CS318 Pintos Project Lab0 Overview

William Wang (xwill@bu.edu)

Outline

- Administrivia
- Lab overview
- Environment Setup
- Dev Tool
- Tips

Administrivia

Lab0 deadline: 09/19 23:59PM Friday, individually

score = code (70%) + design doc (30%)

Submit through GradeScope (only lab 0)

Outline

- Administrivia
- Lab overview
- Environment Setup
- Dev Tool
- Tips

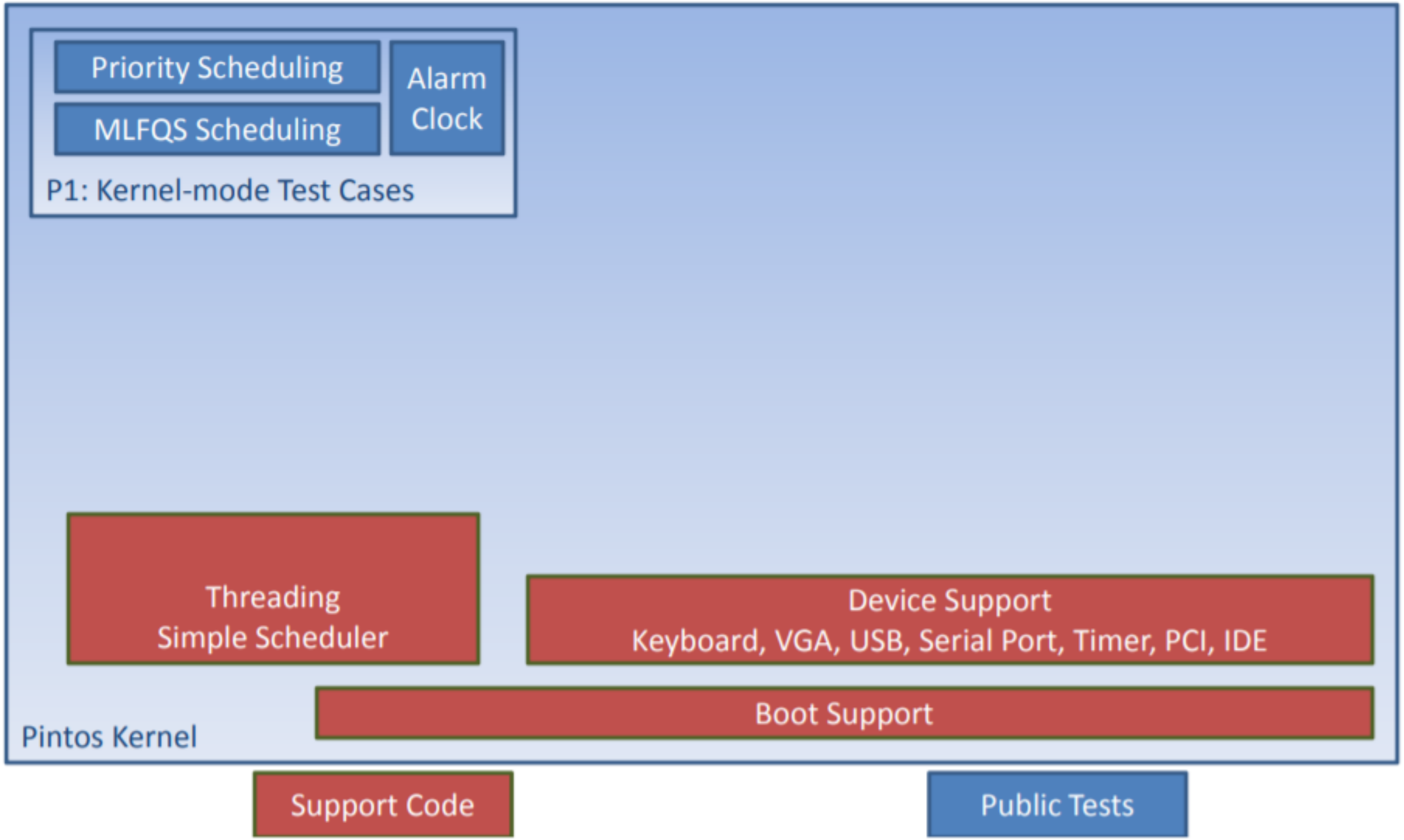
What Is Pintos?

- Pintos is a teaching operating system for 80x86
  - Developed in 2005 for Stanford’s CS 140 OS class
  - Small enough so entire code can be read and understood by students

- Pintos supports kernel threads, virtual memory, user programs, and file system
  - Premature or incomplete

In this course project, you will improve all of these areas of Pintos to make it complete

Pintos Kernel(Pre Project)



Priority Scheduling

MLFQS Scheduling

Alarm Clock

P1: Kernel-mode Test Cases

Threading
Simple Scheduler

Device Support
Keyboard, VGA, USB, Serial Port, Timer, PCI, IDE

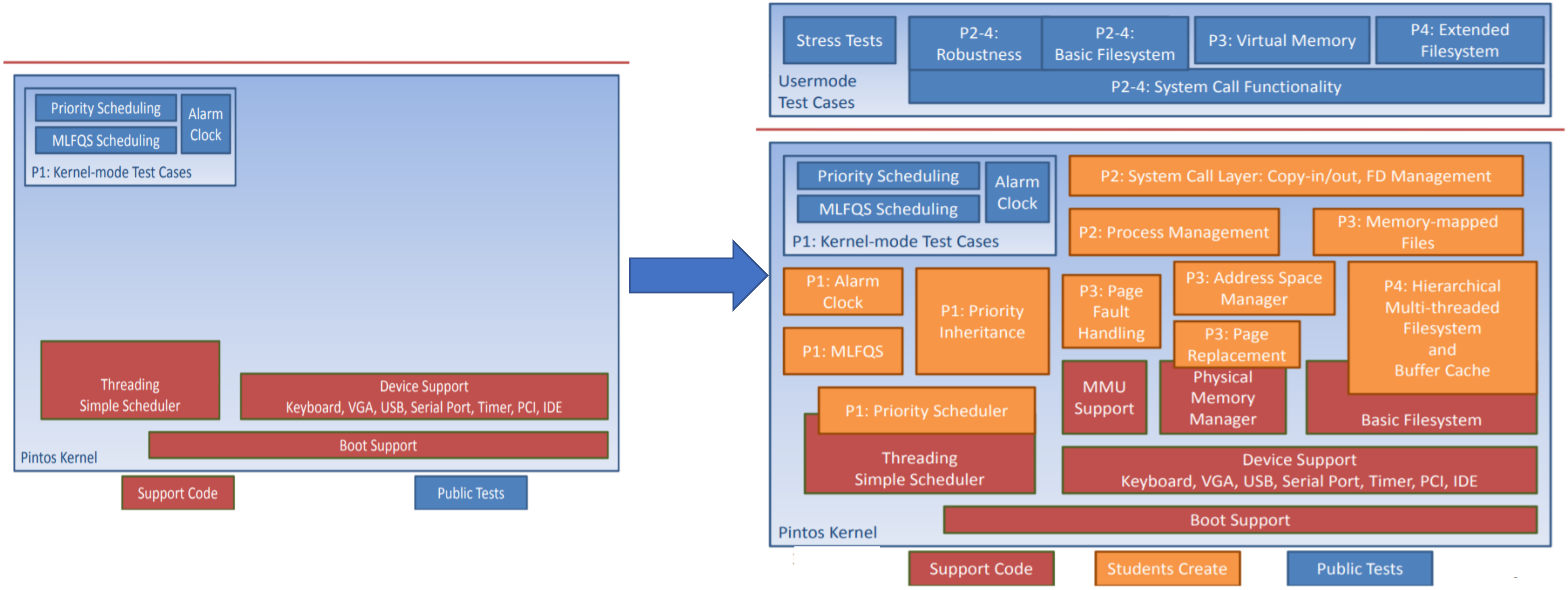
Boot Support

Pintos Kernel

Support Code

Public Tests

Pintos Kernel(Post Project)



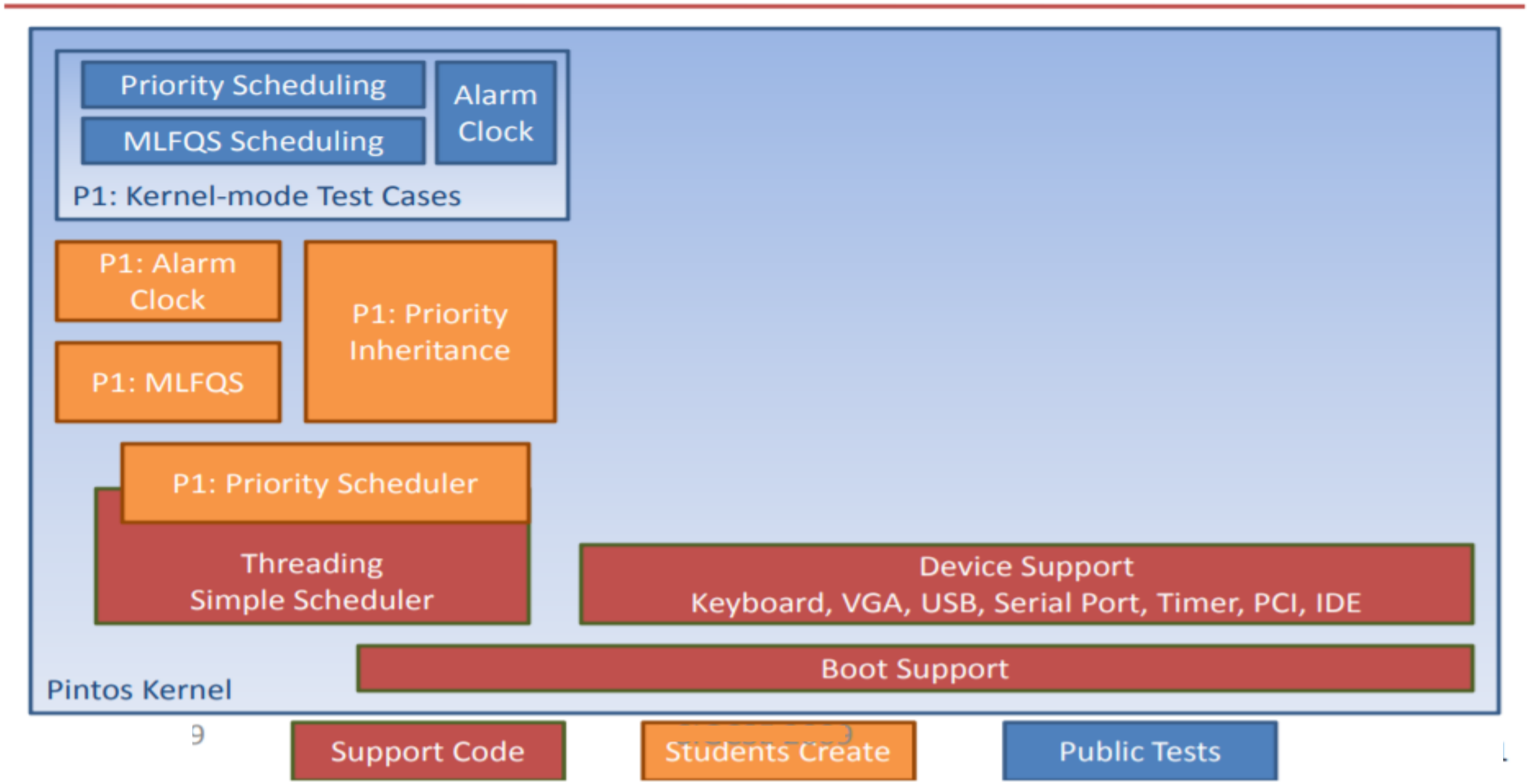
Lab 0 Overview

- Lab 0 is a warm-up exercise
  - Preparing you for the later Pintos projects
- In Lab 0, you will:
  - Install and boot Pintos
  - Go through the PC Bootstrap
  - Learn how to debug Pintos in QEMU and Bochs
  - Add a tiny kernel monitor to Pintos

