C From Zero to Hero

Dr. Naser Al Madi

Learning objectives

- Project 3
- Learn C
- Include guards
- Make files
- fork + wait + exit
- Go over Xv6 and qemu

Remove min_vruntime

Remove min_vruntime

- The remove method for a RB-Tree is complicated.
- For our application (CFS), we can implement a simple remove method that works in constant time.

min_vruntime

- We can store a reference (pointer) to the node with the min_vruntime in the tree.
- Min_vruntime is always the left most node.
- Everytime we insert a node into the tree, we check if the value of the new node is less than min_vruntime. If the value is smaller, we update min_vruntime.
- This update is done in constant time, we didn't have to search for the minimum value or traverse the tree to the leftmost node.

Remove min_vruntime

Since we always remove the leftmost node, we need to account for two simple cases only:

1. Leftmost has no children: min_vruntime becomes its parent
2. Leftmost has one right child: min_vruntime becomes the right child

Red-Black Tree Visualization

https://www.cs.usfca.edu/~galles/visualization/RedBlack.html

Project 3

C From Zero to Hero

Why learn C?

- Middle-level language created in 1972
- One of the Top 10 most used languages
- Mother of all languages (almost all programming languages are influenced by C)
- Used in embedded systems, OSs, automation, self-driving cars, … etc
- It was developed to write the UNIX operating system.
- Very similar to Java, if you learn it you practically learned C++ too.
- Very very fast!

Keep in mind

- C is not object-oriented, so no classes
- C++ is an extension of C that is object oriented
- No garbage collection like Java, so memory allocation/deallocation is done manually
- gcc is the compiler of choice for C (clang also works)
- Next project is going to be in C, so the sooner you get your computer ready the better!

C Basics

Basic C code structure

```
#include <stdio.h>


int main() {
  printf("Hello World!");
  return 0;
}
```

Basic C code structure

```c
#include <stdio.h>


int main() {

  printf("Hello World!");

  return 0;

}
```

Standard IO, to print to terminal and enter text from keyboard.

Basic C code structure

```c
#include <stdio.h>


int main() {

  printf("Hello World!");

  return 0;

}
```

Return type: int
Function name: main
Parameters: ()

Basic C code structure

```
#include <stdio.h>

int main() {
  printf("Hello World!");
  return 0;
}
```

sends formatted output to stdout

Basic C code structure

```c
#include <stdio.h>

int main() {
  printf("Hello World!");
  return 0;
}
```

return 0 means the program ended normally

return 1 means something went wrong

What we need to write and run C code?

- An IDE of your choosing
  - I recommend VS Code, I will use a text editor called Sublime
- C compiler
  - On Mac and Linux we will use gcc
  - On Windows we will use MinGW or Just download Ubuntu on Windows (I recommend Ubuntu)

Compile and run C code

test.c

```c
#include <stdio.h>


int main() {

  printf("Hello World!");

  return 0;

}
```

Compile on terminal

```
>gcc test.c
```

Compile and run C code

test.c

```c
#include <stdio.h>


int main() {

  printf("Hello World!");

  return 0;

}
```

run on terminal

```
>./a.out
Hello World!
```

Compile and run C code

test.c

```c
#include <stdio.h>


int main() {

  printf("Hello World!");

  return 0;

}
```

Compile on terminal

```
>gcc test.c  -o test
```

Compile and run C code

test.c

```c
#include <stdio.h>


int main() {

  printf("Hello World!");

  return 0;

}
```

run on terminal

```
>./test
Hello World!
```

Installing C environment

Install instructions

Mac	Windows 10
● Install VS Code ● Install the following extensions to VS Code: ○ C/C++ by Microsoft ○ Code Runner ● In a terminal, check if you have a c compiler: ○ clang –version ○ If not installed, type: xcode-select --install Check this page for troubleshooting.	● Install VS Code ● Install the following extensions to VS Code: ○ C/C++ by Microsoft ○ Code Runner ● Install Windows Subsystem for Linux (WSL) ○ Open PowerShell ○ wsl --install -d ubuntu ● Run Ubuntu ● Configure Ubuntu with username ● sudo apt-get update && sudo apt-get install git nasm build-essential qemu gdb Check this page for troubleshooting

