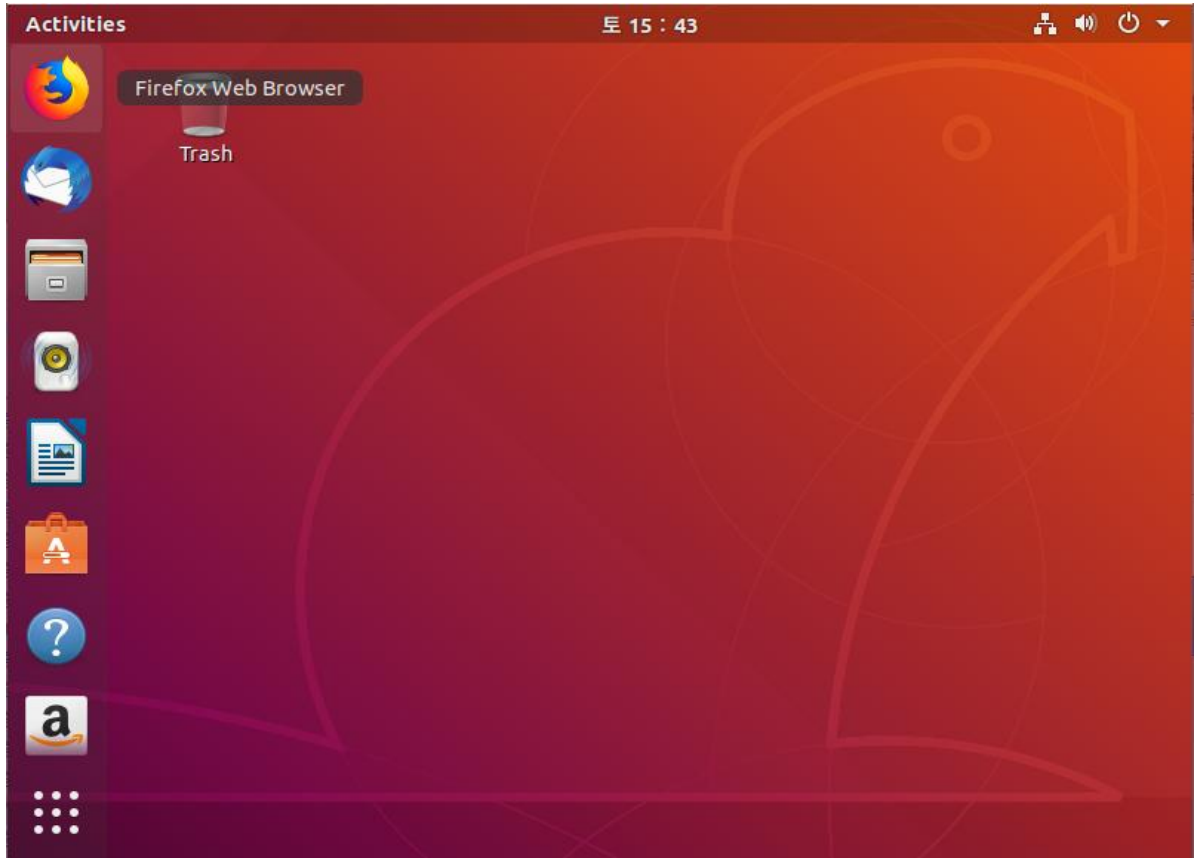
1) Download a desktop image, ‘Ubuntu-14.04.6-desktop-i386.iso’
2) Start the VM and choose the image file for Ubuntu installation
3) Follow the instructions to install Ubuntu Linux