https://yigonghu.github.io/EC440/fall25/projects/lab0

Lab 1 Overview

- Extending the functionality of Pintos thread system



Priority Scheduling

MLFQS Scheduling

Alarm Clock

P1: Kernel-mode Test Cases

P1: Alarm Clock

P1: MLFQS

P1: Priority Inheritance

P1: Priority Scheduler

Threading
Simple Scheduler

Device Support
Keyboard, VGA, USB, Serial Port, Timer, PCI, IDE

Boot Support

Pintos Kernel

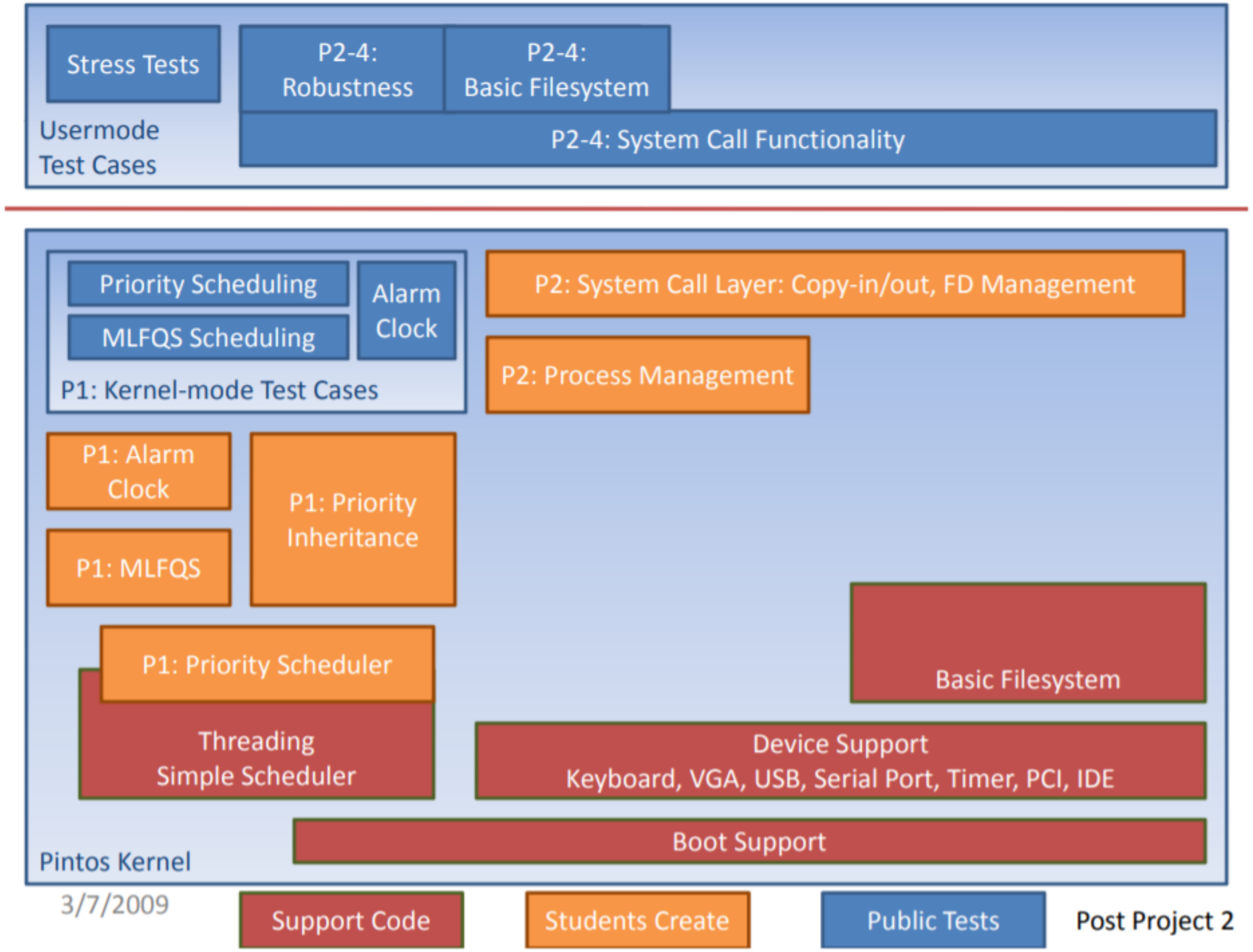
Support Code

Students Create

Public Tests

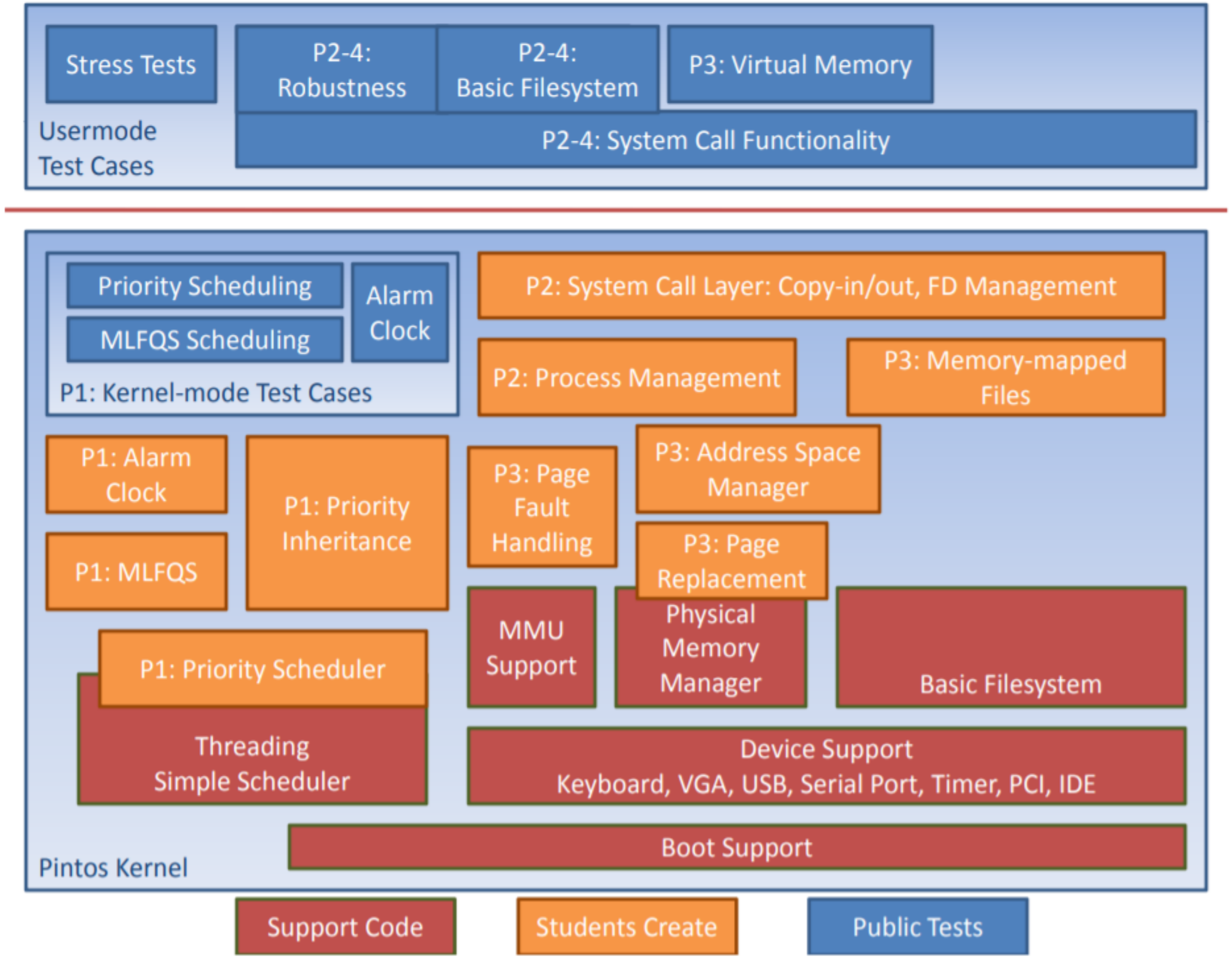
Lab 2 Overview

- Enable user programs to interact with the OS via system calls



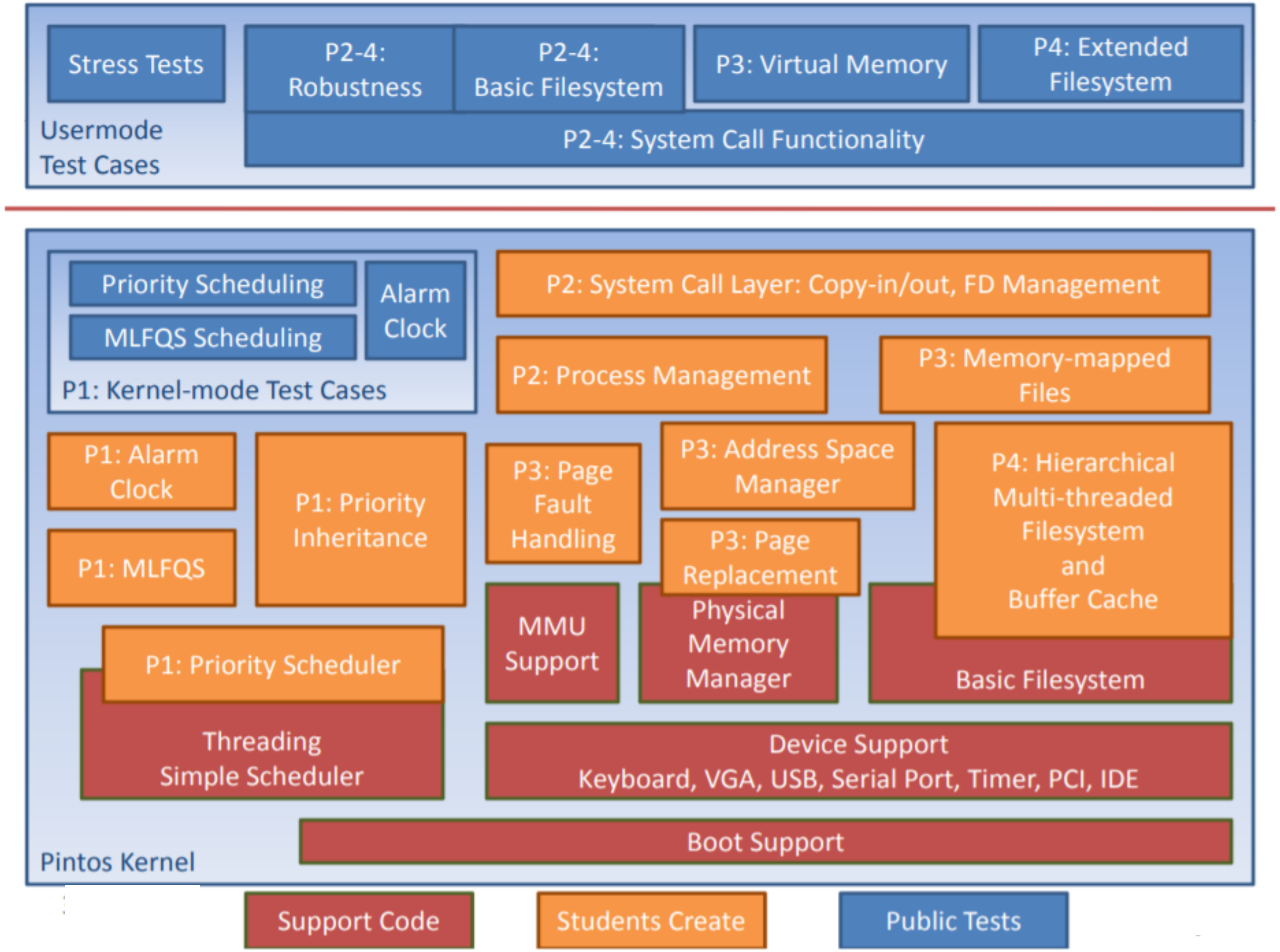
Lab 3 Overview

- Implement the virtual memory management

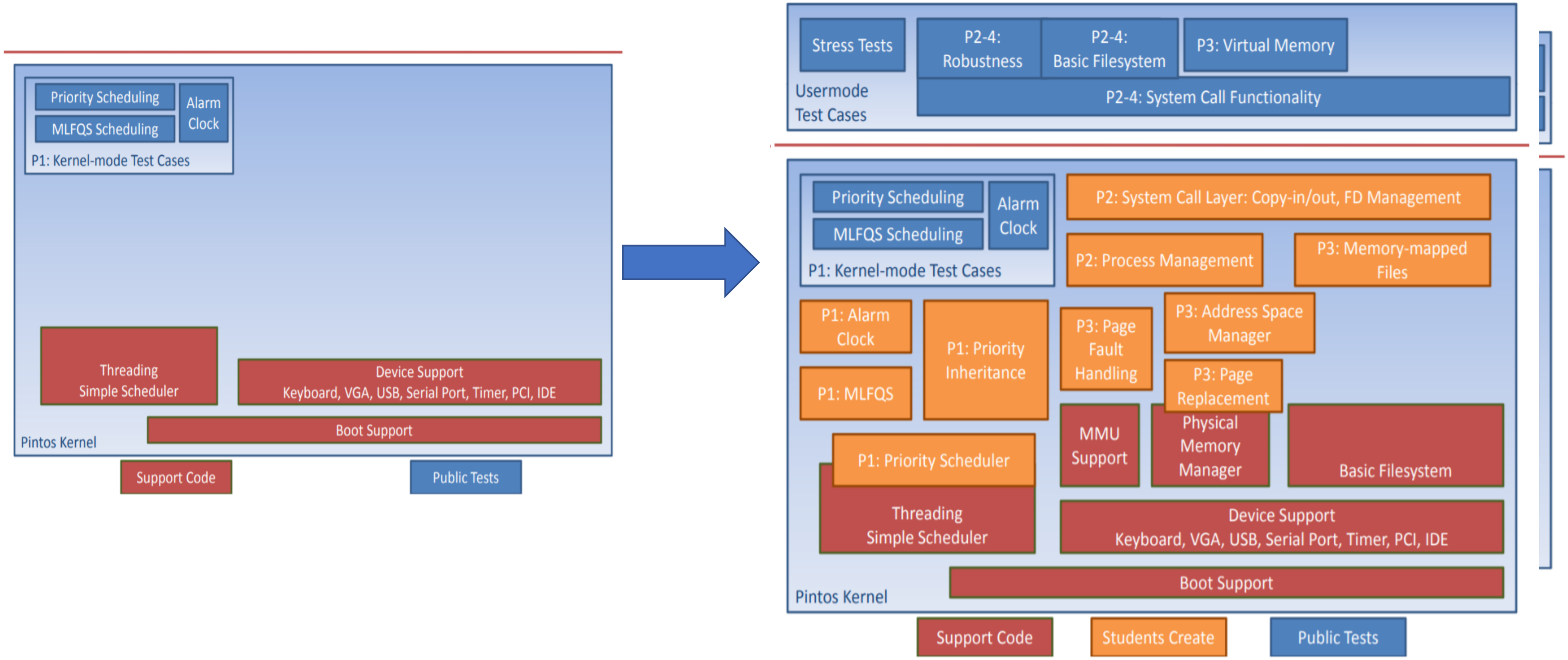


Lab 4 Overview

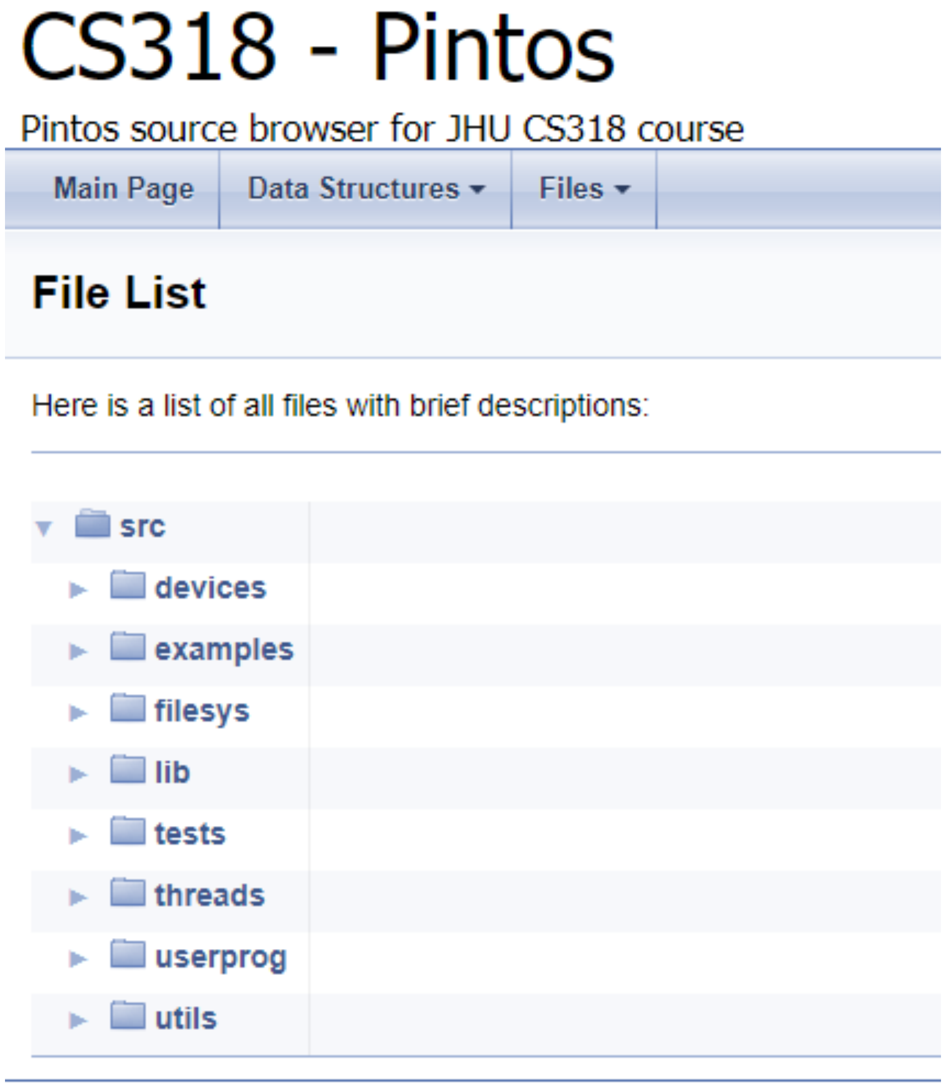
- Extending basic filesystem to hierarchical filesystem



Pintos Kernel(Post Project)



Pintos Source Code Overview



CS318 - Pintos

Pintos source browser for JHU CS318 course

File List

Here is a list of all files with brief descriptions:

- ▼ src
  - ▶ devices
  - ▶ examples
  - ▶ filesys
  - ▶ lib
  - ▶ tests
  - ▶ threads
  - ▶ userprog
  - ▶ utils

https://jhu-cs318.github.io/pintos-doxygen/html/files.html

Pintos Source Tree

- threads/
  - Source code for the base kernel, which you will modify starting in project 1
- userprog/
  - Source code for the user program loader, which you will modify starting with project 2
- vm/
  - An almost empty directory. You will implement virtual memory here in project 3.
- filesys/
  - Source code for a basic file system. You will use this file system starting with project 2, but you will not modify it until project 4

Pintos Source Tree

- devices/
  - Source code for I/O device interfacing: keyboard, timer, disk, etc. You will modify the timer implementation in project 1.
- lib/
