CMPSC 473
Operating Systems
Design & Construction

Instructor: Ruslan Nikolaev

The Pennsylvania State University

August 22, 2023 – Lecture 1

Instructor

- Ruslan Nikolaev
  - rnikola@psu.edu
  - Office hours: 12:30PM – 2:30PM (Tuesday, except today!)
  - Lectures (Tuesday, Thursday)
    - Section 1: 9:05AM – 10:20AM
    - Section 2: 10:35 AM – 11:50AM

Welcome!

Teaching Assistants

- Rohan Puri, rpp5475@psu.edu
- Mustafa Goktan Gudukbay, mfg5472@psu.edu
- Yunjin Wang, yjw5279@psu.edu
- Md Amit Hasan Arovi, mqa6105@psu.edu

- TA **office hours** will be posted shortly!

Requirements

Prerequisites

- CMPSC311
- CMPEN331

Prior knowledge

- C programming
- Linux command line
- GDB debugger

Textbooks

- OSTEP: Operating Systems: Three Easy Pieces (required, online)
- CSAPP: Computer Systems: A Programmer's Perspective (recommended)
- The Little Book of Semaphores (recommended, online)

Grading (tentative)

This is a project-based course

- 50% Projects, 10% Homeworks
- 15% Midterm: in-class
- 25% Comprehensive final
- Late policy: at most 1 day/assignment at -15%

Note: We **may** also add in-class quizzes

(reallocate up to 5% from the above items)

What you can expect from me

1. When contacted, I will try to respond to all of your inquires in a timely manner. Due to a high volume of requests, it is typically recommended to attend office hours (with TAs or the instructor) or post corresponding questions in the course discussion page. However, please do not hesitate to follow up with the instructor when the matter is urgent and requires a faster response.
2. I want you to learn and succeed so do not be afraid to ask for clarification if you have questions about your assignments, grades, and any related feedback.
3. Your opinions, thoughts and words matter to me and will make a difference in this class. It is my job as your instructor to ensure that you feel safe and supported as you stretch your mind around new concepts in a new learning environment.
4. Your success is important to me. I understand that there are many challenges for students in today's world and we will be here to support you when questions arise and you need guidance. Please do not hesitate to contact the instructor when you have a concern and want assistance in this course.

What I expect from you

1. A desire to learn and a commitment to obtain skills that you can use in other courses as well as out in the real world.
2. Be respectful. Technology affects so much in our daily modern life and so it's important that everyone feels secure and comfortable to explore it, talk about it, and ask questions whenever we are learning something new in the class.
3. Complete all coursework (projects, homeworks, exams, etc.) to the best of your ability and please communicate whenever you have a concern that might prevent you from meeting a particular learning objective or due date.
4. Computer Science is both fun and creative, yet also analytical and scientific at the very same time. Be ready to embrace new concepts and recognize that mistakes are a vital part of the learning process, so it is crucial to allocate enough time to cover the course material.

Logistics

Slides and Projects

- Will be posted on the course's Canvas page

Questions

- Ask during/after the lecture
- During office hours
- Try to get help from TA(s) first, then contact me
- Schedule a separate meeting

Logistics

Projects

- 3-4 projects
- We will use Github for submissions
- To be completed individually

Homeworks

- 1-2 homeworks
- To be completed individually

Topics

- Introduction, C programming
- GDB, GIT, Linux basics, Makefiles (recap/tutorials)
- CPU scheduling, threads, and processes
- CPU modes, interrupts, traps, context switches
- Kernel threads, user threads
- Virtual memory, paging, page tables, TLB, page replacement algorithms

Topics

- Dynamic memory allocation
- Concurrency, mutual exclusion, locks (Peterson's and Bakery algorithms)
- Race conditions, deadlocks
- Semaphores, condition variables
- I/O devices, DMA
- Storage, HDD, SSD
- Filesystems, file descriptors, i-nodes, free space management

Academic integrity

All work is to be done **individually** and **independently**.

No outside help of any form is permitted.

For example, the following are strictly forbidden:

- Looking at another student's code
- Debugging another student's code
- Looking online for hints on how to solve an assignment
- Getting hints from another student on how to solve an assignment
- Providing hints to another student on how to solve an assignment
- Using any code from any online source
- Posting any course-related code to any online public location (e.g., public github repository, online forums, etc.)
- Assisting anyone in violating any academic integrity policy either intentionally or unintentionally from not securing your work
- Getting any help from any source other than a course staff member

Why operating systems?

Decade-long experience with teaching computer systems [see the CSAPP textbook] has shown the importance of discussing the stuff under the hood. Thus, we need to know operating systems to become better programmers and design fast, secure, and reliable software!