1. VM ware install

B) Install Ubuntu 14.04

3) Start the VM and choose the image file for Ubuntu installation





Screen you should see after installation is done

2. Environment Software Install

- Terminal
  - Ctrl + Alt + T

a) Install gcc-4.4

- **sudo apt-get install gcc-4.4**
- **sudo mv /usr/bin/gcc-4.4 /usr/bin/gcc**
- **gcc –v**
- **sudo apt-get install g++**

```
a@ubuntu:~$ sudo apt-get install gcc-4.4
```

```
a@ubuntu:~$ sudo mv /usr/bin/gcc-4.4 /usr/bin/gcc
a@ubuntu:~$ gcc -v
Using built-in specs.
Target: i686-linux-gnu
Configured with: ../src/configure -v --with-pkgversion='Ubuntu/Linaro 4.4.7-8ubun
tu1' --with-bugurl=file:///usr/share/doc/gcc-4.4/README.Bugs --enable-languages=c
,c++,fortran --prefix=/usr --program-suffix=-4.4 --enable-shared --enable-linker-
build-id --with-system-zlib --libexecdir=/usr/lib --without-included-gettext --en
able-threads=posix --with-gxx-include-dir=/usr/include/c++/4.4 --libdir=/usr/lib
--enable-nls --with-sysroot=/ --enable-clocale=gnu --enable-libstdcxx-debug --dis
able-libmudflap --enable-targets=all --disable-werror --with-arch-32=i686 --with-
tune=generic --enable-checking=release --build=i686-linux-gnu --host=i686-linux-g
nu --target=i686-linux-gnu
Thread model: posix
gcc version 4.4.7 (Ubuntu/Linaro 4.4.7-8ubuntu1)
```

```
a@ubuntu:~$ sudo apt-get install g++
```

2. Environment Software Install

b) Go to the link in the Ubuntu environment and download the file

- https://sourceforge.net/projects/bochs/files/bochs/2.6.2/bochs-2.6.2.tar.gz/download

c) Go to the Download directory, and unzip the file

- **cd Downloads/**
- **tar xvf bochs-2.6.2.tar.gz**
- **cd bochs-2.6.2**
- **./configure –enable-gdb-stub –with-nogui**
- **make**
- **sudo make install**

d) Install qemu

- **sudo apt-get install qemu**

```
a@ubuntu:~$ ls
Desktop    Downloads         fontconfig  Pictures  Templates
Documents  examples.desktop  Music       Public    Videos
a@ubuntu:~$ cd Downloads/
a@ubuntu:~/Downloads$ ls
bochs-2.6.2.tar.gz
a@ubuntu:~/Downloads$ tar xvf bochs-2.6.2.tar.gz
```

```
a@ubuntu:~/Downloads$ cd bochs-2.6.2/
a@ubuntu:~/Downloads/bochs-2.6.2$ ls
aclocal.m4       cpu                logio.cc        pc_system.cc
bios             cpudb.h            ltdl.c          pc_system.h
bochs.h          crc.cc             ltdlconf.h.in   plugin.cc
build            disasm             ltdl.h          plugin.h
bx_debug         doc                ltmain.sh       README
bxversion.h.in   docs-html          main.cc         README-plugins
bxversion.rc.in  extplugin.h        Makefile.in     README.rfb
CHANGES          gdbstub.cc         memory          README-wxWidgets
config.cc        gui                misc            TESTFORM.txt
config.guess     host               msrs.def        TODO
config.h.in      install-sh         osdep.cc        win32_enh_dbg.rc
config.sub       instrument         osdep.h         win32res.rc
configure        iodev              param_names.h   wxbochs.rc
configure.in     LICENSE            PARAM_TREE.txt
COPYING          load32bitOShack.cc patches
a@ubuntu:~/Downloads/bochs-2.6.2$ ./configure -enable-gdb-stub -with-nogui
checking build system type... i686-pc-linux-gnu
checking host system type... i686-pc-linux-gnu
checking target system type... i686-pc-linux-gnu
checking if you are configuring for another platform... no
checking for standard CFLAGS on this platform
```