  - An implementation of a subset of the standard C library.
- tests/
  - All the test cases for each project
- examples/
  - Example user programs for use starting with project 2

Pintos Source Tree

- misc/
- utils/
    - These files may come in handy if you decide to try working with Pintos on your own machine. Otherwise, you can ignore them

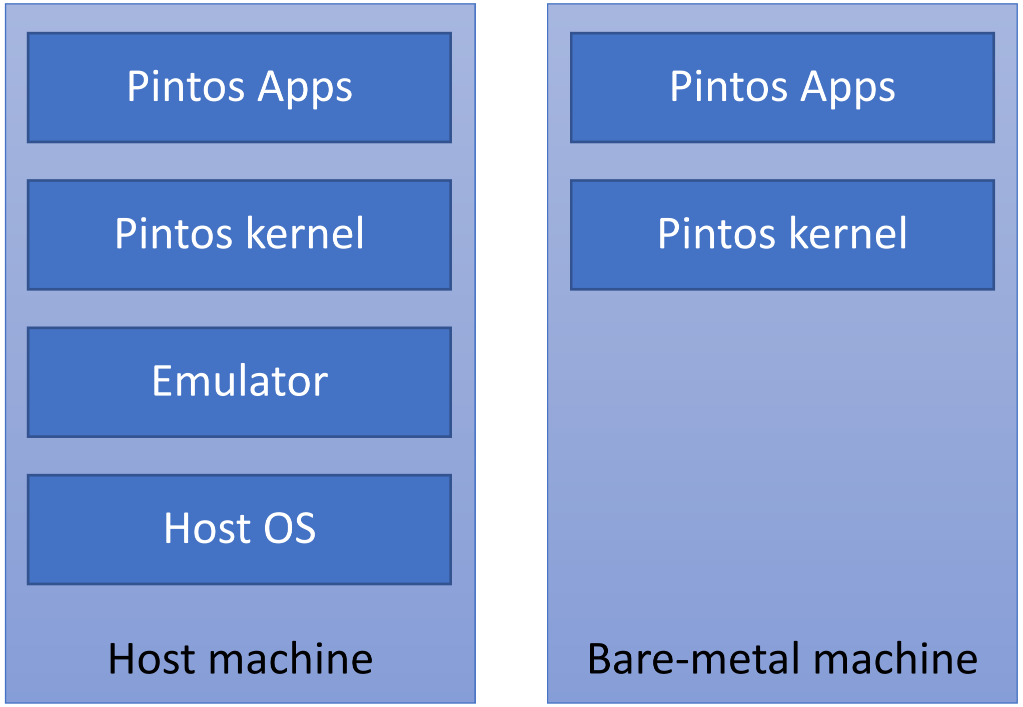
Outline

- Administrivia
- Lab 0 overview
- Environment Setup
- Dev Tool
- Tips

Pintos Running Environment

- Pintos can run and debug in
  - Emulated environment
  - Real hardware

In this course, we will use emulated environment.



Environment

- Required Tools for PintOS
  - 80x86 cross-compiler toolchain for 32-bit architecture
    - C compiler, assembler, linker, and debugger.
  - x86 emulator
    - QEMU or Bochs

- Working Environment
  - SCC lab machine
  - Pre-made docker/utm image
  - Your own machine

Use premade docker/virtual machine image

- Premade containers already have toolchains installed
- Environments are setup in advance, use out-of-the-box
- Useful Resource
  - Docker image

    ```
    docker run -it —name pintos buec440/pintos bash
    ```

  - UTM image

    To be added

Installing Pintos on You Own Machine

- You may want to work on your own machines to be more productive
- Need to build the toolchain
- Useful Links
  - Build script (tested on Mac, Ubuntu and Fedora)
    pintos/src/misc/toolchain-build.sh
  - Installation guide
    https://yigonghu.github.io/EC440/fall25/projects/setup

Environment setting on SCC Lab Machine

- CS lab machines already have required tools installed
- To include tools, you need source the ec440 tools
  - Change working directory into /projectnb/ec440/projects and:

```
source envsetup.sh
```

- Build pintos
  - git clone https://github.com/yigonghu/ec440-pintos.git
  - cd pintos/src/threads
  - make