More C Basics

Comments

```
printf("Hello World!"); // This is a comment


/* The code below will print the words Hello World!

to the screen, and it is amazing */

printf("Hello World!");
```

Variables

```
// Create variables
int myNum = 5;              // Integer (whole number)
float myFloatNum = 5.99;    // Floating point number
char myLetter = 'N';        // Character

// Print variables
printf("%d\n", myNum);
printf("%f\n", myFloatNum);
printf("%c\n", myLetter);
```

Formatting output

```c
#include <stdio.h>

int main () {
  int ch;

  for( ch = 75 ; ch <= 100; ch++ ) {
    printf("ASCII value = %d, Character = %c\n", ch , ch );
  }

  return(0);
}
```

```
ASCII value = 75, Character = K
ASCII value = 76, Character = L
ASCII value = 77, Character = M
ASCII value = 78, Character = N
ASCII value = 79, Character = O
ASCII value = 80, Character = P
ASCII value = 81, Character = Q
...
```

More info on printf formatting: https://www.tutorialspoint.com/c_standard_library/c_function_printf.htm

Constants

```
const int MYNUM = 15;  // myNum will always be 15

MYNUM = 10;  // error: assignment of read-only variable 'myNum'
```

Logical Operators

&& Logical and

|| Logical or

! Logical not

Sizeof in bytes

```
int myInt;
float myFloat;
double myDouble;
char myChar;

printf("%lu\n", sizeof(myInt));
printf("%lu\n", sizeof(myFloat));
printf("%lu\n", sizeof(myDouble));
printf("%lu\n", sizeof(myChar));
```

If-statements

```
int time = 22;
if (time < 10) {
  printf("Good morning.");
} else if (time < 20) {
  printf("Good day.");
} else {
  printf("Good evening.");
}
// Outputs "Good evening."
```

Switch statements

```
int day = 4;

switch (day) {
  case 6:
            printf("Today is Saturday");
    break;
  case 7:
            printf("Today is Sunday");
    break;
  default:
            printf("Looking forward to the Weekend");
}

// Outputs "Looking forward to the Weekend"
```

While loops

```
int i = 0;


while (i < 5) {
  printf("%d\n", i);
  i++;
}
```

For loops

```
int i;

for (i = 0; i < 5; i++) {
  printf("%d\n", i);
}
```

Break and Continue

```c
int i;

for (i = 0; i < 10; i++) {
  if (i == 4) {
    break;
  }
  printf("%d\n", i);
}
```

```c
int i;

for (i = 0; i < 10; i++) {
  if (i == 4) {
    continue;
  }
  printf("%d\n", i);
}
```

What will be printed?

Break and Continue

```c
int i;

for (i = 0; i < 10; i++) {
  if (i == 4) {
    break;
  }
  printf("%d\n", i);
}
```

```
0
1
2
3
```

What will be printed?

```c
int i;

for (i = 0; i < 10; i++) {
  if (i == 4) {
    continue;
  }
  printf("%d\n", i);
}
```

```
0
1
2
3
5
6
7
8
9
```

Arrays

```
int myNumbers[] = {25, 50, 75, 100};
```

Arrays (no size function)

```
int myNumbers[] = {25, 50, 75, 100};

int i;


for (i = 0; i < 4; i++) {

  printf("%d\n", myNumbers[i]);

}
```

Strings

```
char greetings[] = "Hello World!";
```

Strings

```
char greetings[] = "Hello World!";

printf("%s", greetings);
```

Strings are chars

```
char greetings[] = "Hello World!";

printf("%c", greetings[0]);
```

Strings end with '\0'

```
char greetings[] = {'H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd', '!', '\0'};

char greetings2[] = "Hello World!";


printf("%lu\n", sizeof(greetings));   // Outputs 13

printf("%lu\n", sizeof(greetings2));  // Outputs 13
```