Returning data via pointer params

- Technique for returning (multiple) data items

Example:

```
void sum_diff(int a, int b, int* sum, int* diff);
```

Usage:

```
int s;
int d;
sum_diff(7, 3, &s, &d);
```

Implementation:

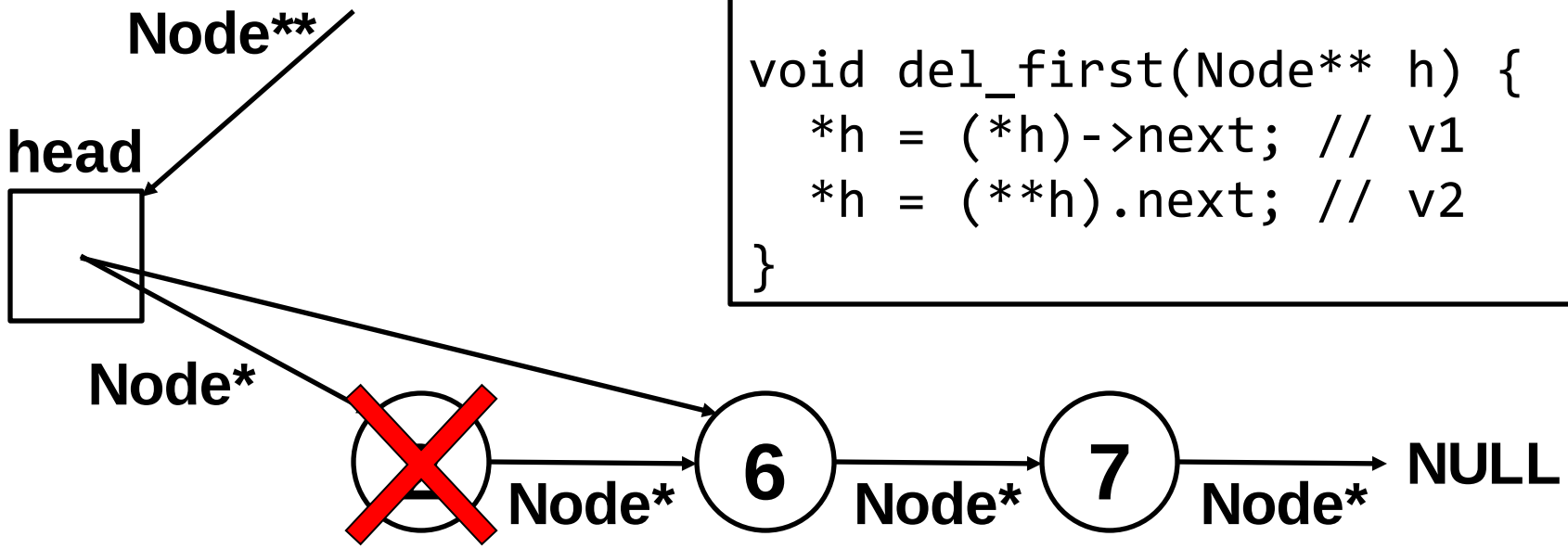
```
void sum_diff(int a, int b, int* sum, int* diff) {
    *sum = a + b;
    *diff = a - b;
}
```

Double pointers

- Pointers can point to pointers (e.g., int**)
- Example: linked-list

```
Node* head = init_list();
del_first(head);
del_first(&head);

void del_first(Node** h) {
  *h = (*h)->next; // v1
  *h = (**h).next; // v2
}
```