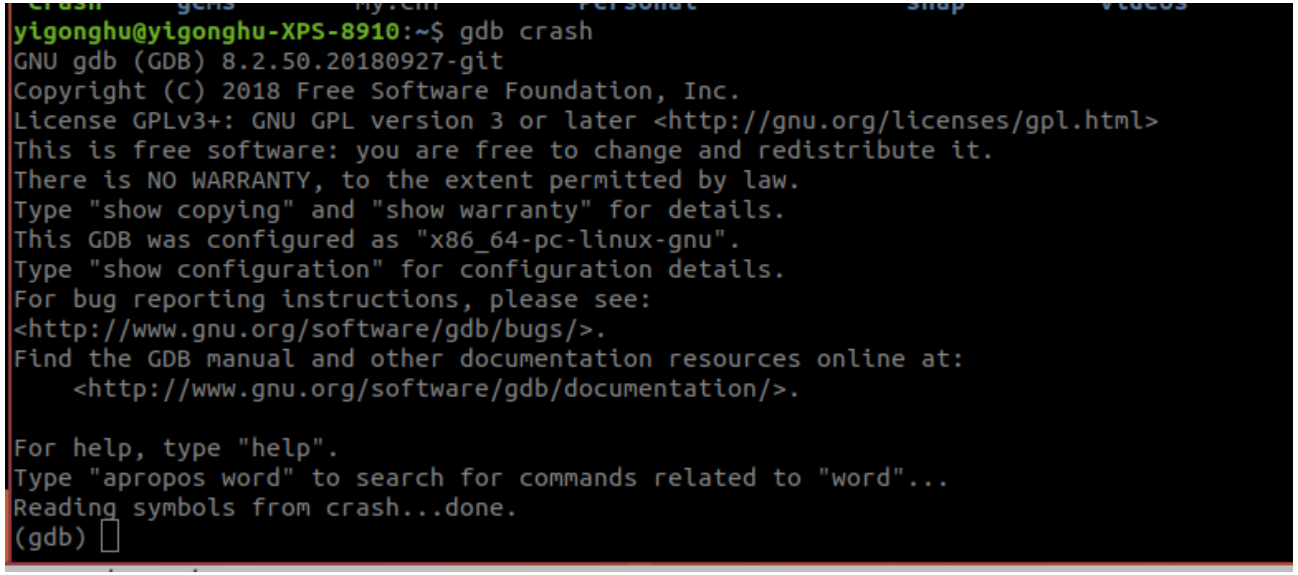
Outline

- Administrivia
- Lab 0 overview
- Environment Setup
- Dev Tool
- Tips

GDB

- “GNU Debugger”
- A debugger for several languages, including C and C++
- The Pintos uses GDB as the default debugging tool.
- Online manual

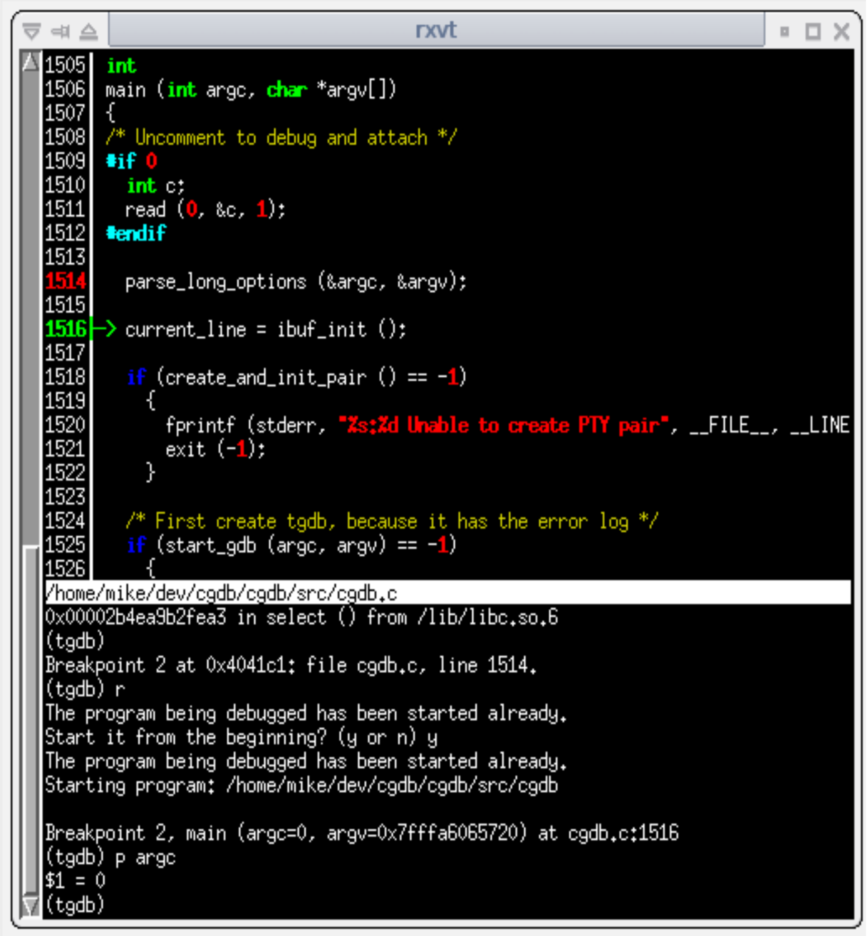
  https://sourceware.org/gdb/current/onlinedocs/gdb/



```
yigonghu@yigonghu-XPS-8910:~$ gdb crash
GNU gdb (GDB) 8.2.50.20180927-git
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from crash...done.
(gdb) 
```

CGDB

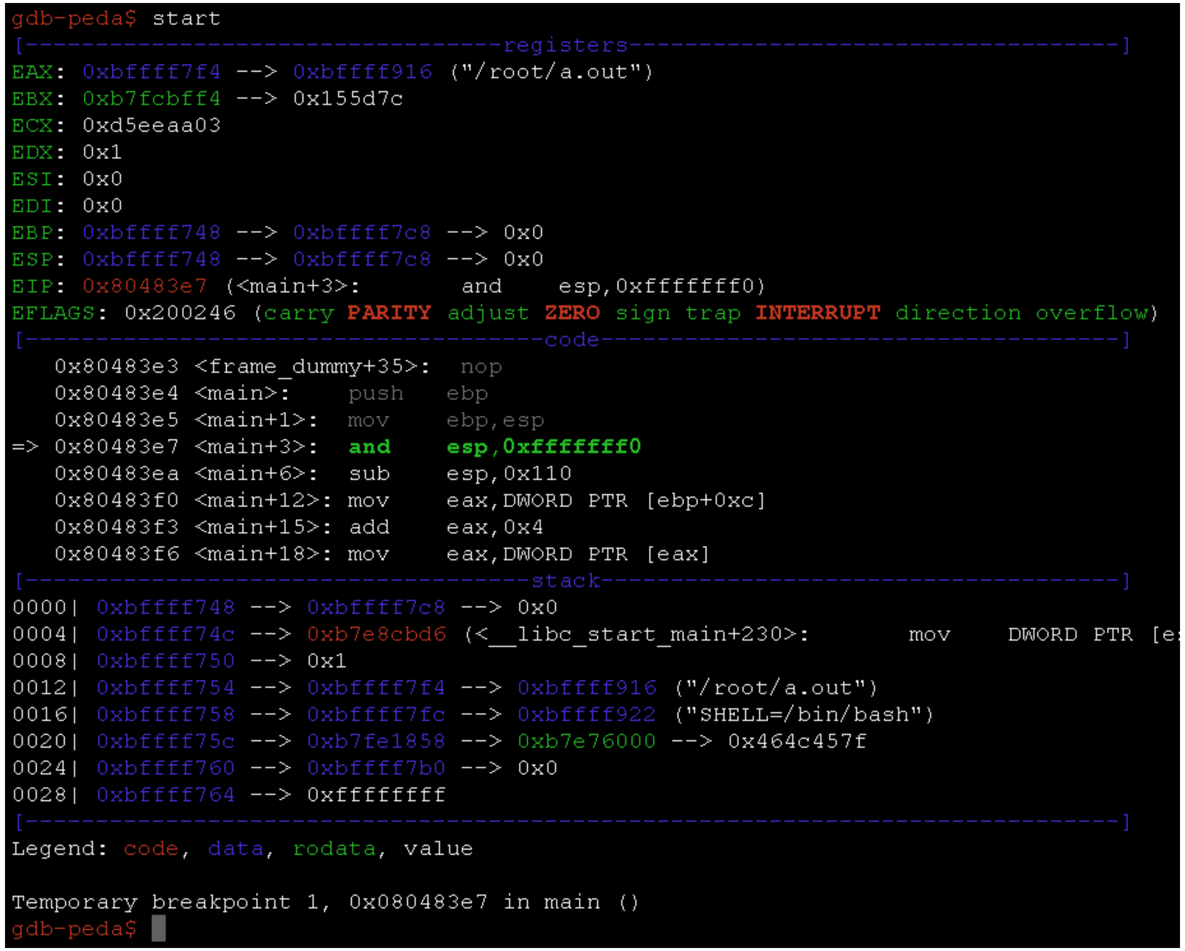
- CGDB
  - A lightweight curses interface to GDB
  - Standard gdb console
  - A split screen view that displays the source code
- Strongly recommend using CGDB to debug Pintos
- Reference
  - https://cgdb.github.io/



```
1505| int
1506| main (int argc, char *argv[])
1507| {
1508| /* Uncomment to debug and attach */
1509| #if 0
1510|   int c;
1511|   read (0, &c, 1);
1512| #endif
1513|
1514|   parse_long_options (&argc, &argv);
1515|
1516|-> current_line = ibuf_init ();
1517|
1518|   if (create_and_init_pair () == -1)
1519|     {
1520|       fprintf (stderr, "%s:%d Unable to create PTY pair", __FILE__, __LINE
1521|       exit (-1);
1522|     }
1523|
1524|   /* First create tgdb, because it has the error log */
1525|   if (start_gdb (argc, argv) == -1)
1526|     {
/home/mike/dev/cgdb/cgdb/src/cgdb.c
0x00002b4ea9b2fea3 in select () from /lib/libc.so.6
(tgdb)
Breakpoint 2 at 0x4041c1: file cgdb.c, line 1514.
(tgdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y
The program being debugged has been started already.
Starting program: /home/mike/dev/cgdb/cgdb/src/cgdb

Breakpoint 2, main (argc=0, argv=0x7fffa6065720) at cgdb.c:1516
(tgdb) p argc
$1 = 0
(tgdb)
```