User input

```
// Create an integer variable that will store the number we get from the user
int myNum;

// Ask the user to type a number
printf("Type a number: \n");

// Get and save the number the user types
scanf("%d", &myNum);

// Output the number the user typed
printf("Your number is: %d", myNum);
```

User input

```
// Create an integer variable that will store the number we get from the user
int myNum;

// Ask the user to type a number
printf("Type a number: \n");

// Get and save the number the user types
scanf("%d", &myNum);

// Output the number the user typed
printf("Your number is: %d", myNum);
```

The scanf() function takes two arguments: the format specifier of the variable (%d in the example above) and the reference operator (&myNum), which stores the memory address of the variable.

User input

```
// Create a string
char firstName[30];

// Ask the user to input some text
printf("Enter your first name: \n");

// Get and save the text
scanf("%s", firstName);

// Output the text
printf("Hello %s.", firstName);
```

you don't have to specify the reference operator (&) when working with strings

Memory and pointers

Memory Address

```
int myNum = 10;

printf("%p", &myNum); // Outputs 0x7ffe5367e044
```

Pointers

```
int myNum = 10;      // An int variable
int* ptr = &myNum ;  // A pointer variable, with the name ptr, that stores the address
of myNum

// Output the value of myNum (10)
printf("%d\n", myNum);

// Output the memory address of myNum (0x7ffe5367e044)
printf("%p\n", &myNum);

// Output the memory address of myNum with the pointer (0x7ffe5367e044)
printf("%p\n", ptr);
```

Dereference

```
int myNum = 10;      // Variable declaration
int* ptr = &myNum;   // Pointer declaration

// Reference: Output the memory address ofmyNum with the pointer (0x7ffe5367e044)
printf("%p\n", ptr);

// Dereference: Output the value ofmyNum with the pointer (10)
printf("%d\n", *ptr);
```

Pointer Definition

In C++ a pointer is a variable that stores the address of another variable

Declaring Pointers

```
int* p ;
int *p ;
```

Operators (not declaration)

* → content

& → address

Looking at Memory

Int x = 5;

Address	Name	Value
0000		
0008		
0016		
0024	x	5
0032		

Looking at Memory

```
int x = 5;
int* p;
```

Address	Name	Value
0000	p	
0008		
0016		
0024	x	5
0032		

Looking at Memory

```
int x = 5;
int* p;
p = &x;
```

Address	Name	Value
0000	p	0024
0008		
0016		
0024	x	5
0032		

Looking at Memory

```
int x = 5;
int* p;
p = &x;
```

Address	Name	Value
0000	p	0024
0008		
0016		
0024	x	5
0032		

Looking at Memory

```
int x = 5;
int* p;
p = &x;

*p = 10;
```

Address	Name	Value
0000	p	0024
0008		
0016		
0024	x	~~5~~ 10
0032		

Looking at Memory

```
int x = 5;
int* p;
p = &x;

*p = 10;
int y = 12;
```

Address	Name	Value
0000	p	0024
0008		
0016		
0024	x	10
0032		
0040	y	12
0048		

Looking at Memory

```
int x = 5;
int* p;
p = &x;

*p = 10;
int y = 12;
p = &y;
```

Address	Name	Value
0000	p	0040
0008		
0016		
0024	x	10
0032		
0040	y	12
0048		

Looking at Memory

```
int x = 5;
int* p;
p = &x;

*p = 10;
int y = 12;
p = &y;
*p = 50;
```

Address	Name	Value
0000	p	0040
0008		
0016		
0024	x	10
0032		
0040	y	50
0048		

Summary: create a pointer variable

A pointer that points to an int:

```
int *p;
```

A pointer that points to a float:

```
float *S;
```

Summary: dereference a pointer

dereference the pointer: access the data/value in the memory that the pointer points to.

```
*p = 10;
```

