Pintos introduction
Project 0

“Hello World!” and lists

Operating Systems, Spring 2025

Pintos overview

- Educational Operating System developed at Stanford
- Code is well commented and has extensive documentation
- Written in C (minimal assembly code) to run on regular x86 architecture
- The basic kernel provides minimal features:
  - Thread creation and destruction
  - Thread blocking and unblocking
  - Preemptive round-robin scheduler
- We will run Pintos on Bochs (opens-source x86 emulator)

Pintos folder structure

- **threads/** - threads and scheduling
- **userprog/** - user programs
- **vm/** - virtual memory
- **filesys/** - filesystem
- **devices/** - interface to I/O devices: keyboard, disks, timer, …
- **lib/** - contains a subset of the standard C library
- **utils/** - scripts to run and debug pintos
- **tests/** - unit tests

Adding a test

- Change directory to `/tests/threads` and create a file `test_hello.c`

```
void test_hello ()
{
     printf(“Hello, world!\n”);
}
```

- Add the name of the new test function in `tests.h` and `tests.c`
- Add `test_hello.c` to `Make.tests`
- Go to `threads/`, recompile and run `test_hello`

```
$ cd <pintos-directory>/threads
$ make
$ pintos -v -- run test_hello
```

Debugging a test

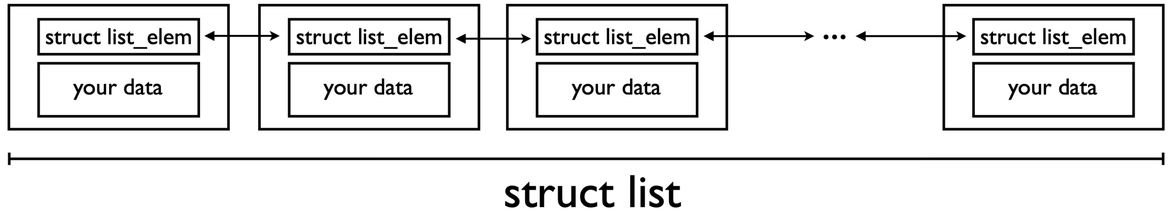
```
### at <pintos-directory>/threads

$ cd pintos/threads
$ pintos --gdb -v -- run test_hello

### another terminal, at <pintos-directory>/threads/build ###

$ cd pintos/threads/build
$ pintos-gdb kernel.o
(gdb) debugpintos
(gdb) break test_hello
(gdb) continue
(gdb) backtrace

### should output something like this ###

#0  test_hello () at ../../tests/threads/test_hello.c:25
#1  0xc0029c6d in run_test (name=name@entry=0xc0007d42 "test-hello")
    at ../../tests/threads/tests.c:57
#2  0xc0020187 in run_task (argv=0xc0034a00 <argv.2029>)
    at ../../threads/init.c:290
#3  0xc00206f7 in run_actions (argv=<optimized out>)
    at ../../threads/init.c:340
#4  main () at ../../threads/init.c:133
```

Lists in Pintos

- `/lib/kernel/` contains data types such as lists, hash tables, bitmaps, …
- We use (double-linked) lists: `#include “kernel/lists.h”`
  - Lists are used all over Pintos code and are fundamental for implementing missing features. You can check the API reference, with documentation, on `lib/kernel/list.h`
- How to use lists?
  - Declare a `struct list` and initialize it with: `list_init(struct list *)`
  - Each element is a struct that must contain a field of type: `struct list_elem`
  - An element is appended to the list with: `list_push_back(struct list *, struct list_elem *)` The allocation of the element is responsibility of the user (no deep copy is performed by this function)!

```
#include "lib/kernel/list.h"

struct item {
   struct list_elem elem;
   int value;
};

void some_function(void) {
   struct list item_list;
   list_init(&item_list);

   struct item newitem;
   newitem.value = 12;

   list_push_back(&item_list,
                  &newitem.elem);

   // ... some more code
}
```

More on lists



- To iterate trough a list (iteration is done on the `struct_elem` field):
  `list_begin(struct list *)`
  `list_end(struct list *)`
  `list_next(struct list_elem *)`
- To access an element of the list:
  `list_entry(struct list_elem *, <user struct's name>, <list_elem field name>)`
- To sort the list elements:
  `list_sort(struct list *, compare_function *, void*)`
- To free an element of the list: retrieve it with `list_entry()`, then call `free()` with the retrieved element as argument

```
#include "lib/kernel/list.h"

struct item {
   struct list_elem elem;
   int value;
};

void visit_all(struct list * lst) {
   struct list_elem * pos;

   for (pos  = list_begin(lst) ;
        pos != list_end(lst)   ;
        pos  = list_next(pos))
   {
      struct item * it;
      it = list_entry(pos,
                      struct item,
                      elem);

      // ... do something with it
      // e.g.: it->value = 20
   }
}
```

Sorting a list

- To sort the list elements:
  `list_sort(struct list *, compare_function *, void*)`
- A compare function is required:
  `func(const struct list_elem * a, const struct list_elem * b, void * aux)`
  - It should return `true` if the data in `a` precedes the data in `b`, `false` otherwise
  - The `aux` buffer can be ignored
  - It can be used in other functions:
    `list_insert_ordered()`, `list unique()`, `list_max()`, `list_min()`, …

```
#include "lib/kernel/list.h"

struct item {
   struct list_elem elem;
   int value;
};

bool compare_items
      (const struct list_elem * a,
       const struct list_elem * b,
       void * aux)
{
   struct item * ia =
      list_entry(a, struct item, elem);
   struct item * ib =
      list_entry(b, struct item, elem);
   return (ia->value < ib->value);
}

int main() {
   // create and populate item_list
   list_sort(&item_list,
             compare_items,
             NULL);
}
```

Project 0: List sorting test

- Create a list of elements that have a field called `priority` (type `int`)
- Create a function
  `void populate(struct list * l, int * a, int n)`
  that fills the list `l` with the elements of array `a`, which contains `n` integers
- Download and include `listpop.h` to populate your list using the array `ITEMARRAY`, which has `ITEMCOUNT` elements
- Create a function
  `void print_sorted(struct list * l)`
  that sorts the elements of your list `l` and prints them in ascending order of priority
- Turn your code into a test to be run on top of Pintos, changing the necessary files (check the `HelloWorld` example)