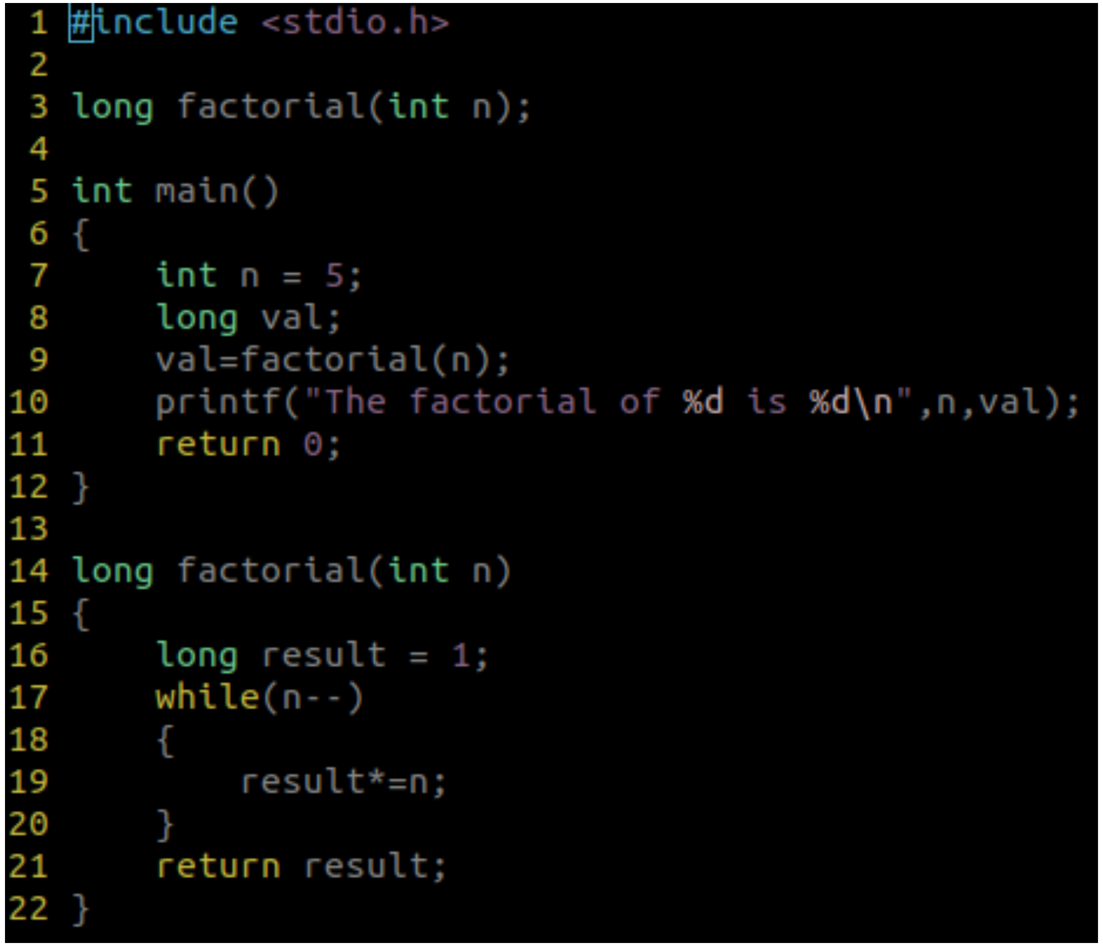
PEDA

- PEDA
  - ○ An extension for GDB written in Python
  - ○ Standard gdb console
  - ○ A split screen view that displays registers, assembly and stack
- Reference
  https://github.com/longld/peda



```
gdb-peda$ start
[----------------------------------registers-----------------------------------]
EAX: 0xbffff7f4 --> 0xbffff916 ("/root/a.out")
EBX: 0xb7fcbff4 --> 0x155d7c
ECX: 0xd5eeaa03
EDX: 0x1
ESI: 0x0
EDI: 0x0
EBP: 0xbffff748 --> 0xbffff7c8 --> 0x0
ESP: 0xbffff748 --> 0xbffff7c8 --> 0x0
EIP: 0x80483e7 (<main+3>:       and    esp,0xfffffff0)
EFLAGS: 0x200246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x80483e3 <frame_dummy+35>:  nop
   0x80483e4 <main>:    push   ebp
   0x80483e5 <main+1>:  mov    ebp,esp
=> 0x80483e7 <main+3>:  and    esp,0xfffffff0
   0x80483ea <main+6>:  sub    esp,0x110
   0x80483f0 <main+12>: mov    eax,DWORD PTR [ebp+0xc]
   0x80483f3 <main+15>: add    eax,0x4
   0x80483f6 <main+18>: mov    eax,DWORD PTR [eax]
[------------------------------------stack-------------------------------------]
0000| 0xbffff748 --> 0xbffff7c8 --> 0x0
0004| 0xbffff74c --> 0xb7e8cbd6 (<__libc_start_main+230>:       mov    DWORD PTR [e
0008| 0xbffff750 --> 0x1
0012| 0xbffff754 --> 0xbffff7f4 --> 0xbffff916 ("/root/a.out")
0016| 0xbffff758 --> 0xbffff7fc --> 0xbffff922 ("SHELL=/bin/bash")
0020| 0xbffff75c --> 0xb7fe1858 --> 0xb7e76000 --> 0x464c457f
0024| 0xbffff760 --> 0xbffff7b0 --> 0x0
0028| 0xbffff764 --> 0xffffffff
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Temporary breakpoint 1, 0x080483e7 in main ()
gdb-peda$ 
```

How to Use CGDB to Debug A Toy Program?

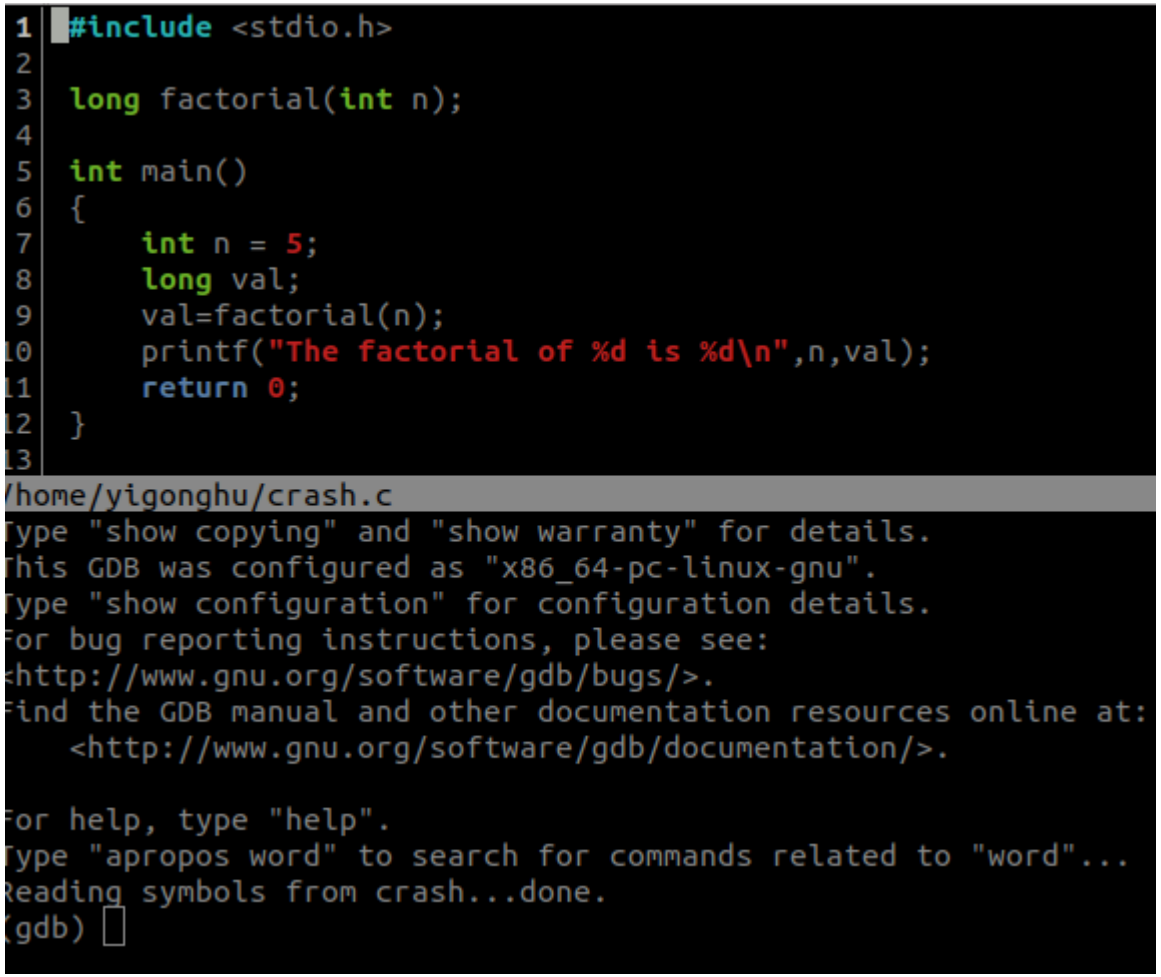


```c
#include <stdio.h>

long factorial(int n);

int main()
{
    int n = 5;
    long val;
    val=factorial(n);
    printf("The factorial of %d is %d\n",n,val);
    return 0;
}

long factorial(int n)
{
    long result = 1;
    while(n--)
    {
        result*=n;
    }
    return result;
}
```

The given code computes the factorial of a number erroneously. The program always outputs 0, regardless of the input

Starting Up CGDB

- To run the cgdb, just first try “cgdb <filename>” (build with `gcc -g`)



```c
#include <stdio.h>

long factorial(int n);

int main()
{
    int n = 5;
    long val;
    val=factorial(n);
    printf("The factorial of %d is %d\n",n,val);
    return 0;
}

```

```
/home/yigonghu/crash.c
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from crash...done.
(gdb) 
```

Running Program

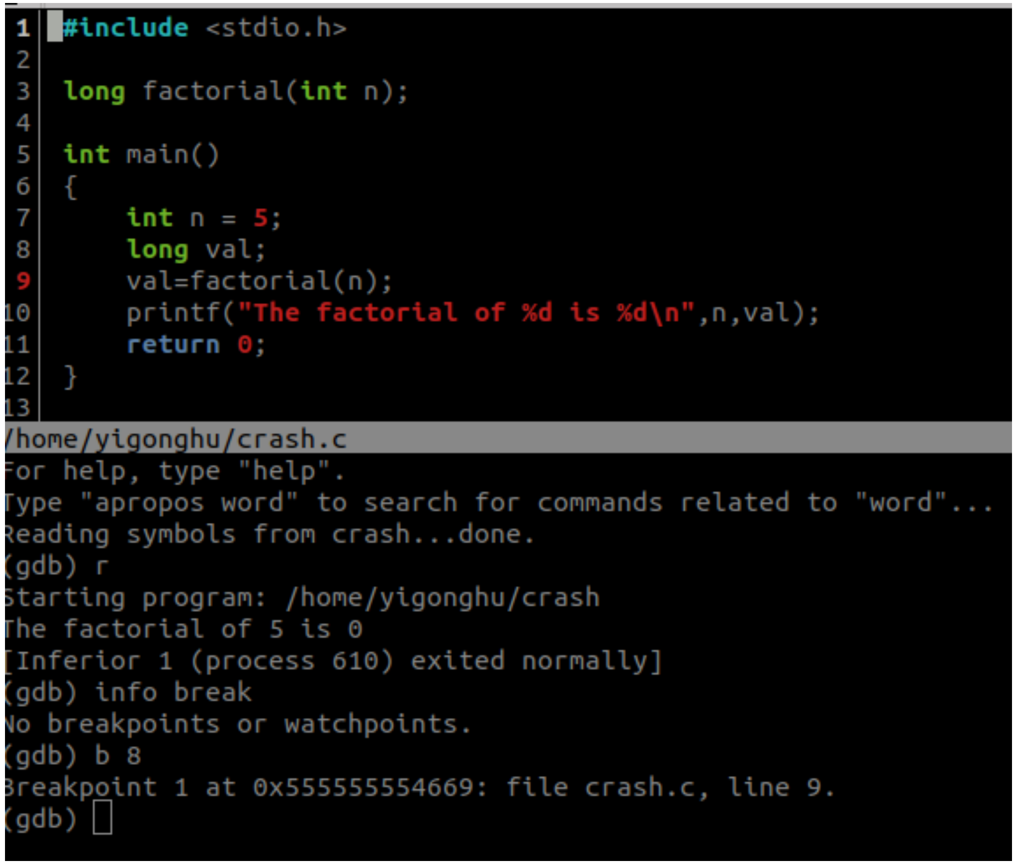
- To run the buggy program, use:

```
(gdb) run
```

- This runs the program
  - ➢ If it has no serious problems, the program should run fine here too.
  - ➢ If the program did have issues, then you should get some useful information like the line number where it crashed, and parameters to the function that caused the error

Setting Breakpoints

- What if the program output doesn’t match your expectation?
  - Step through your code a bit at a time, until you arrive upon the error
- Breakpoints can be used to stop the program run in the middle, at a designated point.

