

# **CSC 2405: Computer Systems II**

Spring 2009 (MW 9:00-10:20 in G86)

Mirela Damian

Teaching Assistant: Nawar Molla

<http://www.csc.villanova.edu/~mdamian/csc2405/>

1

## **Goals for Today's Class**

---

- **CSC 2405 overview**
  - Goals of the course
  - Structure of the course
  - Learning the material
  - Assignments
  - Course grading
  - Academic policies

2

## What You Learn in This Course

---

- **Skill: shell and network programming**
  - Job management
  - Socket programming
  - Unix scripting
- **Knowledge: how systems work**
  - Multiprocessing and multithreading
  - Process scheduling
  - Memory management
  - Security threats and solutions

3

## Structure of the Course (Part I)

---

- **Review processes**
  - Creating and executing processes
- **Then study process management**
  - Jobs as collections of processes
  - Signaling jobs
- **Review threads**
  - Sharing global data
  - Synchronization with semaphores
- **Then study client-server communication**
  - Server is multi-threaded

4

## Structure of the Course (Part II)

---

- Memory management
  - Virtual and physical addressing
  - Cache hierarchies
  - Integrating cache and virtual memory
- System security
  - Worms, viruses, Trojan horses
  - Attacks and solutions
- Unix scripting

5

## Learning the Material: People

---

- Lecture (Mirela Damian)
  - When: MW 9:00-10:20 in G86
  - Office hours
    - M 1:30 – 2:30
    - Th 4:30 – 5:30
- Teaching Assistant
  - Nawar Molla
    - Office hours to be announced
    - E-mail: [nawar.molla@villanova.edu](mailto:nawar.molla@villanova.edu)
- Help Desk in MSC 292

6

## Learning the Material: Books

---

- Required textbook
  - Randal E. Bryant and David R. O'Hallaron, *Computer Systems - A Programmer's Perspective*
- Online resources
  - Class website
    - <http://www.csc.villanova.edu/~mdamian/csc2405/>
  - Links to other online resources
    - E.g. on socket programming

7

## Assignments

---

- Two major programming projects
  - One on Unix shells
  - One on HTTP proxies
- Several smaller assignments
- Submitting assignments
  - Follow submission instructions
  - Most often, printed copy of your code and a readme file
- Group work
  - None, unless specified in the assignment
  - Work is to be done *individually*

8

## Presentations

---

- Team presentations on system security
  - One or two students per team
  - Presentation topic due before April 1<sup>st</sup>

9

## Facilities for Programming

---

- Unix cluster
  - Machines: csgate, tanner, degas, cezanne, picasso, rodin, cassatt, gauguin, matisse
  - List displayed when logging into csgate
- Linux machine
  - felix
- Logging in to the machines remotely
  - SSH available for download from the CSC website

10

## Grading and Schedule

---

- Assignment (30% total)
- Two exams
  - Midterm exam in week seven (20%)
  - Final exam during exam period (30%)
- Presentation (10%)
- Quizzes (10%)

11

## Policies: Write Your Own Code

---

Programming in an individual creative process much like composition. You must reach your own understanding of the problem and discover a path to its solution. During this time, discussions with friends are encouraged. However, when the time comes to write code that solves the problem, such discussions are no longer appropriate - **the program must be your own work.**

12

## Policies: Write Your Own Code

---

If you have a question about how to use some feature of C, UNIX, etc., you can certainly ask your friends or the TA, but **do not, under any circumstances, copy another person's program.** Letting someone copy your program or using someone else's code in any form is a **violation of academic regulations.**

"Using someone else's code" includes using solutions or partial solutions to assignments provided by commercial web sites, instructors, preceptors, teaching assistants, friends, or students from any previous offering of this course or any other course.

13


**Okay, so let's get started  
with some background...**

14

# Structures in C - aggregate data type

int  
char  
int \*  
int []

```
typedef struct {
    ... /* variables in the structure */
} myStruct; /* don't forget this semi colon!!!! */
```

```
int x; 
myStruct m;
```

```
typedef struct
{
    int a;
    char b [10];
} name;
```

Define a new datatype

## Example

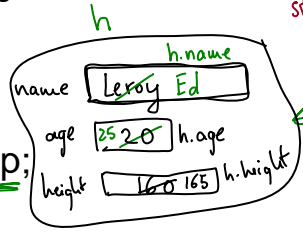
```
char * name;      char * name;      name = "Leroy";
name = "Leroy";  name = "Leroy";  name = "Leroy";
```

```
typedef struct {
    char name[20];
    int age;
    float height;
} Human;
```

h.age = 20;  
variable of type integer  
h.height = 160;

(\*hp).age = 25;  
hp -> age = 25;  
hp -> height = 165;

```
Human h;
Human * hp;
```



```
strcpy (h.name, "Leroy");
strcpy (hp->name, "Ed");
uint: hp = &h;
```



Exercise 1: Assign arbitrary values to h.

## Notice the difference ...

- Declaring a C struct normally

```
struct yourStruct{
```

```
...
```

```
};
```

```
struct yourStruct boohoo;
```

versus using the typedef-method

```
typedef struct {
```

```
...
```

```
} myStruct;
```

Old datatype → NEW  
 typedef (Float) Money;  
 Money m; /\* float m; \*/

```
myStruct haha;
```

17

## Exercise 2

Human h[2];

- Assign to the first human the values

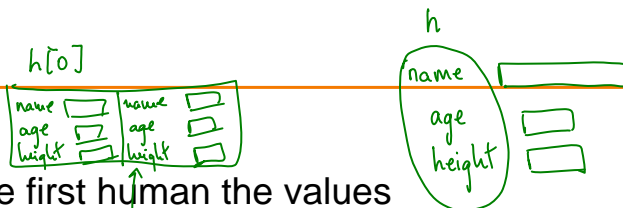
h[0].age = 20;  
 "Aname", 20, 160  
 h[0].height = 160;

- Using the pointer

Human \* hp;

assign to the second human the values

"Bname", 21, 162



```
strcpy(h[0].name, "Aname");
```

```
hp = &h[1];  
strcpy(hp->name, "Bname");
```

```
hp->age = 21;  
hp->height = 162;
```

18

```
struct job_t jobs [MAXJOBS];
```

```
clearjob (& jobs[i]);
```

```
initjobs (jobs);
```

```
clearjob (struct job_t * jobp)
```

```
{
```

```
  =
```

```
  =
```

```
  =
```

```
}
```

```
initjobs (struct job_t * joblist)
```

```
{
```

```
  =
```

```
  =
```

```
  =
```

```
}
```

