CS1003: Intro to CS, Summer 2008

Lecture #07
Data structures, OS

Instructor: Arezu Moghadam
arezu@cs.columbia.edu

Agenda

- Finish up data structure implementation
- Introduction to operating systems
- Introduction to networking
# Recap

- **Data structures**
  - Arrays
  - Lists
    - Stacks
    - Queues
  - Trees
    - Binary trees

- **How to implement data structures**

---

# Recap: Abstraction with data structures

- Primitive datatypes are too primitive
- Often, need to deal with large data
- Enable the application to access and manipulate complex data
- But main memory is not organized as arrays, lists or other proprietary datatypes
- Application doesn’t need to know about details of actual data storage
Lists

- A collection whose entries are arranged sequentially
- Beginning: head
- The end: tail

Head

![Linked list diagram](Image)

Head pointer

```
struct list_el {
    int val;
    struct list_el * next;
};

typedef struct list_el item;
```
Deleting an entry from a linked list

Inserting an entry into a linked list
Implementing heterogeneous arrays

typedef struct {
    char name[25];
    int age;
    float rate;
} employee;

typedef struct {
    char *name;
    int *age;
    float *rate;
} employee;

Binary trees

Implementation using a linked list
Agenda

- More on data structure implementation

How to implement data structures in C?

- Structs
- Used to define your own types
  ```c
  struct structure-name {
    field-type field-name;
    field-type field-name;
    ....
  } variable-name;
  ```
Structs II

- So an example would be
  ```c
  struct bin {
    char name [30];       // name of the part
    int quantity;         // how many in the bin
    int cost;             // the cost of the single part
  }
  printer_cable_bin;     // where we put the cables
  ```

- Here `printer_cable_bin` is a variable of type `struct bin`

- You can omit the variable name

Structs III

- The dot operator
  - In order to access one of the fields of the struct, for a particular variable, use the form `variable.field`
  - eg: `printer_cable_bin.cost = 1295;`
  - eg: `total_cost = printer_cable_bin.cost * printer_cable_bin.quantity`
Structs IV

- I said earlier that you don’t have to define variables when defining the struct
- So can I do, later in the code –
  - bin printer_cables_bin; (i.e. just like I use int or char)
  - Answer: No
- How to do it correctly
  - struct bin printer_cables_bin;
  - But this doesn’t define any of the values inside of bin, therefore those remain undefined
  - So you can either assign them one at a time or you can do the following

```c
struct bin printer_cable_bin = {
    "Printer Cables",
    0,
    1295
}; // However, this notation can only be used at the time of declaration
```
- Note the semicolons and the commas

---

Structs V

- (Shortcut) Initializing values –
  ```c
  struct bin {
      char name [30]; // name of the part
      int quantity; // how many in the bin
      int cost; // the cost of the single part
  }
  printer_cable_bin = {
    "Printer Cables",
    0,
    1295
  };
  ```
- Note the commas and the semicolon
Pointers to structures

```c
struct bin {int a; int b; int c;} ;
void function(struct bin);

main() {
    struct bin z;
    z.a = 10;
    z.a++;
    function(z);
}

Void function (struct bin x)
{
    printf("first member %d \n", x.a);
}
```

```
struct bin {int a; int b; int c;} ;
void function(struct bin *);

main() {
    struct bin *pz;
    pz->a = 10;
    pz->a ++;
    function(pz);
}

Void function (struct bin *x)
{
    printf("first member %d \n", x->a);
}
```

Add a new node to a linked list

```c
addnode(int value, struct NODE *position)
{
    struct NODE *new;
    if(new=(struct NODE*)malloc(sizeof(struct NODE)))
    {
        new->nodevalue = value;
        new->next = position->next;
        position->next=new;
    }
    else
        printf("Memory shortage");
}
```

![Diagram of linked list](image)
Delete a node from data structure

```c
int deletenode(struct NODE *target) {
    struct NODE *curr = list;
    while(curr->next != target) {
        curr = curr->next;
        curr->next = target->next;
        deletedval = target->value;
        free(target);
    }
    return(deletedval);
}
```