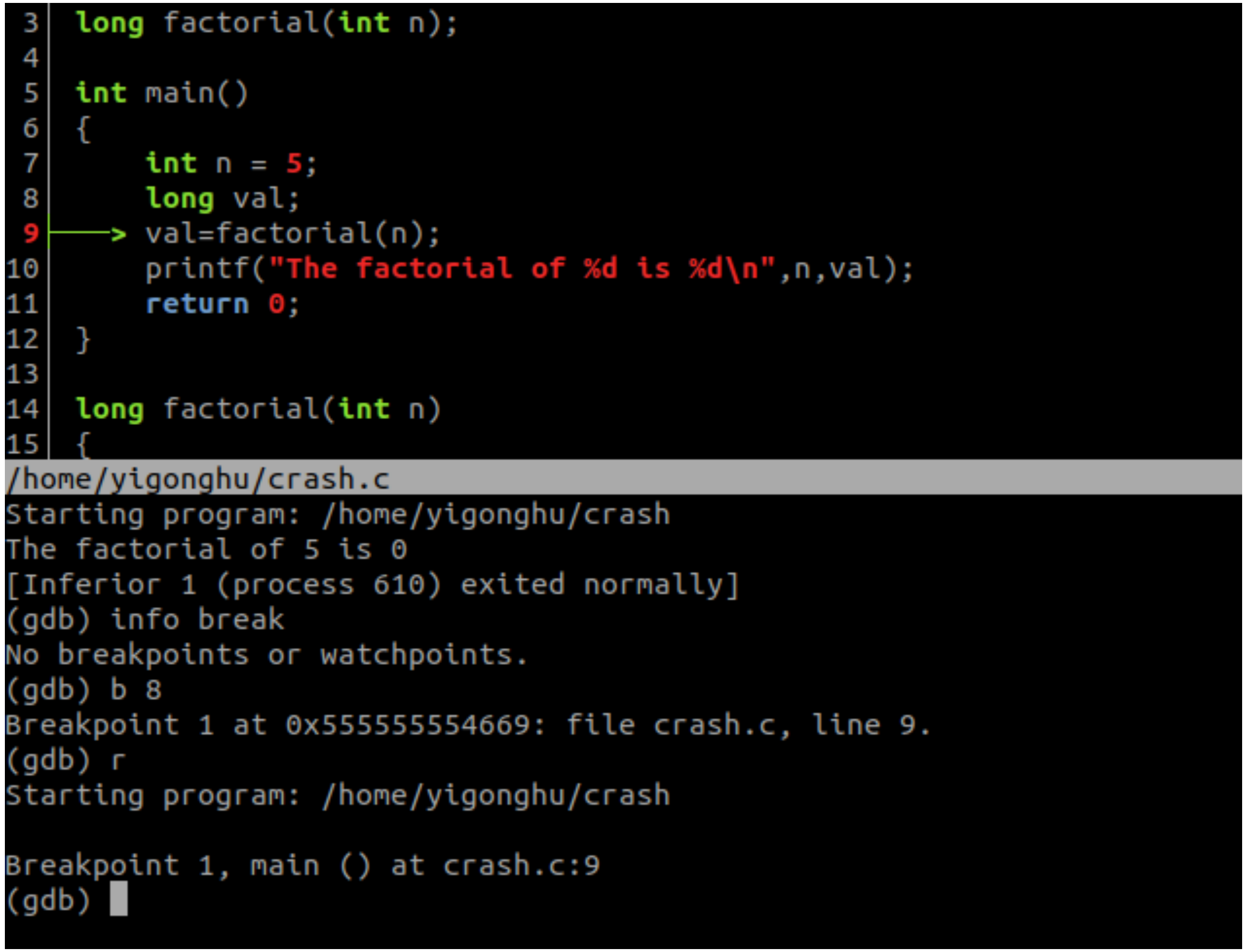


```
 1 #include <stdio.h>
 2
 3 long factorial(int n);
 4
 5 int main()
 6 {
 7     int n = 5;
 8     long val;
 9     val=factorial(n);
10     printf("The factorial of %d is %d\n",n,val);
11     return 0;
12 }
13
```

```
/home/yigonghu/crash.c
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from crash...done.
(gdb) r
Starting program: /home/yigonghu/crash
The factorial of 5 is 0
[Inferior 1 (process 610) exited normally]
(gdb) info break
No breakpoints or watchpoints.
(gdb) b 8
Breakpoint 1 at 0x555555554669: file crash.c, line 9.
(gdb) 
```

Run Program with breakpoints

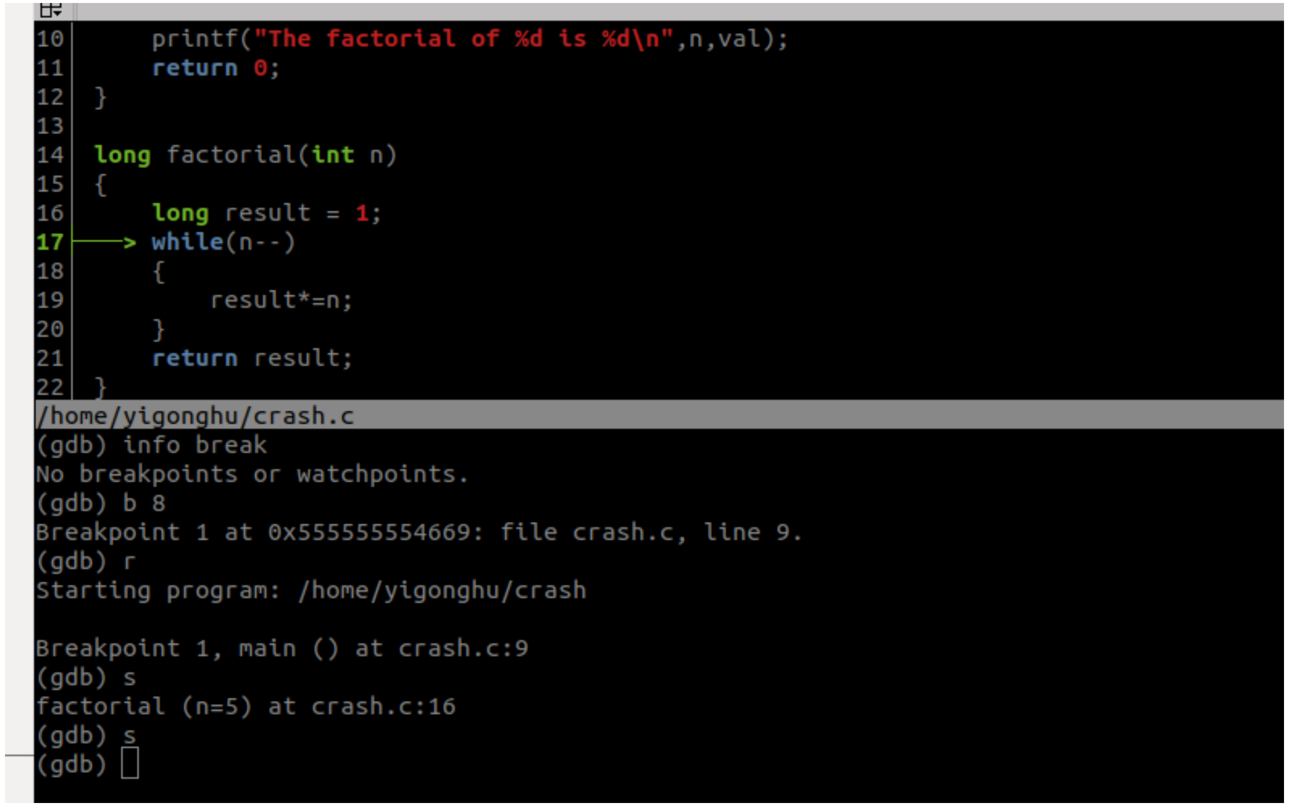
- Once you’ve set a breakpoint, you can try using the run command again, This time, it should stop where you tell it to.



```
 3│ long factorial(int n);
 4│
 5│ int main()
 6│ {
 7│         int n = 5;
 8│         long val;
 9├──────> val=factorial(n);
10│         printf("The factorial of %d is %d\n",n,val);
11│         return 0;
12│ }
13│
14│ long factorial(int n)
15│ {
/home/yigonghu/crash.c
Starting program: /home/yigonghu/crash
The factorial of 5 is 0
[Inferior 1 (process 610) exited normally]
(gdb) info break
No breakpoints or watchpoints.
(gdb) b 8
Breakpoint 1 at 0x555555554669: file crash.c, line 9.
(gdb) r
Starting program: /home/yigonghu/crash

Breakpoint 1, main () at crash.c:9
(gdb) 
```

Step Into Code

- You can single-step (execute just the next line of code) by typing “step.” This gives you really fine-grained control over how the program proceeds. You can do this a lot.

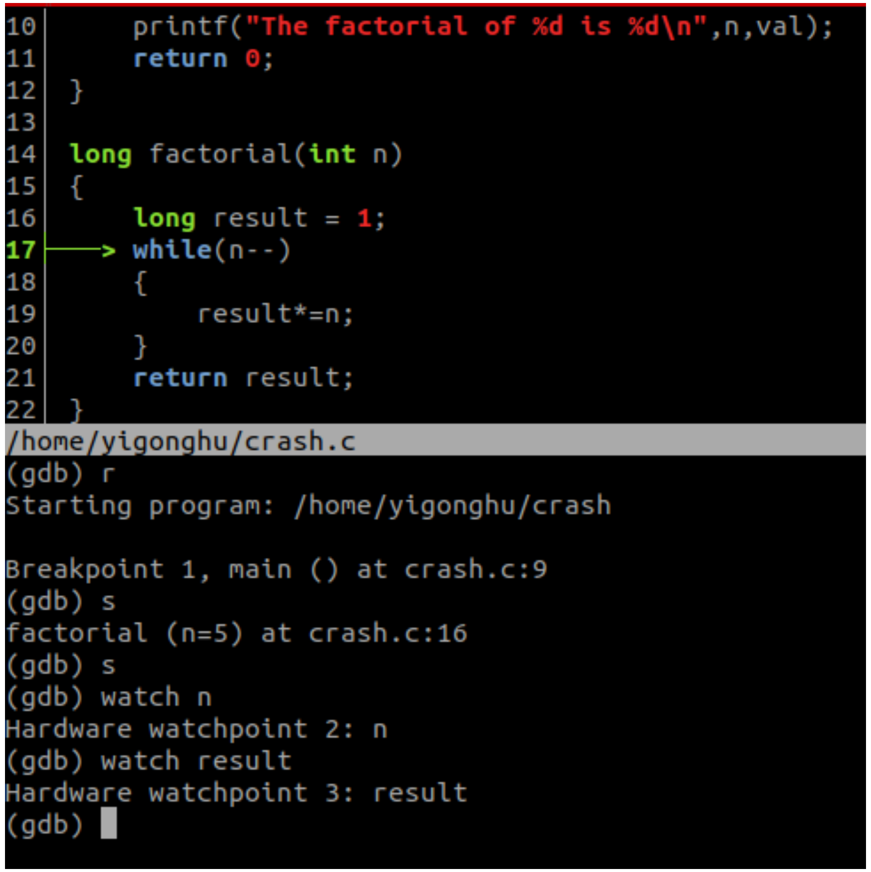


```
        printf("The factorial of %d is %d\n",n,val);
        return 0;
    }

    long factorial(int n)
    {
        long result = 1;
  ───> while(n--)
        {
            result*=n;
        }
        return result;
    }
```

```
/home/yigonghu/crash.c
(gdb) info break
No breakpoints or watchpoints.
(gdb) b 8
Breakpoint 1 at 0x555555554669: file crash.c, line 9.
(gdb) r
Starting program: /home/yigonghu/crash

Breakpoint 1, main () at crash.c:9
(gdb) s
factorial (n=5) at crash.c:16
(gdb) s
(gdb) 
```

How to Check The Variable Change?

