

CS 354 - Machine Organization & Programming

Tuesday Feb 14 and Thursday Feb 16, 2023

Midterm Exam - Thursday, February 23th, 7:30 - 9:30 pm

- ♦ Room: Students will be assigned a room based and sent email with that room
- ♦ UW ID required
- ♦ #2 pencils required
- ♦ closed book, no notes, no electronic devices (e.g., calculators, phones, watches)
- ♦ see “Midterm Exam 1” on course site Assignments for topics
- ♦ see sample exam cover page (last page of this outline)

Project p2A: Due on or before Friday, Feb 17th

Project p2B: Due on or before Friday, Feb 24th

Homework hw1: Due on Monday Feb 13th (solution available Wed morning)

Homework hw2: Due on Monday Feb 20th (solution available Wed morning)

Last Week

Command-line Arguments Recall 2D Arrays 2D Arrays on the Heap 2D Arrays on the Stack 2D Arrays: Stack vs. Heap	Array Caveats Meet Structures Nesting in Structs and Arrays of Structs Passing Structures
--	--

This Week

Pointers to Structures (from L6) Standard & String I/O in <code>stdio.h</code> File I/O in <code>stdio.h</code> Copying Text Files Three Faces of Memory	Virtual Address Space C's Abstract Memory Model Meet Globals and Static Locals Where Do I Live? Linux: Processes and Address Spaces Exam Sample Cover Page
Next Week: The Heap & Dynamic Memory Allocators (p3) Read: B&O 9.1, 9.2, 9.9.1-9.9.6 9.1 Physical and Virtual Addressing 9.2 Address Spaces 9.9 Dynamic Memory Allocation 9.9.1-9.9.6	

Standard and String I/O in `stdio.h`

Standard I/O

Standard Input

`getchar` //reads 1 char
`gets` //reads 1 string ending with a newline char, BUFFER MIGHT OVERFLOW

`int scanf(const char *format_string, &v1, &v2, ...)`
reads formatted input from the console keyboard
returns number of inputs stored, or EOF if error/end-of-file occurs before any inputs

format string

format specifiers

whitespace

Standard Output

`putchar` //writes 1 char
`puts` //writes 1 string

`int printf(const char *format_string, v1, v2, ...)`
writes formatted output to the console terminal window
returns number of characters written, or a negative if error

format string

Standard Error

`void perror(const char *str)`
writes formatted error output to the console terminal window

String I/O

`int sscanf(const char *str, const char *format_string, &v1, &v2, ...)`
reads formatted input from the specified `str`
returns number of characters read, or a negative if error

`int sprintf(char *str, const char *format_string, v1, v2, ...)`
writes formatted output to the specified `str`
returns number of characters written, or a negative if error

File I/O in `stdio.h`

Standard I/O Redirection

File I/O

File Input

`fgetc/getc, ungetc` //reads 1 char at a time
`fgets` //reads 1 string terminate with a newline char or EOF

`int fscanf(FILE *stream, const char *format_string, &v1, &v2, ...)`
reads formatted input from the specified `stream`
returns number of inputs stored, or EOF if error/end-of-file occurs before any inputs

File Output

`fputc/putc` //writes 1 char at a time

`fputs` //writes 1 string

`int fprintf(FILE *stream, const char *format_string, v1, v2, ...)`
writes formatted output to the specified `stream`
returns number of characters written, or a negative if error

Predefined File Pointers

`stdin` is console keyboard
`stdout` is console terminal window
`stderr` is console terminal window, second stream for errors

Opening and Closing

`FILE *fopen(const char *filename, const char *mode)`
opens the specified `filename` in the specified `mode`
returns file pointer to the opened file's descriptor, or NULL if there's an access problem

`int fclose(FILE *stream)`
flushes the output buffer and then closes the specified `stream`
returns 0, or EOF if error

Copying Text Files

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

int main(int argc, char *argv[]) {

    if (argc != 3) {
        fprintf(stderr, "Usage: copy inputfile outputfile\n");
        exit(1);
    }

    FILE *ifp =
    if (ifp == NULL) {
        fprintf(stderr, "Can't open input file %s!\n", argv[1]);
        exit(1);
    }