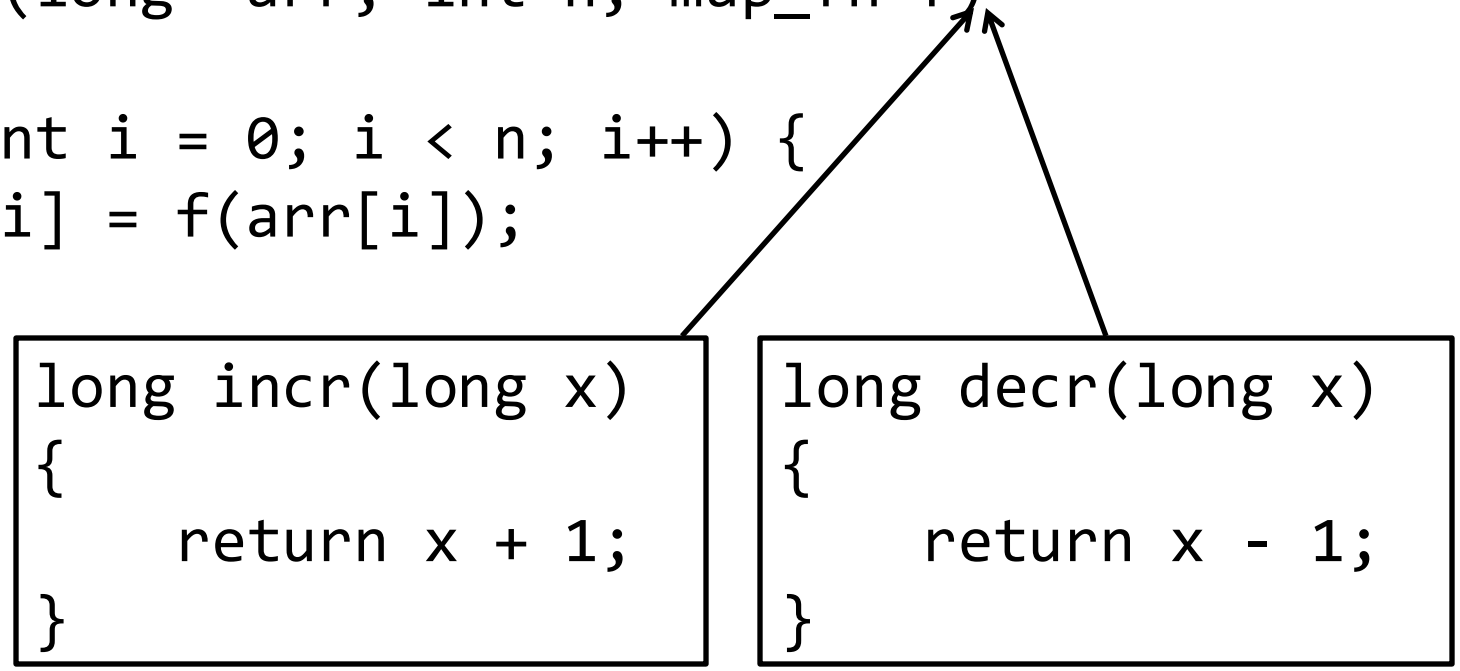


Function pointers

- Deciding which function to call at runtime
- Used for writing more generic code

Example:

```
typedef long (*map_fn)(long);
void map(long* arr, int n, map_fn f)
{
  for (int i = 0; i < n; i++) {
    arr[i] = f(arr[i]);
  }
}
```



```
long incr(long x)
{
    return x + 1;
}
```

```
long decr(long x)
{
    return x - 1;
}
```

Pointer arithmetic

Increments according to size of data pointed to:

Pointer type	Increment size
char*, void*	1 byte
short*	2 bytes
int*	4 bytes
float*	4 bytes
long*	8 bytes
double*	8 bytes
any_type**	8 bytes

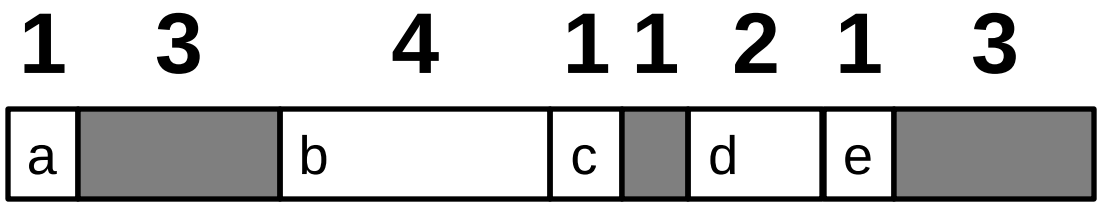
Example: int* p = 0x1000; p + 1 = 0x1004

Relation to arrays: p + n == &p[n]

Useful trick: *(p – 1) == p[-1]

Structure alignment

```
struct {
   char a;
   int b;
   char c;
   short d;
   char e;
};
```



- Ordered according to declaration
- Elements ***typically*** aligned to sizeof(element) or 8 bytes (64-bit CPUs), whatever is smaller

typedef

- Create your own type

Example:

```
typedef int ID;
typedef struct pt { int x; int y; } Point;
typedef int (*compare_fn)(int x, int y);
```

Benefits:

- Know what data you're dealing with: ID vs int
- Easily change types: change ID int → long
- Avoid typing struct: struct pt vs Point
- Avoid messy casts: (int (*)(int, int)) vs (compare_fn)

Header files

- Header files (.h) typically contain definitions (e.g., structs, function prototypes)
- Source files (.c) typically contain implementation

Example header file wrapper:

```
#ifndef FILE_NAME_H
#define FILE_NAME_H
Header content
#endif
```

- Header content only compiled once
- Prevents redefinitions from multiple includes

Defines & C preprocessor

`#define MY_CONSTANT 5`

- Replaces every MY_CONSTANT with 5

`#define MY_STRING hello`

- Replaces every MY_STRING with hello

`#if MY_CONDITION/#else/#endif`

- Tests MY_CONDITION and conditionally compiles code

`#ifdef DEF/#endif (#ifndef DEF/#endif)`

- If DEF is defined (not defined), conditionally compile code

Constants

- Bad style: magic numbers in code
- Better style: `#define CONSTANT 5`
- Even better: `static const int CONSTANT = 5;`
  - Includes type information

Const with pointers:

- Pointer to const data: `const int* p;`
- Const pointer to variable data: `int* const p;`
- Const pointer to const data: `const int* const p;`