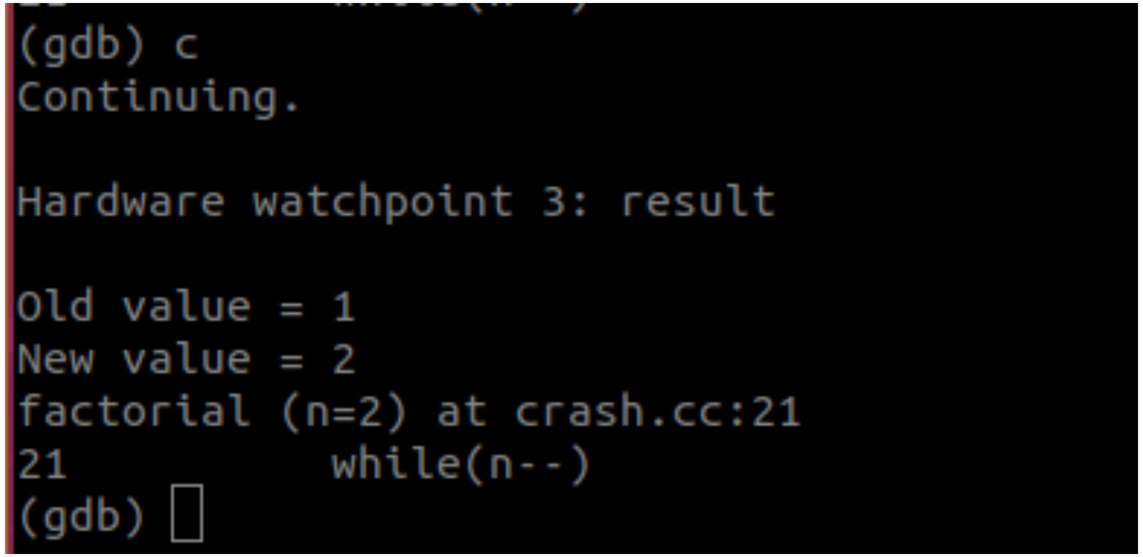
- Setting Watchpoints
  - Watchpoints interrupt the program whenever a watched variable’s value is modified



```
        printf("The factorial of %d is %d\n",n,val);
        return 0;
}

long factorial(int n)
{
        long result = 1;
  ────> while(n--)
        {
            result*=n;
        }
        return result;
}
/home/yigonghu/crash.c
(gdb) r
Starting program: /home/yigonghu/crash

Breakpoint 1, main () at crash.c:9
(gdb) s
factorial (n=5) at crash.c:16
(gdb) s
(gdb) watch n
Hardware watchpoint 2: n
(gdb) watch result
Hardware watchpoint 3: result
(gdb) 
```

Checking the Value of Variable

- After running the program, We've found the first bug! result is supposed to be evaluated by multiplying 3 * 2 * 1 but here the multiplication starts from 2. To correct it, we have to change the loop a little bit, but before that, lets see if the rest of the calculation is correct



```
(gdb) c
Continuing.

Hardware watchpoint 3: result

Old value = 1
New value = 2
factorial (n=2) at crash.cc:21
21          while(n--)
(gdb) 
```

GDB on Pintos (1)

- Pintos uses gdb's remote debugging feature

- Debugging Pintos is a two-step process
  - Starting pintos with gdb option: pintos --gdb -- run mytest
  - A second terminal to invoke GDB on kernel image: pintos-gdb kernel.o
  - Within the GDB shell, issue command: target remote localhost:1234
    - short-hand for the command: debugpintos

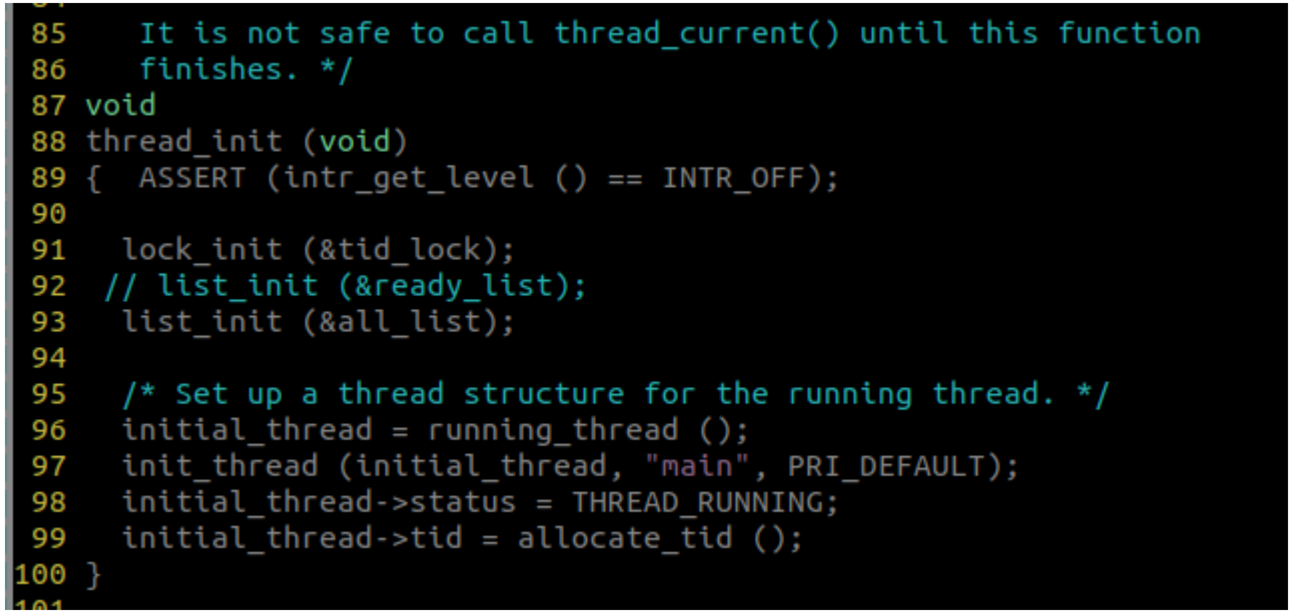
- If using lab machine, you need to use a custom port to avoid port conflict error: --gdb-port = port_number
  - Use the same port number in GDB: target remote localhost:port_number

GDB on Pintos (2)

- CGDB
    - The CS lab machine has cgdb installed
    - pintos-gdb will automatically prefer cgdb if it's available

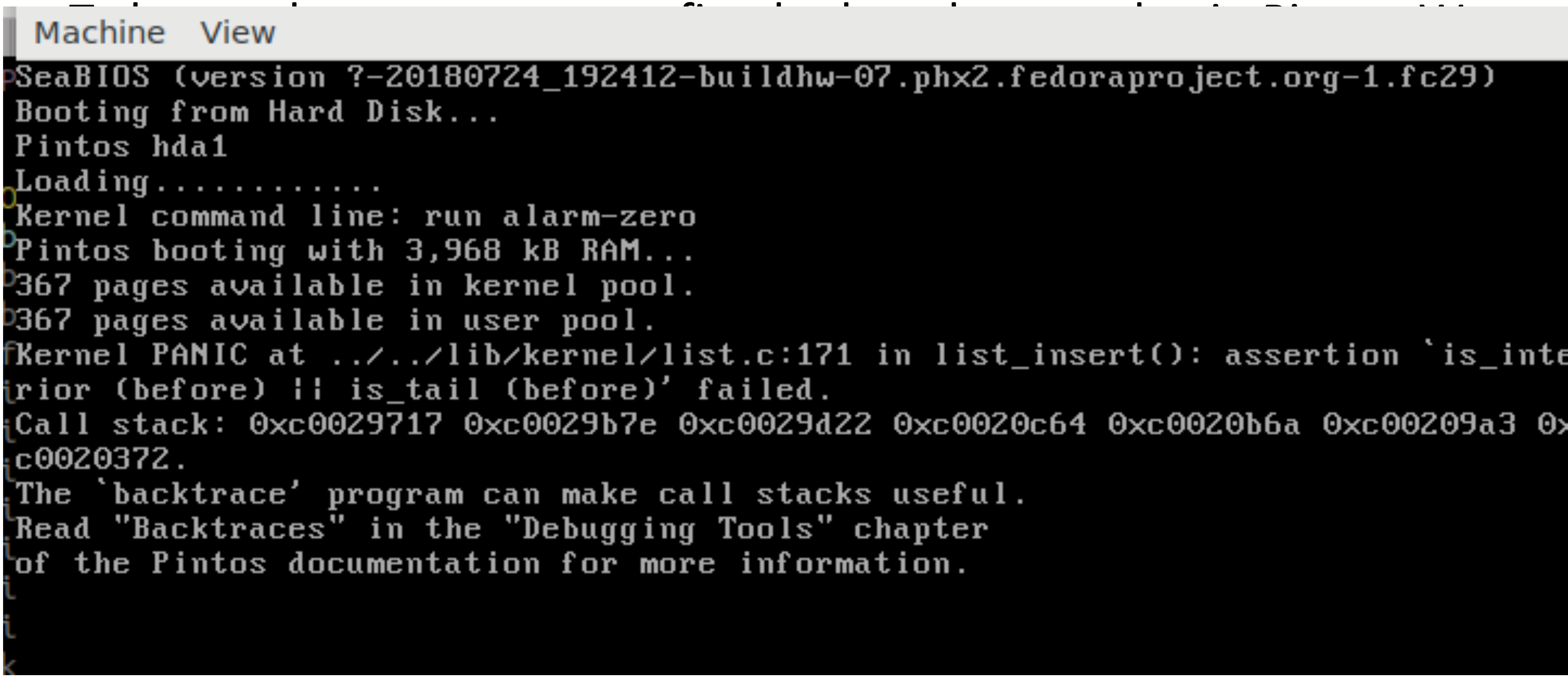
Debugging On Pintos

- We introduce a bug into Pintos by commenting the initialization of ready_list, which will cause an assertion error in Pintos



```
   It is not safe to call thread_current() until this function
   finishes. */
void
thread_init (void)
{  ASSERT (intr_get_level () == INTR_OFF);

  lock_init (&tid_lock);
 // list_init (&ready_list);
  list_init (&all_list);

  /* Set up a thread structure for the running thread. */
  initial_thread = running_thread ();
  init_thread (initial_thread, "main", PRI_DEFAULT);
  initial_thread->status = THREAD_RUNNING;
  initial_thread->tid = allocate_tid ();
}
```

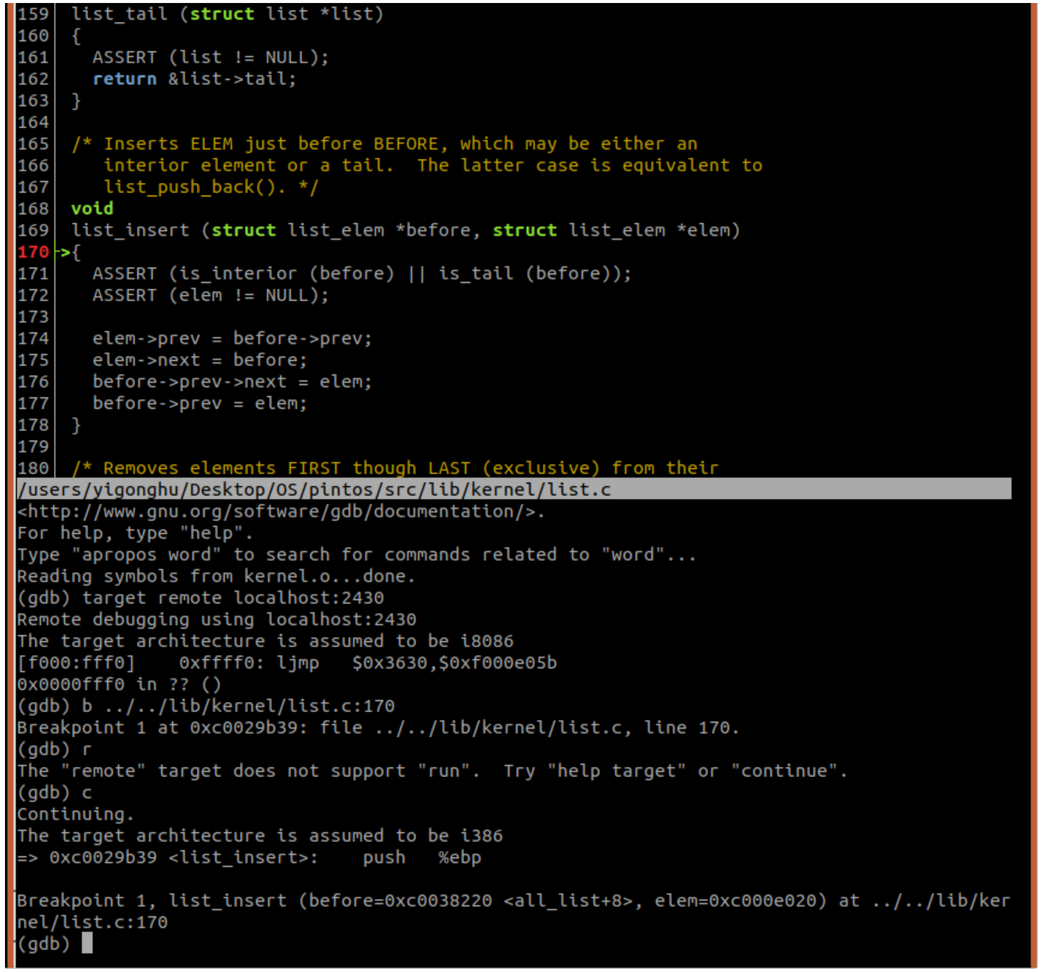
Debugging On Pintos



```
SeaBIOS (version ?-20180724_192412-buildhw-07.phx2.fedoraproject.org-1.fc29)
Booting from Hard Disk...
Pintos hda1
Loading.............
Kernel command line: run alarm-zero
Pintos booting with 3,968 kB RAM...
367 pages available in kernel pool.
367 pages available in user pool.
Kernel PANIC at ../../lib/kernel/list.c:171 in list_insert(): assertion `is_inte
rior (before) || is_tail (before)' failed.
Call stack: 0xc0029717 0xc0029b7e 0xc0029d22 0xc0020c64 0xc0020b6a 0xc00209a3 0x
c0020372.
The `backtrace' program can make call stacks useful.
Read "Backtraces" in the "Debugging Tools" chapter
of the Pintos documentation for more information.
```