    FILE *ofp =
    if (ofp == NULL) {
        fprintf(stderr, "Can't open output file %s!\n", argv[2]);

        exit(1);
    }

    const int bufsize = 257; //WARNING: assumes lines <= 256 chars
    char buffer[bufsize];

    return 0;
}
```

Three Faces of Memory

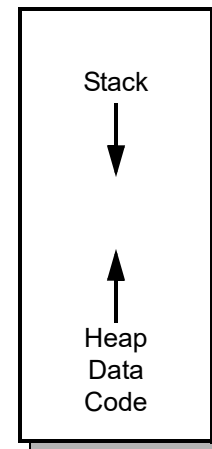
* Abstraction:

Process View = Virtual Memory

Goal:

virtual address space (VAS):

virtual address:

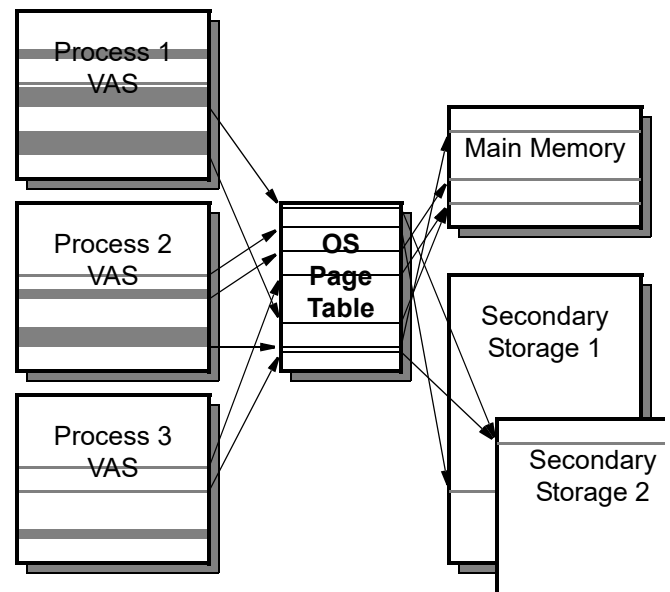


System View = Illusionist (CS 537)

Goal:

pages:

page table:

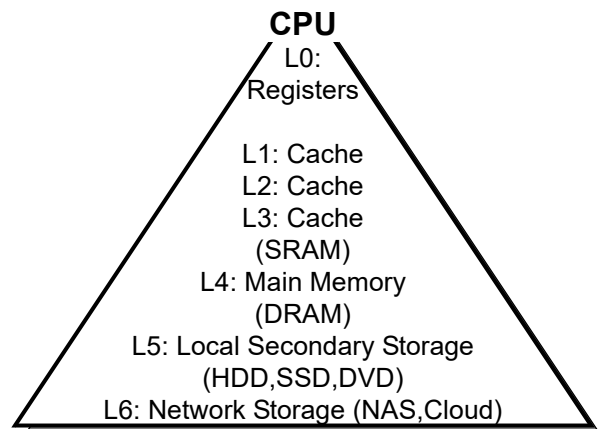


Hardware View = Physical Memory

Goal:

physical address space (PAS):

physical address:



Virtual Address Space (IA-32/Linux)

32-bit Processor = 32-bit Addresses => $2^{32} = 4,294,967,296 = 4\text{GB}$ Address Space

11111111111111111111111111111111 = 0xFFFFFFFF

address space:

process:

11000000000000000000000000000000 = 0xC0000000

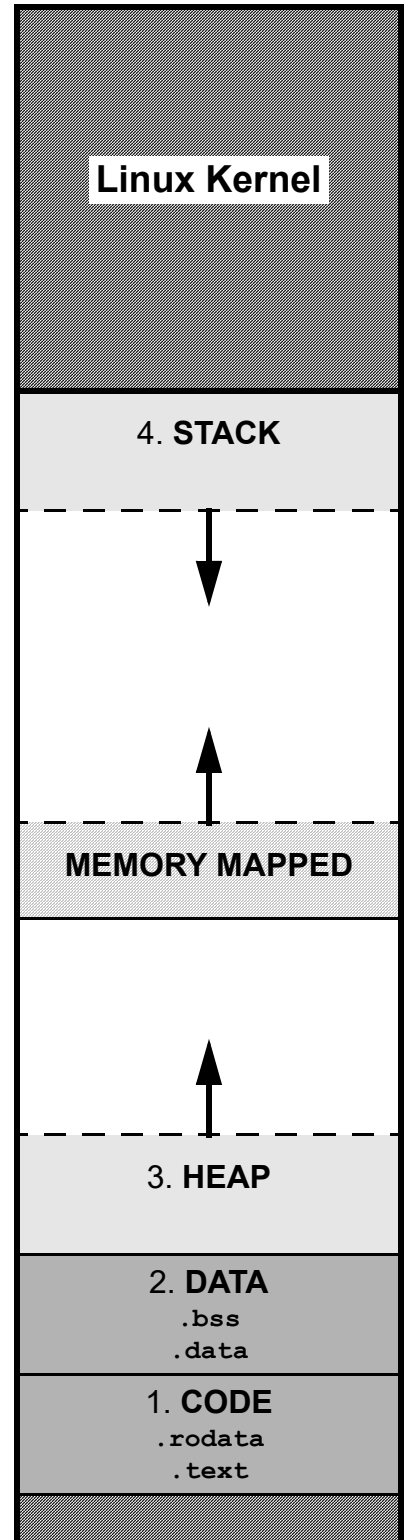
kernel:

user process:

* Every user process

00001000000001001000000000000000 = 0x08048000

00000000000000000000000000000000 = 0x00000000



C's Abstract Memory Model

1. CODE Segment

Contains:

.text section

.rodata section

Lifetime: entire program's execution

Initialization:

Access:

2. DATA Segment

Contains:

Lifetime: entire program's execution

Initialization:

.data section

.bss section

Access: read/write

3. HEAP (AKA Free Store)

Contains:

Lifetime:

Initialization:

Access: read/write

4. STACK (AKA Auto Store)

Contains:

stack frame (AKA activation record)

Lifetime:

Initialization:

Access: read/write

Meet Globals and Static Locals

What?

A global variable is

◆

◆

◆

A static local variable is

◆

◆

◆

Why?

✱ *In general, global variables
Instead use*

How?

