CS 3411 Systems Programming

Department of Computer Science
Michigan Technological University

C vs. C++ (cont.)

A Brief Look at Program Execution

- Text is executable code (also some strings! Usually write-protected)
- Data is *global* data (both initialized and uninitialized)
- Heap is area from which dynamic allocations are made (malloc!)
- Stack is where function *activation records* pushed/popped.
  - Pushed (created) on stack when function invoked, removed on return
  - May contain: function parameters, function locals, return address, temporaries, saved state, control link, access link
- Usual to preallocate a block of storage for initial heap/stack

Program Execution Example

```
#include <stdio.h>
#include <stdlib.h>

float gfloat = 10.5;

void funcA(int parm) {
  int lint;
  char *lptr;

  lint = 10l;
  lptr = (char *)malloc(parm);
  printf("lptr <%x>\n",lptr);
  printf("parm value <%d>\n", parm);
}

int main() {
  funcA(20);
  printf("gfloat <%f>\n",gfloat);
}
```

Calling Convention

- Where parameters and return values are placed
- How the caller and callee divide up the work
- How it's done in the C language:
- http://cm.bell-labs.com/cm/cs/who/dmr/clcs.html

Problems to Avoid

- It is always important to keep system programs as bug-free as possible
- Errant programs running in privileged mode can:
  - Access/modify system configuration files
  - Erase user data
  - Halt the system
  - And so on!

Buffer Overflow

- Writing beyond allocated array bounds

```
int getUserData() {
  char copy[60];
  ...
  /* User can input string of ANY length */
  gets(buf);
  ...
  /* Copies until string termination in buf */
  strcpy(copy,buf);
}

main() {
  ...
  char input[50];
  char *strPtr;
  ...
  getUserData();
  /* No string memory allocation for strPtr */
  strcpy(strPtr,input);
}
```

Memory Leak

- Losing access to allocated memory segment - We can't reclaim it!

```
int func()
{
  void *ptr;
  /* When function returns, value of ptr inaccessible */
  ptr = malloc(100);
}

main() {
  char *bptr;

  for (i=1;i<10;i++) {
    /* Previous ptr value overwritten each iteration */
    bptr = malloc(sizeof(char));
    *bptr = i;
  }
}
```

Dereference Invalid Pointer

```
...
int func(node *n) {
  if (n->value == 0) free(n);
  return (0);
}

main() {
  node *p,*q;
  p = malloc(sizeof(node));
  p->value = 10;
  printf("Node p value <%d>",p->value);

  func(p);
  /* p has already been freed */
  printf("After func p value <%d>\n", p->value);
  /* q was never initialized */
  printf("Node q value <%d>\n",q->value);
}
```

Printing out interesting errors

Look at 'man 3 perror'

```
#include <unistd.h>
#include <stdio.h>
#include <errno.h>

main() {
   int retCode;

   retCode = close(5);
   if(retCode < 0) perror("close");
   if(errno == EBADF) printf("Got␣EBADF␣as␣expected.\n");
}
```

GCC

- GNU Compiler and Linker
- Some options:
  - `-c` => compile but do not link
  - `-static` => prevent linking with shared libraries
  - `-g` => produce debugging information
- And many more in the manual!

The Unix File System

"...the most important role of the system is to provide a file system. Ritchie and Thompson, CACM ’74.
Types of files in Unix:

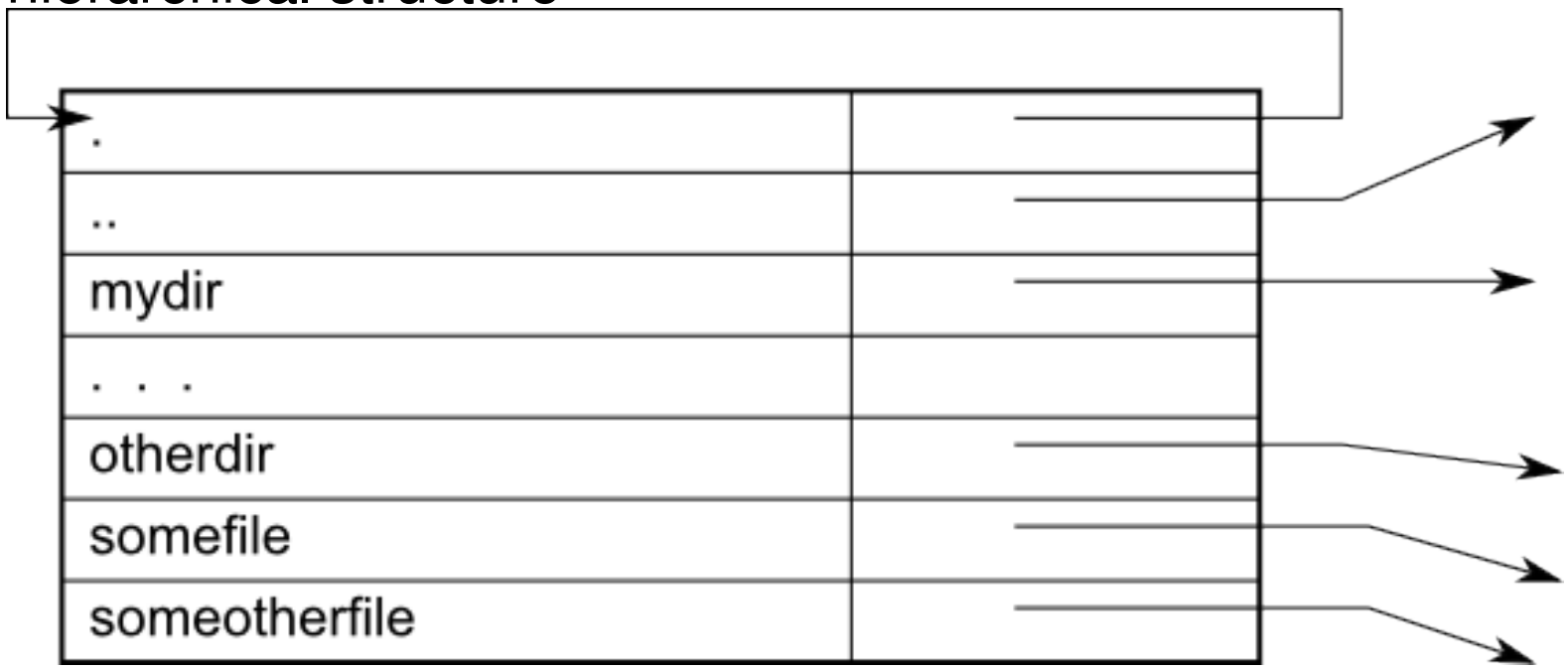
- Ordinary
- Directory
- Special (Character and Block)
- Symbolic Link
- FIFO
- Socket