3. PintOS install in Virtual Machine

a) Go to home directory and install PintOS by typing below

- **wget http://web.Stanford.edu/class/cs140/projects/pintos/pintos.tar.gz**
- **tar xvf pinots.tar.gz**

```
a@ubuntu:~$ ls
Desktop    Downloads         fontconfig  Pictures  Templates
Documents  examples.desktop  Music       Public    Videos
a@ubuntu:~$ pwd
/home/a
a@ubuntu:~$ wget http://www.stanford.edu/class/cs140/projects/pintos/pintos.tar.g
z
```

```
a@ubuntu:~$ ls
Desktop    Downloads         fontconfig  Pictures  pintos.tar.gz  Templates
Documents  examples.desktop  Music       pintos    Public         Videos
a@ubuntu:~$ cd pintos
a@ubuntu:~/pintos$ ls
src
a@ubuntu:~/pintos$ cd src
a@ubuntu:~/pintos/src$ ls
devices   lib          Makefile         Makefile.userprog  threads   vm
examples  LICENSE      Makefile.build   misc               userprog
filesys   Make.config  Makefile.kernel  tests              utils
a@ubuntu:~/pintos/src$
```

3. PintOS install in Virtual Machine

b) Environment Settings (PATH setting)

1) Type below to open vi editor
   - **vi ~/.bashrc**
2) Put following at the end of the file, then save and close the file (name should be your admin name)
   - **export PATH="$PATH:/home/AdminName/pintos/src/utils"**
3) Let Ubuntu take care of the path by executing
   - **source ~/.bashrc**

```
if [ -f /etc/bash_completion ] && ! shopt -oq posix; then
    . /etc/bash_completion
fi
export PATH="$PATH:/home/a/pintos/src/utils"
:wq
```

c) Environment Settings (PintOS setting)

1) Open 'pintos' file using vi
   - **vi /home/name/pintos/src/utils/pintos**
2) Change path to kernel.bin in the line 259 as follows:
   - **/home/name/pintos/src/threads/build/kernel.bin**
3) Open 'Pintos.pm' file using vi
   - **vi /home/name/pintos/src/utils/Pintos.pm**
4) Change path to kernel.bin in the line 362 as follows:
   - **/home/name/pintos/src/threads/build/loader.bin**

```
258     if (!exists $parts{KERNEL}) {
259         my $name = find_file ('/home/a/pintos/src/threads/build/kernel.bi
n');
260         die "Cannot find kernel\n" if !defined $name;
261         do_set_part ('KERNEL', 'file', $name);
262     }
```

```
360 sub read_loader {
361     my ($name) = @_;
362     $name = find_file ("/home/a/pintos/src/threads/build/loader.bin") if
!defined $name;
363     die "Cannot find loader\n" if !defined $name;
364
```

3. PintOS install in Virtual Machine

d) Environment Settings (PintOS emulator setting for qemu)

1) Open ‘pintos-gdb’ file using vi
   - **cd /home/name/pintos/src/utils**
   - **vi /home/name/pintos/src/utils/pintos-gdb**

2) Change GDBMACROS as follows:
   - **/home/name/pintos/src/misc/gdb-macros**

```
#! /bin/sh

# Path to GDB macros file.  Customize for your site.
GDBMACROS=/home/a/pintos/src/misc/gdb-macros
```

3) Compile utils
   - **make**

4) Open ‘Makefile’ file using vi
   - **vi /home/name/pintos/src/utils/Makefile**

3. PintOS install in Virtual Machine

d) Environment Settings (PintOS emulator setting for qemu)

5) Edit Makefile and change LDFLAGS = -lm to LDLIBS = -lm

```
all: setitimer-helper squish-pty squish-unix

CC = gcc
CFLAGS = -Wall -W
LDLIBS = -lm
setitimer-helper: setitimer-helper.o
squish-pty: squish-pty.o
squish-unix: squish-unix.o
```