```
#include <stdio.h>
int g = 11;

void f1(int p) {
    static int x = 22;
    x = x + p * g;
    printf("%d\n", x);
}

int main(void) {
    f1(g);
    g = 2;
    int g = 1;
    f1(g);
    return 0;
}
```

shadowing:

✱ *Avoid shadowing; don't use the same identifier*

Where do I live?

→ Identify the segment (and section) for each memory allocation in the code below.

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

int gus = 14;
int guy;

int madison(int pam) {

    static int max = 0;
    int meg[] = {22,44,88};
    int *mel = &pam;
    max = gus--;
    return max + meg[1] + *mel;
}

int *austin(int *pat){

    static int amy = 33;
    int *ari = malloc(sizeof(int)*44);
    gus--;
    *ari = *pat;
    return ari;
}

int main(int argc, char *argv[]) {

    int vic[] = {33,66,99};
    int *wes = malloc(sizeof(int));
    *wes = 55;
    guy = 66;
    free(wes);
    wes = vic;
    wes[1] = madison(guy);
    wes = austin(&gus);
    free(wes);
    printf("Where do I live?");
    return 0;
}
```

* *Arrays, structs, and variables*

Linux: Processes and Address Spaces

Process and Job Control

- ♦ Linux is

ps

jobs

&

ctrl+z

bg

fg

ctrl+c

Program Size

size <executable or object_file>

```
$gcc -m32 myProg.c
```

```
$size a.out
```

text	data	bss	dec	hex	filename
1029	276	4	1309	51d	a.out

Virtual Address Space Maps

- ♦ Linux enables

```
$pmap <pid_of_process>
```

```
$cat /proc/<pid_of_process>/maps
```

```
$cat /proc/self/maps
```

/proc:

[] Lec 001 9:30am TR		
[] Lec 002 1:00pm TR		
Lecture	Print Netid	Print Name (first last)

Computer Sciences 354
Midterm Exam 1 Secondary
Thursday, October 6th, 2022
60 points (15% of final grade)
Instructors: Debra Deppeler

1. MARK an X in box by your lecture number above.
2. PRINT your NET ID (UW login name not your photo id number) in box above.
3. PRINT your first and last name in box above.
4. FILL-IN all fields and their bubbles on the scantron form (use # 2 pencil).
 - (a) LAST NAME -fill in your last (family) name starting at leftmost column.
 - (b) FIRST NAME -fill in first five letters of your first (given) name.
 - (c) IDENTIFICATION NUMBER is your UW Student ID number.
 - (d) Under ABC of SPECIAL CODES, write your lecture number as a three digit value 001 or 002.
 - (e) Under F of SPECIAL CODES, write the number 2 for Secondary and fill in the number (2) bubble.
5. DOUBLE-CHECK THAT YOU HAVE FILLED IN ALL ID FIELDS and that you have FILLED IN ALL CORRESPONDING BUBBLES ON SCANTRON.
6. Taking this exam indicates that you agree: to not write answers in large letters and to keep your answers covered; to not view or use another's work or any unauthorized devices in any way; to not make any type of copy of any portion of this exam; and that you understand that being caught doing any of these actions, or other actions that permit any student to submit work that is not wholly their own will result in automatic failure of the exam and possible failure of the course. Penalties are reported to the Deans Office for all involved.

Parts	Number of Questions	Question Format	Possible Points
I	10	Simple Choice	20
II	10	Multiple Choice	30
III	2	Written	10
	22	Total	60

Assumptions unless instructions explicitly state otherwise:
addresses and integers are 4 bytes.
code questions are about C and IA-32/x86 assembly code on our Linux platform.

Reference: Powers of 2

$$2^5 = 32, 2^6 = 64, 2^7 = 128, 2^8 = 256, 2^9 = 512, 2^{10} = 1024$$

$$2^{10} = K, 2^{20} = M, 2^{30} = G$$

$$2^A * 2^B = 2^{A+B}, 2^A / 2^B = 2^{A-B}$$

Turn off and put away all electronic devices and
wait for the proctor to signal the start of the exam.