CS 354 - Machine Organization & Programming
Tuesday Sept 24 and Thursday Sept 26, 2024

Midterm Exam - Thursday, October 3rd, 7:30 - 9:30 pm

- **Room: Students will be assigned a room 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**

PM BYOL: Start p2A and p2B if you have not yet started either

Activity A04: due on or before this week Saturday, A05 is also available

Homework hw1: Due on or before this week Monday (solution available Wed morning)

Homework hw2: Due on or before next week Monday

Project p2A: Due on or before this week Friday, Sep 27

Project p2B: Due on or before next week Sunday, Oct 6th

Week 4 Learning Objectives (at a minimum be able to)

- use **<stdio.h>** functions: **printf**, **scanf**, **fopen**, **fclose**, **fgets**, **fputs**
- use predefined file pointers: **stdin** and **stdout**
- use format specifiers: **%c %f %i %d %s %p %x**
- use Linux I/O redirection at the command line: < input_file > output_file >> append_file
- describe C's abstract memory model: **Process View = Virtual Memory**
- diagram C's abstract memory model: **CODE**, **DATA**, **HEAP**, **STACK**
- meet IA-32 memory hierarchy: **Hardware View = Physical Memory**
- understand difference and use of **global** vs **static local** variables

This Week

2D array on stack (video and notes)
Pointers to Structures (from last week)
Standard & String I/O in `stdio.h`
File I/O in `stdio.h`
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
which may be fewer than expected, or even zero in the event of an early matching failure

format string contains format specifiers and chars in input to skip

format specifiers such as %d, %f, %p %s, %i (reads octal/decimal/hex),
each format specifier must match its corresponding destination variable

whitespace input separator (space, tab, newline), leading whitespace is skipped

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 contains format specifiers and chars to display
recall \n flushes output buffer
each format specifier must match its corresponding source variable

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 in Linux terminal (shells)

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 Files

`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

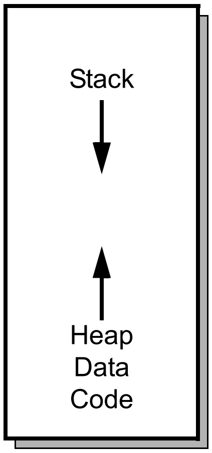
```
#include <stdio.h>
#include <stdlib.h>

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: Provide a simple view of memory.

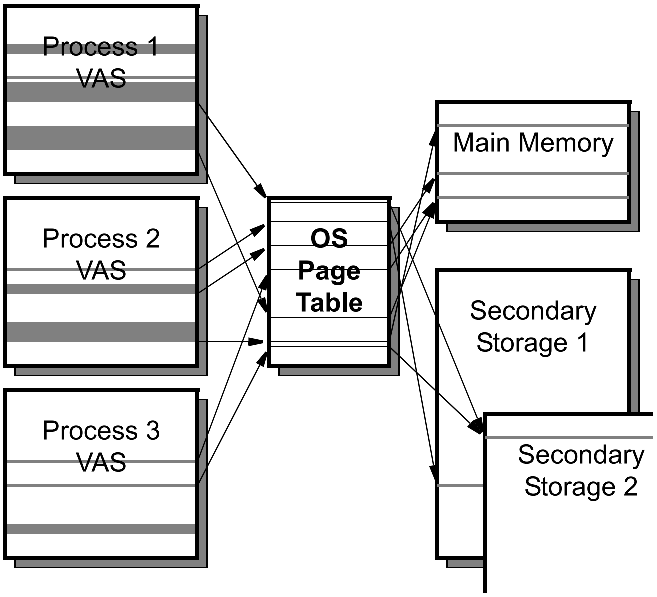


virtual address space (VAS):

virtual address:

System View = Illusionist (CS 537)

Goal: Make memory sharable and secure.



pages:

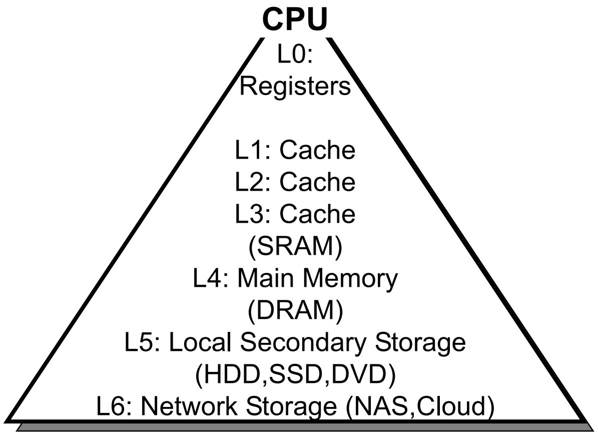
page table:

Hardware View = Physical Memory

Goal: Keep the CPU busy.

physical address space (PAS):

physical address:



Virtual Address Space (IA-32/Linux)

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