6) Edit Make.vars in ***/src/threads*** and change --bochs in SIMULATOR to --qemu

```
a@ubuntu:~/pintos/src/utils$ cd ..
a@ubuntu:~/pintos/src$ ls
devices   lib          Makefile         Makefile.userprog  threads   vm
examples  LICENSE      Makefile.build   misc               userprog
filesys   Make.config  Makefile.kernel  tests              utils
a@ubuntu:~/pintos/src$ cd threads
a@ubuntu:~/pintos/src/threads$ ls
flags.h      interrupt.h   kernel.lds.S  Make.vars  palloc.h  switch.S  thread.h
init.c       intr-stubs.h  loader.h      malloc.c   pte.h     synch.c   vaddr.h
init.h       intr-stubs.S  loader.S      malloc.h   start.S   synch.h
interrupt.c  io.h          Makefile      palloc.c   switch.h  thread.c
a@ubuntu:~/pintos/src/threads$ vi Make.vars
```

7) Compile pintos kernel

- **make**

```
# -*- makefile -*-

kernel.bin: DEFINES =
KERNEL_SUBDIRS = threads devices lib lib/kernel $(TEST_SUBDIRS)
TEST_SUBDIRS = tests/threads
GRADING_FILE = $(SRCDIR)/tests/threads/Grading
SIMULATOR = --qemu
```

3. PintOS install in Virtual Machine

d) Environment Settings (PintOS emulator setting for qemu)

8) Edit /pintos/src/utils/pintos as follows:

- **vi home/name/pintos/src/utils/pintos**
- Line 103: **$sim = “bochs” -> $sim = “qemu”**

```
103     $sim = "qemu" if !defined $sim;
```

- Line 623: **my (@cmd) = (‘qemu’) -> my (@cmd) = (‘qemu-system-i386’)**

```
623     my (@cmd) = ('qemu-system-i386');
```

9) Build pinots for the project

- **make clean**
- **make**

3. PintOS install in Virtual Machine

e) Test pintos

0) Change directory

- **cd /home/name/pintos/src/threads**

1) Execute following

- **pintos -q run alarm-multiple**

```
(alarm-multiple) thread 0: duration=10, iteration=7, product=70
(alarm-multiple) thread 1: duration=20, iteration=4, product=80
(alarm-multiple) thread 3: duration=40, iteration=2, product=80
(alarm-multiple) thread 2: duration=30, iteration=3, product=90
(alarm-multiple) thread 4: duration=50, iteration=2, product=100
(alarm-multiple) thread 1: duration=20, iteration=5, product=100
(alarm-multiple) thread 3: duration=40, iteration=3, product=120
(alarm-multiple) thread 1: duration=20, iteration=6, product=120
(alarm-multiple) thread 2: duration=30, iteration=4, product=120
(alarm-multiple) thread 1: duration=20, iteration=7, product=140
(alarm-multiple) thread 4: duration=50, iteration=3, product=150
(alarm-multiple) thread 2: duration=30, iteration=5, product=150
(alarm-multiple) thread 3: duration=40, iteration=4, product=160
(alarm-multiple) thread 2: duration=30, iteration=6, product=180
(alarm-multiple) thread 3: duration=40, iteration=5, product=200
(alarm-multiple) thread 4: duration=50, iteration=4, product=200
(alarm-multiple) thread 2: duration=30, iteration=7, product=210
(alarm-multiple) thread 3: duration=40, iteration=6, product=240
(alarm-multiple) thread 4: duration=50, iteration=5, product=250
(alarm-multiple) thread 3: duration=40, iteration=7, product=280
(alarm-multiple) thread 4: duration=50, iteration=6, product=300
(alarm-multiple) thread 4: duration=50, iteration=7, product=350
(alarm-multiple) end
Execution of 'alarm-multiple' complete.
Timer: 580 ticks
Thread: 0 idle ticks, 580 kernel ticks, 0 user ticks
Console: 2954 characters output
Keyboard: 0 keys pressed
Powering off...
a@ubuntu:~/pintos/src/threads$
```