Ordinary Files

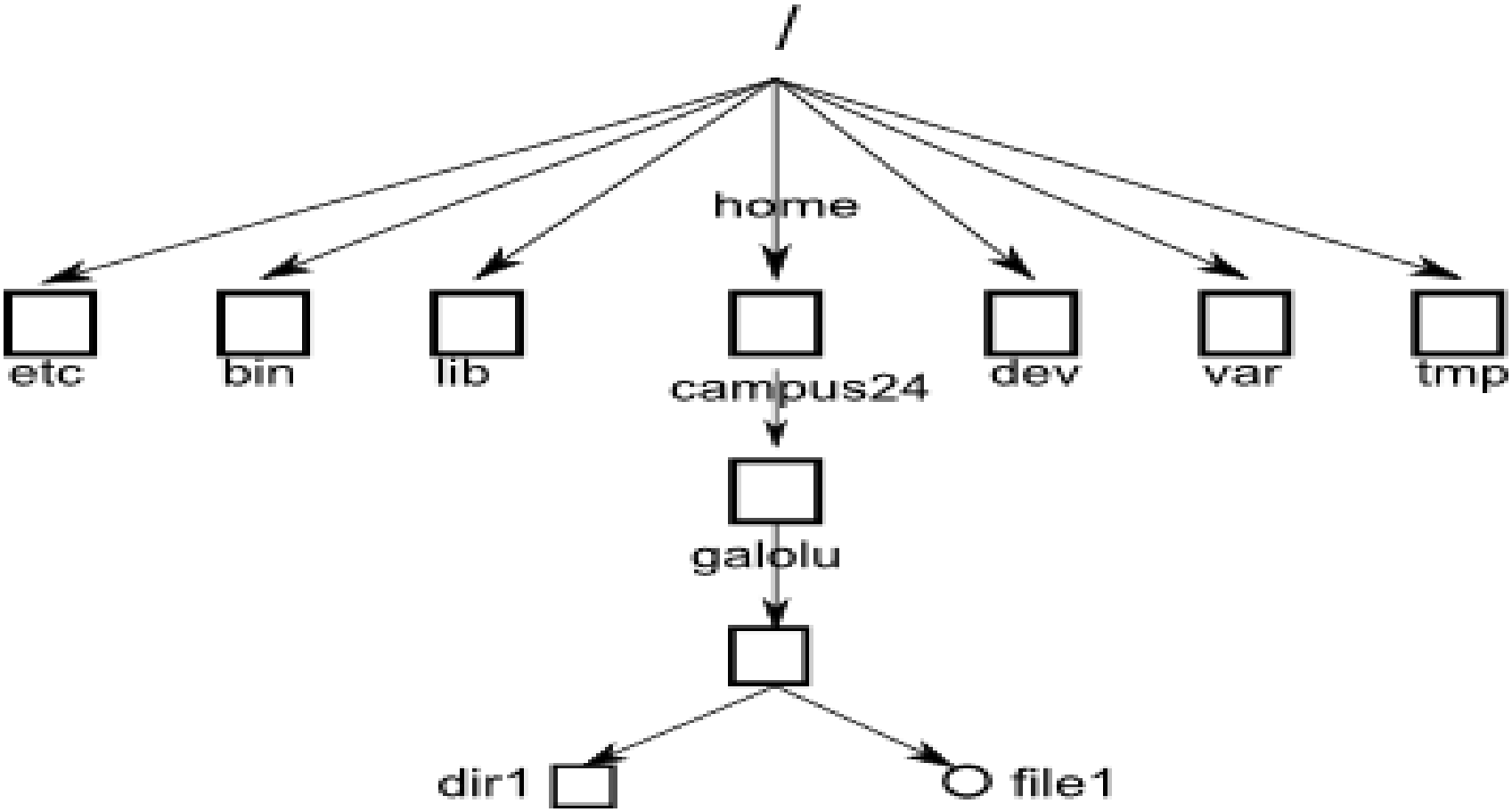
- Sequence of bytes
- Can be binary or text

Directories

- Is a file that contains information on contained files
- Maps symbolic file name onto "file descriptor"
- Writing a directory is done under strict system control
- Directories being in directories implies a tree-structured, hierarchical structure



Directories



- Absolute path names vs. Relative Path Names
- Home directory

Some Directory Modification Commands

- `cp f1 f2`
- `rm f2`
- `mv f1 f2`
- `mkdir d1`
- `rmdir d1`
- `ln f1 f2`
- `ln -s anything f2`

Special Files

- Makes physical devices appear to be part of file system hierarchy
- Provides uniform I/O interface (Can read or write just like an ordinary file!)
- Read/Write maps onto direct I/O on the device
- `/dev/sda` or `/dev/sda0` or `/dev/hdb2` or ...
- `/dev/tty`, `/dev/mem`, `/dev/kmem`, `/dev/null`

Next ..

- Links (Hard and Symbolic)
- FIFOs
- Protection
- Reading files!