or

```
printf(*p);
```

Quiz

```
printf(x);
printf(*p);
printf(p);
printf(&x);
```

Address	Name	Value
0000	p	0040
0008		
0016		
0024	x	10
0032		
0040	y	50
0048		

May the odds be ever in your favor 

Quiz

```
printf(x);
printf(*p);
printf(p);
printf(&x);
```

Address	Name	Value
0000	p	0040
0008		
0016		
0024	x	10
0032		
0040	y	50
0048		

Quiz

```
printf(x);
printf(*p);
printf(p);
printf(&x);
```

Address	Name	Value
0000	p	0040
0008		
0016		
0024	x	10
0032		
0040	y	50
0048		

Quiz

```
printf(x);
printf(*p);
printf(p);
printf(&x);
```

Address	Name	Value
0000	p	0040
0008		
0016		
0024	x	10
0032		
0040	y	50
0048		

Quiz

```
printf(x);
printf(*p);
printf(p);
printf(&x);
```

Address	Name	Value
0000	p	0040
0008		
0016		
0024	x	10
0032		
0040	y	50
0048		

Questions?

Functions

Functions should be defined before they are called

```
// Create a function
void myFunction() {
  printf("I just got executed!");
}

int main() {
  myFunction(); // call the function
  return 0;
}
```

Function declaration

```
// Function declaration
void myFunction();

// The main method
int main() {
  myFunction();  // call the function
  return 0;
}

// Function definition
void myFunction() {
  printf("I just got executed!");
}
```

Struct is similar to class without methods

```c
struct Test {

    int a;

    int b;

};
```

```c
int main() {

  struct Test test;
  test.a = 1;
  test.b = 2;

  printf("test a is: %d ", test.a);

  return 0;
}
```

Compiling multiple files

Multiple files

func1.h

```c
#include <stdio.h>

void myFunction2() {
  printf("I just got executed!");
}
```

main.c

```c
#include “func1.h”

int main() {

  myFunction2();

  return 0;
}
```

Multiple files

func1.h

```c
#include <stdio.h>

void myFunction2() {
  printf("I just got executed!");
}
```

main.c

```c
#include “func1.h”

int main() {

  myFunction2();

  return 0;
}
```

```
>gcc main.c
```

More interesting multiple files

func1.h

```c
void myFunction2();
```

func1.c

```c
#include <stdio.h>

void myFunction2() {
  printf("I just got executed!");
}
```

main.c

```c
#include “func1.h”

int main() {

  myFunction2();

  return 0;
}
```

```
>gcc main.c func1.c
```

Object files

func1.h

```c
void myFunction2();
```

func1.c

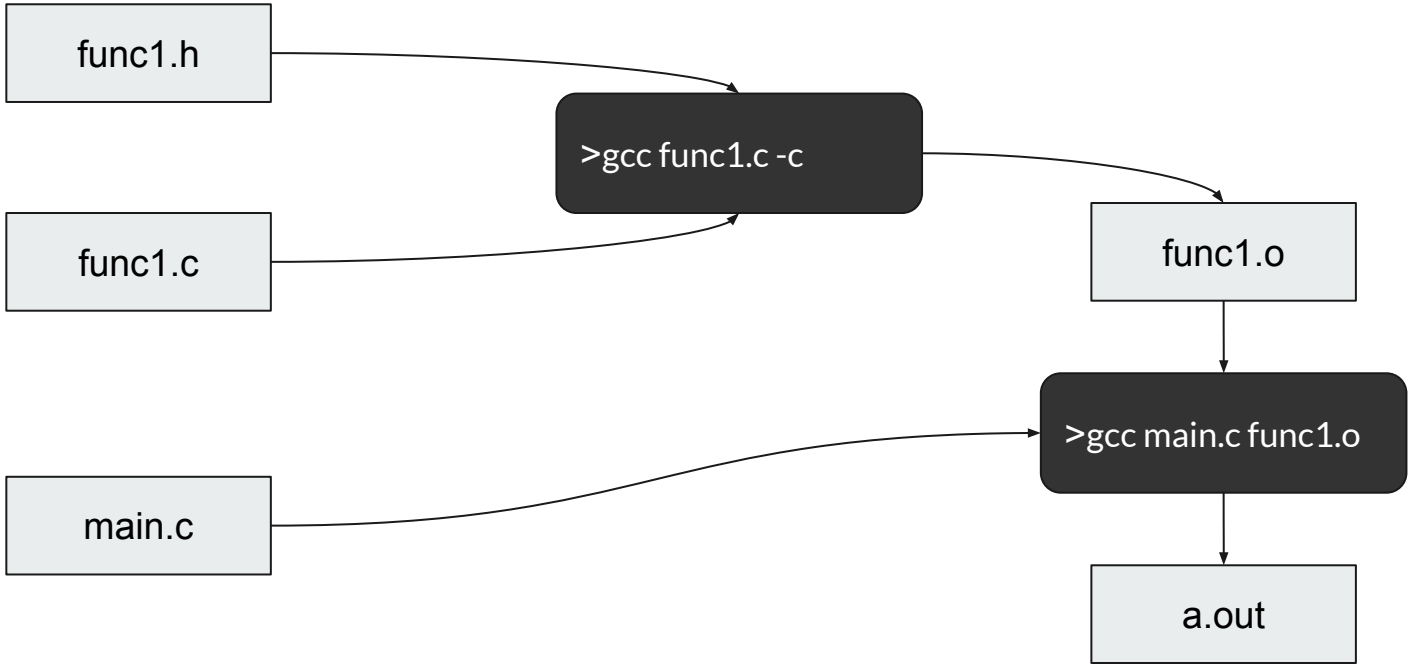
```c
#include <stdio.h>

void myFunction2() {
  printf("I just got executed!");
}
```