4. Alarm Clock

A. Reference

- https://web.stanford.edu/class/cs140/projects/pintos/pintos.html

B. Description

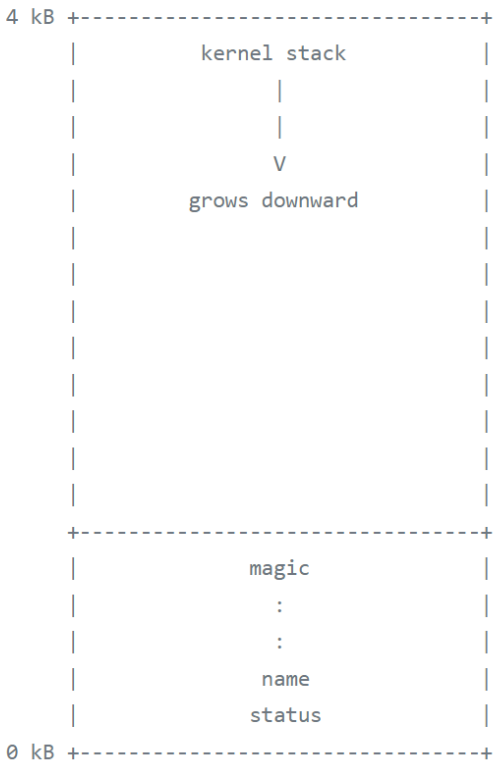
- In this assignment, your job is to extend the functionality of thread system to gain a better understanding of **synchronization** problems, and you will start from the basic function of timer.

- You will work primarily in the threads directory “~/pintos/src/threads” for this assignment, with some work in the devices directory “~ /pintos/src/devices” on the side. **You should compile in the threads directory “~ /pintos/src/threads”.**