Debugging On Pintos

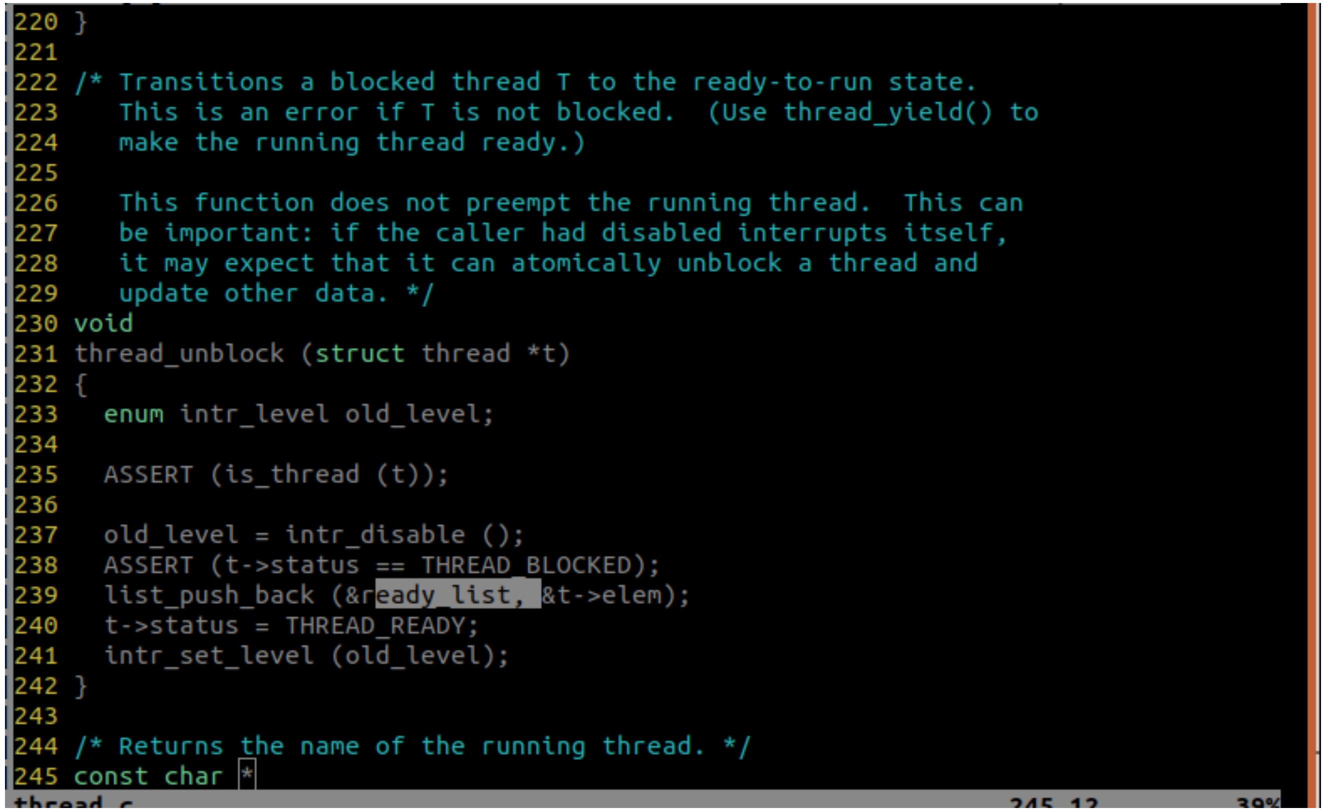
- To trace the call stack that cause the failure, we set a break point in list.c:170 and run the Pintos



```
159  list_tail (struct list *list)
160  {
161    ASSERT (list != NULL);
162    return &list->tail;
163  }
164
165  /* Inserts ELEM just before BEFORE, which may be either an
166     interior element or a tail.  The latter case is equivalent to
167     list_push_back(). */
168  void
169  list_insert (struct list_elem *before, struct list_elem *elem)
170 >{
171    ASSERT (is_interior (before) || is_tail (before));
172    ASSERT (elem != NULL);
173
174    elem->prev = before->prev;
175    elem->next = before;
176    before->prev->next = elem;
177    before->prev = elem;
178  }
179
180  /* Removes elements FIRST though LAST (exclusive) from their
/users/yigonghu/Desktop/OS/pintos/src/lib/kernel/list.c
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from kernel.o...done.
(gdb) target remote localhost:2430
Remote debugging using localhost:2430
The target architecture is assumed to be i8086
[f000:fff0]    0xffff0: ljmp   $0x3630,$0xf000e05b
0x0000fff0 in ?? ()
(gdb) b ../../lib/kernel/list.c:170
Breakpoint 1 at 0xc0029b39: file ../../lib/kernel/list.c, line 170.
(gdb) r
The "remote" target does not support "run".  Try "help target" or "continue".
(gdb) c
Continuing.
The target architecture is assumed to be i386
=> 0xc0029b39 <list_insert>:    push   %ebp

Breakpoint 1, list_insert (before=0xc0038220 <all_list+8>, elem=0xc000e020) at ../../lib/ker
nel/list.c:170
(gdb) 
```

Debugging On Pintos

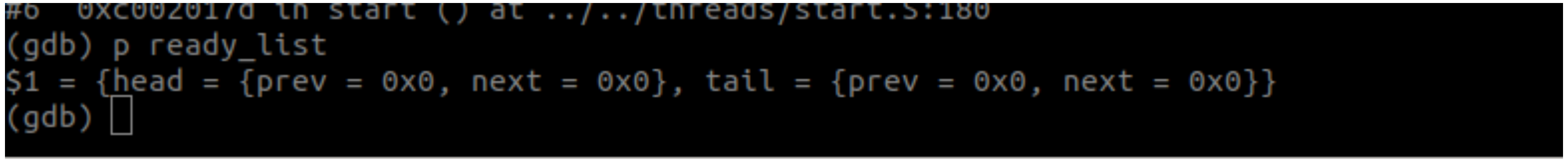
- Then we continue to run the program until it reaches the assertion error. We check the call stack of list_insert using `backtrace` (`bt`). We find that the thread_unblock function calls the insert_error.



```
}

/* Transitions a blocked thread T to the ready-to-run state.
   This is an error if T is not blocked.  (Use thread_yield() to
   make the running thread ready.)

   This function does not preempt the running thread.  This can
   be important: if the caller had disabled interrupts itself,
   it may expect that it can atomically unblock a thread and
   update other data. */
void
thread_unblock (struct thread *t)
{
  enum intr_level old_level;

  ASSERT (is_thread (t));

  old_level = intr_disable ();
  ASSERT (t->status == THREAD_BLOCKED);
  list_push_back (&ready_list, &t->elem);
  t->status = THREAD_READY;
  intr_set_level (old_level);
}

/* Returns the name of the running thread. */
const char *
thread.c                                                  245,12         39%
```

Debugging On Pintos

- We then check the value of ready_list and we find that the list is not initialized yet. Then we know the root cause is the uninitialized data.



```
#6  0xc002017d in start () at ../../threads/start.S:180
(gdb) p ready_list
$1 = {head = {prev = 0x0, next = 0x0}, tail = {prev = 0x0, next = 0x0}}
(gdb) 
```

Git

- Be familiar with daily commands
  - git clone
  - git commit
  - git push
  - …
- A little more advanced
  - git checkout
  - git reset
  - …
- Reference

  https://www.atlassian.com/git/tutorials

  https://www.katacoda.com/courses/git

Tips for using Git

- When using Git, you should:
  - Use .gitignore to avoid checking in unnecessary files (e.g., object files)
  - Write meaningful commit messages
  - Create branches for different exercises/features
  - Use pull requests to merge feature branch into master
    - Have teammates review the changes
  - Integrate your team's changes early and often
- When using Git, you should avoid:
  - Check in object files
  - Use git add -A command to commit all the files
    - Use git status to see modified & staged files
  - Use git push --force to force remote update

IDE settings: vim

- Vim is sufficient for daily development on servers
  - Recommended Vim plugins: NERDTree, YouCompleteMe, COC, fzf.vim
  - Recommended NeoVim plugins: COC.nvim
  - Plugin manager: vim-plug

- cscope is a developer's tool for browsing source code
  - Reference: http://tuxdiary.com/2012/04/03/code-browsing-using-ctags-and-cscope
  - Use cscope in Pintos: https://www.cs.jhu.edu/~huang/cs318/fall21/project/pintos_12.html#SEC170
    - `make cscope` in src/

IDE settings: vim

- "Inconvenient" to use at the beginning
  - Try to force yourself using them
  - Typing keys is in general much faster than mouse-click
  - Once you develop the habit, this set of tools will be your lifetime companions that boost programming productivity

IDE settings: VSCode

- VSCode is a GUI IDE by Microsoft
  - Free of charge
  - Easy to use for remote development via SSH

- Cache locally, sync automatically, execute remotely
  - Install “Remote Development Extension Pack” plugin
  - Connect to a remote host
  - Remote Development Reference: https://code.visualstudio.com/docs/remote/ssh

QEMU/bochs

- QEMU: faster

```
pintos --qemu …
make check SIMULATOR=--qemu
```

- Bochs: support reproducible mode

```
pintos --bochs ...
make check SIMULATOR=--bochs
```

(For reproducible mode, refer to http://bochs.sourceforge.net/doc/docbook/user/bochsrc.html, section 4.3.16)

(Lab 1 will use bochs & Lab 2-4 will use QEMU. Change SIMULATOR for local test-only.)

Outline

- Administrivia
- Lab 0 overview
- Environment Setup
- Dev Tool
- Tips

Development Suggestions

- Bad coding habit
  - Divided the assignment into pieces
  - Each group member worked on his or her piece in isolation until just before the deadline
  - Reconvened to combine their code and submit
- Why is it bad?
  - Conflict with each other
  - Requiring lots of last-minute debugging

Development Suggestions

- Good coding habit
  - Integrating your team's changes early and often
  - Do incremental function testing
  - Using a source code control system such as Git
  - Read the compiler WARNINGs

- This is less likely to produce surprises
- These systems also make it possible to review changes and, when a change introduces a bug, drop back to working versions of code

Code Style

- Can your group member and TAs understand your code easily?
  - E.g.: GNU code style: http://www.gnu.org/prep/standards/
  - Limit source file & function lines
  - Following the naming convention
  - Adding meaningful comments
  - If you remove existing Pintos code, delete it from your source file entirely

- The style will be accounted for during grading
  - Bad code or code with messy styles will get points deducted even if it passes tests.

General Tips

- Think about design before you start coding
  - You can run through your design with the TAs or Professor if you are unsure
- Reserve enough time to write design doc (30% of score)
- Carefully read the project documentation pintos.pdf/.html
  - The appendix of the doc is particularly helpful
  - Many confusions come from not reading the documentation thoroughly

Reference

Project Website

https://yigonghu.github.io/EC440/fall25/projects/

Pintos

https://www.cs.jhu.edu/~huang/cs318/fall21/project/pintos_7.html

CGDB

https://cgdb.github.io/

Git

https://www.atlassian.com/git/tutorials

Recap C/Assemble Language

Reference for C:

https://en.wikibooks.org/wiki/C_Programming/Advanced_data_types
https://en.wikibooks.org/wiki/C_Programming/Pointers_and_arrays

Reference for assembly:

https://en.wikibooks.org/wiki/X86_Assembly/GAS_Syntax
https://wiki.osdev.org/Inline_Assembly

Thanks for attending! Happy hacking!