main.c

```c
#include “func1.h”

int main() {

  myFunction2();

  return 0;
}
```

```
>gcc func1.c -c

>ls

>func1.h  func1.c  main.c  func1.o

>gcc main.c func1.o

>./a.out

>I just got executed!
```

Object files



funcs.h

```c
void start();
void myFunction2();
```

func1.c

```c
#include <stdio.h>

void myFunction2() {
  printf("I just got executed!");
}
```

func2.c

```c
#include <stdio.h>

void start() {
  printf("Starting!");
}
```

main.c

```c
#include “funcs.h”

int main() {
  start();
  myFunction2();

  return 0;
}
```

```
># how to compile this?
```

funcs.h

```c
void start();
void myFunction2();
```

func1.c

```c
#include <stdio.h>

void myFunction2() {
  printf("I just got executed!");
}
```

func2.c

```c
#include <stdio.h>

void start() {
  printf("Starting!");
}
```

main.c

```c
#include “funcs.h”

int main() {
  start();
  myFunction2();

  return 0;
}
```

```
>gcc main.c func1.c func2.c
```

funcs.h

```c
void start();
void myFunction2();
```

func1.c

```c
#include <stdio.h>

void myFunction2() {
  printf("I just got executed!");
}
```

func2.c

```c
#include <stdio.h>

void start() {
  printf("Starting!");
}
```

main.c

```c
#include “funcs.h”

int main() {
  start();
  myFunction2();

  return 0;
}
```

```
>gcc main.c func1.c func2.c
```

OR

funcs.h

```c
void start();
void myFunction2();
```

func1.c

```c
#include <stdio.h>

void myFunction2() {
  printf("I just got executed!");
}
```

func2.c

```c
#include <stdio.h>

void start() {
  printf("Starting!");
}
```

main.c

```c
#include “funcs.h”

int main() {
  start();
  myFunction2();

  return 0;
}
```

```
>gcc -c func1.c

>gcc -c func2.c

>gcc main.c func1.o func2.o
```

funcs.h

```c
void start();
void myFunction2();
```

func1.c

```c
#include <stdio.h>

void myFunction2() {
  printf("I just got executed!");
}
```

func2.c

```c
#include <stdio.h>

void start() {
  printf("Starting!");
}
```

main.c

```c
#include “funcs.h”

int main() {
  start();
  myFunction2();

  return 0;
}
```

```
>gcc -c func1.c

>gcc -c func2.c

>gcc main.c func1.o func2.o
```

References

- W3schools
-