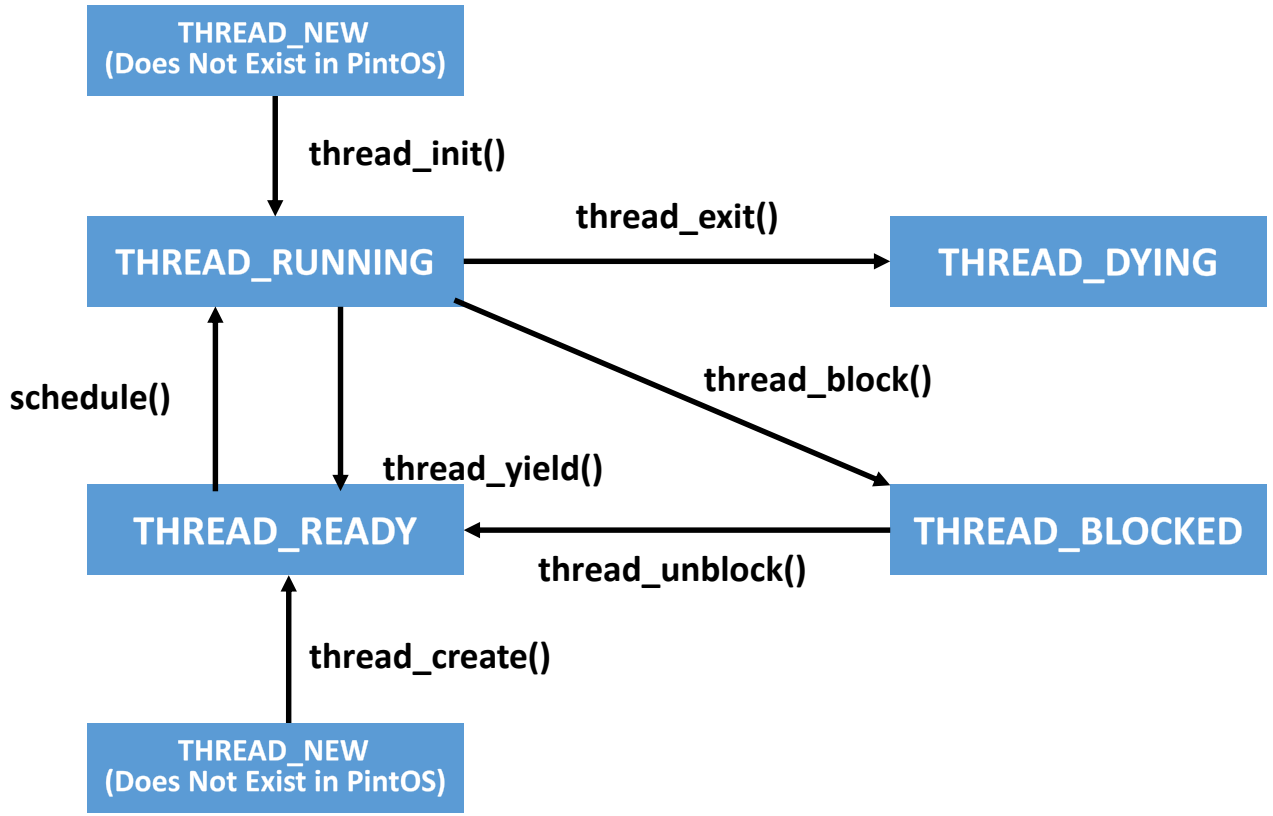
4. Alarm Clock

B. Description

- In PintOS, each process includes only a single thread and uploaded to memory as a 4KB page



Thread Definition in thread.h



4. Alarm Clock

B. Description

- Let's see how PintOS runs by looking at init.c

```
/* Pintos main program. */
Int main (void) {
  char **argv;

  /* Clear BSS. */
  bss_init ();

  /* Break command line into arguments and parse options. */
  argv = read_command_line ();
  argv = parse_options (argv);

  /* Initialize ourselves as a thread so we can use locks,
     then enable console locking. */
  thread_init ();
  console_init ();

/* Greet user. */
  printf ("Pintos booting with %'"PRIu32" kB RAM...\n",
          init_ram_pages * PGSIZE / 1024);

  /* Initialize memory system. */
  palloc_init (user_page_limit);
  malloc_init ();
  paging_init ();

  /* Segmentation. */
#ifdef USERPROG
  tss_init ();
  gdt_init ();
#endif

  /* Initialize interrupt handlers. */
  intr_init ();
  timer_init ();
  kbd_init ();
  input_init ();
#ifdef USERPROG
  exception_init ();
  syscall_init ();
#endif

  /* Start thread scheduler and enable interrupts. */
  thread_start ();
  serial_init_queue ();
  timer_calibrate ();

#ifdef FILESYS
  /* Initialize file system. */
  ide_init ();
  locate_block_devices ();
  filesys_init (format_filesys);
#endif

  printf ("Boot complete.\n");

  /* Run actions specified on kernel command line. */
  run_actions (argv);

  /* Finish up. */
  shutdown ();
  thread_exit ();
}
```

4. Alarm Clock

C. Requirements

- Reimplement ***timer_sleep()*** in “~ /pintos/src/devices/timer.c”.
  - Current implementation uses the mechanism of "busy waits,“
    -> It spins a calling thread in a loop while checking the current time and calling *thread_yield()* until enough time elapses.
  - **Reimplement the *timer_sleep()* to avoid busy waiting.**

- *timer_sleep(int64_t ticks)*
  - Suspends execution of the calling thread until the system time elapses at least x timer ticks.
  - The calling thread does not need to wake up until exactly x ticks pass.
  - Just put the calling thread on the ready queue and dequeue it after they have waited for the right amount of time.