The big picture
The big picture (II)

- Given hardware and compiled (machine) code, you can run it directly, but that’s a huge hassle
  - What if you want to run multiple programs?
  - If so, how do we share resources between programs?
  - How do we let the user manipulate various programs?
  - How do we let *multiple users* manipulate various programs?
- Solution: employ a special piece of software that allows multiple user applications/tasks to cooperate

Operating System – OS

- Controls the operation of a computer
- Manages different tasks on behalf of the user
  - Store and retrieve files
  - Executing user’s programs
  - Networking tasks
  - Multi-processing, multi-user environment
  - Peripheral devices
- Examples; windows, UNIX, Mac OS and etc.
History of operating systems

- Batch processing: back in the single-task days, people would submit jobs to the computer for the entire company, and wait in line for their job to be done
  - Used a queue abstraction to handle the job list
  - No interactivity – submit job, wait for results
  - Very cumbersome for iterative development

- Interactive processing
  - Allow the user to interact
  - Still had to wait for your shot to use the computer
  - DOS – single user, single task

- Modern OSes multitask

Operating systems

- Considered system software, as compared to application software
  - The latter run as processes alongside an OS

- Two major components:
  - A kernel, which handles resource management, multitasking, etc. in the background;
  - A shell, which provides a user front-end to the operating system
Kernels

- Several important components
  - Device drivers: used to enable the OS to communicate with computer hardware
    - Device drivers abstract the hardware away from the OS, so that you can “plug-in” new drivers
  - Memory manager: Keeps track of computer’s memory allocation per process; also supports virtual memory, which enables the use of hard disks as additional memory
  - Scheduler: Control what tasks are running on the processor at any given time
  - Network stack: Provides networking facilities

The Linux kernel

- Popular learning kernel, since it’s open source
- You can grab your own copy from www.kernel.org, if you want to take a look
- A Linux operating system distribution (like Red Hat, Fedora) consists of the Linux kernel and a bunch of tools (including GNU tools)
- Here’s the directory structure of the kernel...
Multitasking

- Given multiple processes, coordinate them so that they can run concurrently.
- Well, not concurrently – the CPU handles a fixed number of instructions at any given time.
  - Instead, timeslice, so that each process does a little work at a time, and keep on switching.
  - Operating system keeps separate register sets, etc. for each application, and magically handles them cleanly for you.
  - “Virtual machine”: As an application designer, you feel like you have control over the machine, but the OS is actually managing many such processes.
Multitasking (II)

The “&” operator
- “emacs &” starts up emacs as a background process
- Lets you continue to use the shell while running emacs in its own window
- “jobs” lists the currently running jobs in the background

Or... multiple ssh sessions

The machine is actually handling all of these user sessions in parallel as collections of processes
- UNIX is multiuser, unlike older client versions of Windows

Run the command `top` to see all user processes running in the background
Process competition?

- What if two different processes need to access the same resource?
  - In the old days, if two programs want to print, you’d get a printout that was a mix of both
  - Now, a print spooler coordinates output and keeps them separate
  - The OS is responsible for handling such race conditions between processes

Process competition (II)

- More complicated resource contention requires locking; concept is similar to the barriers at a train track crossing
  - Semaphores == fancy locks implemented by flags
- Avoid deadlock:
Networks

- Now that we’ve discussed all the pieces on one computer, let’s talk about networking computers together
- More and more computing solutions are distributed across networks
- Several different kinds:
  - LAN (Local Area Network)
  - WAN (Wide Area Network)

LANs

- Most common LAN architecture today is Ethernet
- 10BASE-T/100BASE-T Ethernet use telephone-like wire to network computers together
  - Very cheap, and popular ("CAT 5" wiring)
- Topology: how to organize these networks?
  - Typically a hierarchical star topology nowadays
  - Columbia’s network is a hybrid of Ethernet and fiber
Next time

- Network
- AI