```
11111111111111111111111111111111 = 0xFFFFFFFF
```

address space:

process:

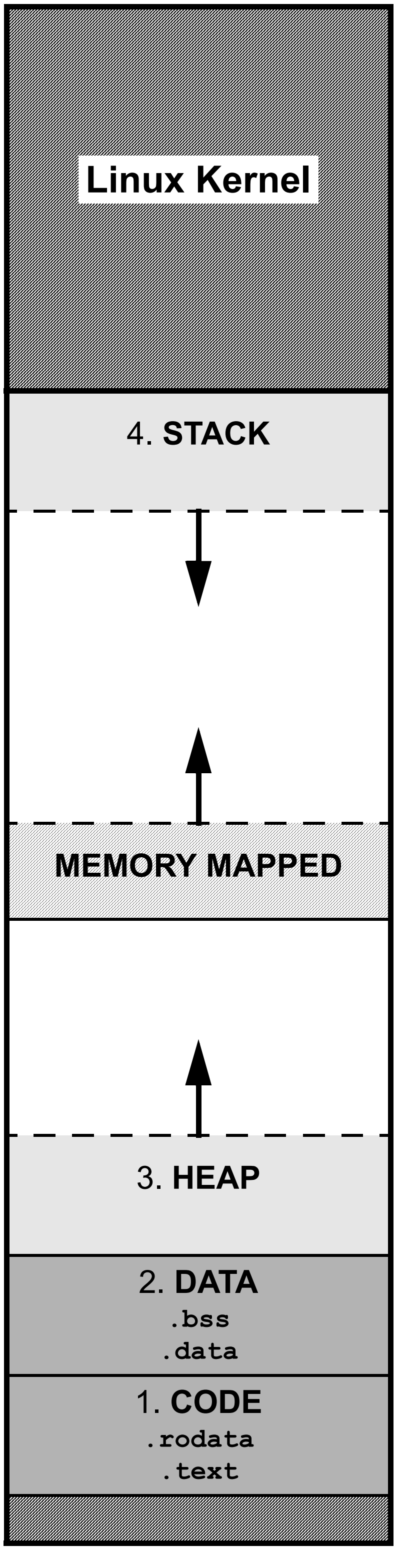
```
11000000000000000000000000000000 = 0xC0000000
```

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*

Pointer variables can

Linux: Processes and Address Spaces

Process and Job Control

- Linux is a multitasking OS where you can run multiple processes concurrently.

`ps` lists a snapshot of all the user’s processes, for everyone’s processes: `ps -e`

`jobs` lists only the processes the user started from the command line

`&` put a process in the background
ctrl+z suspend running process

`bg` put suspended process in the background
`fg` bring a process to the foreground

ctrl+c stop running foreground process

Program Size

size <executable or object_file>

displays size of a program’s mem segs (.text, .data, .bss) and the total

```
$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 you to see the VAS (memory map) of each process

```
$pmap <pid_of_process>
```

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

magic number, stack, libraries, vDSO (Virtual Dynamically linked Shared Objects)

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

notice heap

/proc: virtual filesystem that reveals kernel data in ASCII text form can be read by progs

SPEC CODES EF 10	UW LOGIN NAME	Last, First Name (as in email and on scantron)

Computer Sciences 354

Midterm Exam 1 Primary

Thursday, October 5th, 2023

60 points (15% of final grade)

Instructor: Debra Deppeler

1. **RECORD Special Codes for EF on scantron. Ask Proctor if there are not 2 digits.**.
2. **PRINT your UWNET ID (login name not photo id number) in box above**.
3. **PRINT Last, First Name in box above**.
4. **SCANTRON Fill in all fields and their bubbles on the scantron form (must use #2 pencil on scantron form).**
   (a) LAST NAME field - left align last name as given in room email
   (b) FIRST NAME field - left align first five letters of your first name as given in email
   (c) IDENTIFICATION NUMBER your UW Student WiscCard ID number
   (d) SPECIAL CODES E - write and fill-in bubble for your exam version number 1
   (e) SPECIAL CODES F - write and fill-in bubble for your room number 0
5. **FILL IN BUBBLES FOR ALL IDENTIFICATION FIELDS and for SPECIAL CODES COLUMNS E and F.**
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 may 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	2 pt Simple Choice	20
II	12	3 pt Multiple Choice (+bonus)	36
III	4	Survey	4
	28	Total	60

Assumptions unless instructions explicitly state otherwise:

+ addresses and integers are 4 bytes unless explicitly stated otherwise.
+ code questions are about C std=gnu99 and IA-32 on our Linux platform

Reference: Powers of 2

$2^5 = 32$, $2^6 = 64$, $2^7 = 128$, $2^8 = 256$, $2^9 = 512$, $2^{10} = 1024$
$2^{10} = \mathrm{K}$, $2^{20} = \mathrm{M}$, $2^{30} = \mathrm{G}$
$2^{\mathrm{A}} * 2^{\mathrm{B}} = 2^{\mathrm{A+B}}$, $2^{\mathrm{A}} / 2^{\mathrm{B}} = 2^{\mathrm{A-B}}$

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