- Another sleep functions in PintOS
  - *timer_msleep()*, *timer_usleep()*, and *timer_nsleep()* forces a calling thread sleep a specific number of milliseconds, microseconds, or nanoseconds, respectively.
  - These functions will call *timer_sleep()* automatically when they need to do it.
  - **You do not need to modify the three sleep functions.**

4. Alarm Clock

D. Problem manual

- Current state
  - A calling thread turns over its context to timer sleep
  - timer_sleep() gets the current time(in ticks)
  - Assert the calling thread if the interrupt level is on
  - Until the aforementioned time elapses, kernel yield the thread

  -> **repeat this EVERY tick**
  - thread_yield()
    - Get the current thread
    - Put this thread to the ready state
    - Schedule the ready threads again
    - Set the interrupt level disable

```
void
timer_sleep (int64_t ticks)
{
  int64_t start = timer_ticks ();

  ASSERT (intr_get_level () == INTR_ON);
  while (timer_elapsed (start) < ticks)
    thread_yield ();
}
```

```
void
thread_yield (void)
{
  struct thread *cur = thread_current ();
  enum intr_level old_level;

  ASSERT (!intr_context ());

  old_level = intr_disable ();
  if (cur != idle_thread)
    list_push_back (&ready_list, &cur->elem);
  cur->status = THREAD_READY;
  schedule ();
  intr_set_level (old_level);
}
```

- Main issue
  - Busy waiting (repeating yielding every tick)
- How to solve the problem?

4. Alarm Clock

D. Problem manual

```
a@ubuntu:~/pintos/src/threads$ sudo ma
```

```
gcc -c ../../devices/timer.c -o device
tector -nostdinc -I../.. -I../../lib -
pes -Wmissing-prototypes -Wsystem-head
../../devices/timer.c: In function ‘ti
../../devices/timer.c:96: error: expec
make[1]: *** [devices/timer.o] Error 1
make[1]: Leaving directory `/home/a/pi
make: *** [all] Error 2
```

```
1.o tests/threads/mlfqs-load-60.o test
fqs-recent-1.o tests/threads/mlfqs-fai
objcopy -R .note -R .comment -S kernel
make[1]: Leaving directory `/home/a/pi
a@ubuntu:~/pintos/src/threads$
```

```
(alarm-multiple) thread 1: duration=20, iteration=5, product=100
(alarm-multiple) thread 3: duration=40, iteration=3, product=120
(alarm-multiple) thread 2: duration=30, iteration=4, product=120
(alarm-multiple) thread 1: duration=20, iteration=6, product=120
(alarm-multiple) thread 1: duration=20, iteration=7, product=140
(alarm-multiple) thread 4: duration=50, iteration=3, product=150
(alarm-multiple) thread 2: duration=30, iteration=5, product=150
(alarm-multiple) thread 3: duration=40, iteration=4, product=160
(alarm-multiple) thread 2: duration=30, iteration=6, product=180
(alarm-multiple) thread 4: duration=50, iteration=4, product=200
(alarm-multiple) thread 3: duration=40, iteration=5, product=200
(alarm-multiple) thread 2: duration=30, iteration=7, product=210
(alarm-multiple) thread 3: duration=40, iteration=6, product=240
(alarm-multiple) thread 4: duration=50, iteration=5, product=250
(alarm-multiple) thread 3: duration=40, iteration=7, product=280
(alarm-multiple) thread 4: duration=50, iteration=6, product=300
(alarm-multiple) thread 4: duration=50, iteration=7, product=350
(alarm-multiple) end
Execution of 'alarm-multiple' complete.
Timer: 929 ticks
Thread: 550 idle ticks, 382 kernel ticks, 0 user ticks
Console: 2952 characters output
Keyboard: 0 keys pressed
Powering off...
```

4. Alarm Clock

E. Report: Problem, Design, and Implementation

- Please submit a report (limit 5 pages, Korean/English) that includes:
  - Student IDs, names, and team number
  - Timer solution should be described in the form of
    - Problem definition
    - Policy and algorithm design)
    - Mechanism (implementation)
      - What you have added to solve the problem and modified file names