CS 354 - Machine Organization & Programming
Thursday, September 26, 2019

Midterm Exam - Thursday, October 3rd, 7:15 - 9:15 pm

- **Lec 1 (2:30 pm):** room 3650 of Humanities
- **Lec 2 (4:00 pm):** room B10 of Ingraham Hall
- UW ID required
- closed book, no notes, no electronic devices (e.g., calculators, phones, watches)
- see "Midterm Exam 1" on course site Assignments for topics

Project p2A (3%) DUE: 10 pm, Monday, September 30th

Project p2B (3%) DUE: 10 pm, Monday, October 7th

TIP: Use blank outlines to study for the exam.

Homework hw1 (1.5%) DUE TOMORROW: 10 pm, Friday, September 27th

Homework hw2 (1.5%) DUE: 10 pm, Wednesday, October 2nd

Last Time

Array Caveats
Command-line Arguments
Meet Structures
Nested Structures and Arrays of Structures
Passing Structures
Pointers to Structures

Today

Pointers to Structures (from last time)
Standard & String I/O and `stdio.h`
File I/O and `stdio.h`
Copying Text Files

Three Faces of Memory
Virtual Address Space
C's Abstract Memory Model
Where Do I Live?

Next Time

Globals and Static Locals
Linux Processes and Address Spaces
---------- END of Exam 1 Material -----------
The Heap & Dynamic Memory Allocators
Read: B&O 9.9.4 - 9.9.5

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

Standard I/O

Standard Output (console)

```
putchar
puts
int printf(const char *format_string, comma-separated-list-of-vars)
```

returns number of characters written, or a negative if error

format string

format specifiers

Standard Input (keyboard)

```
getchar
gets
int scanf(const char *format_string, comma-separated-list-of-var-addrs)
```

returns number of inputs successfully matched and assigned

format string

whitespace

Standard Error (console)

```
void perror(const char *str)
```

String I/O

```
int sprintf(char *str, const char *format, ...)
```

```
int sscanf(const char *str, const char *format, ...)
```

File I/O in `stdio.h` Library

Standard I/O Redirection

File I/O

File Output

```
fputc/~~putc~~
fputs
int fprintf(FILE *stream, const char *format, ...)
```

returns number of characters written, or a negative if error

File Input

```
fgetc/~~getc~~, ungetc
fgets
int fscanf(FILE *stream, const char *format, ...)
```

returns non-negative value, or EOF if error

File Pointers and Descriptors

`stdin`, `stdout`, `stderr`

Opening and Closing

```
FILE *fopen(const char *filename, const char *mode)
```

returns open file pointer, or NULL if access problem

```
int fclose(FILE *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, *ofp;

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


      exit(1);
   }

   ofp =
   if (ofp == NULL) {
      fprintf(stderr, "Can't open output file %s!\n", argv[2]);
      exit(1);
   }



   const int bufsize = 257;
   char buffer[bufsize];









   return 0;
}
```

Three Faces of Memory

✵ *A key OS goal is*

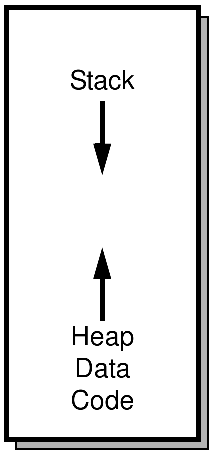
process:

Process View = Virtual Memory

Goal:

virtual address space (VAS):

virtual address:

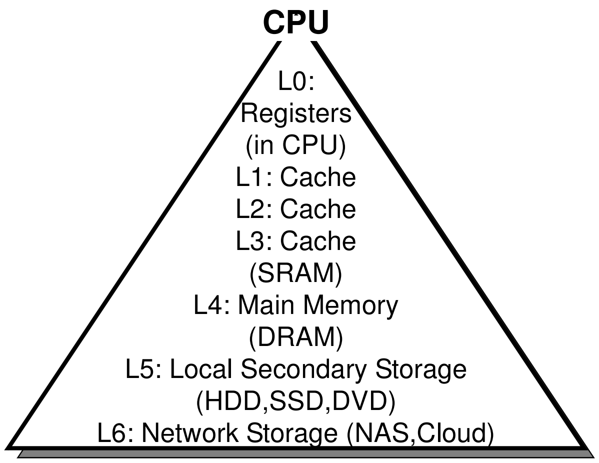


Hardware View = Physical Memory

Goal:

physical address space (PAS):

physical address:

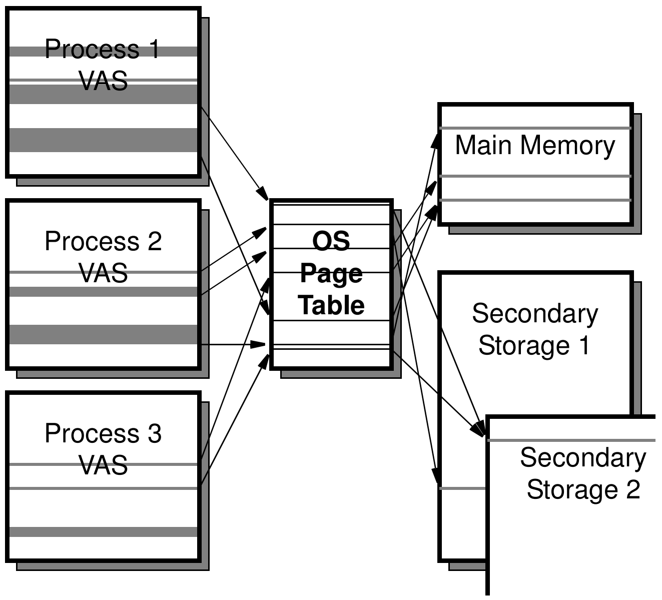


System View = Illusionist (CS 537)

Goal:

pages:

page table:

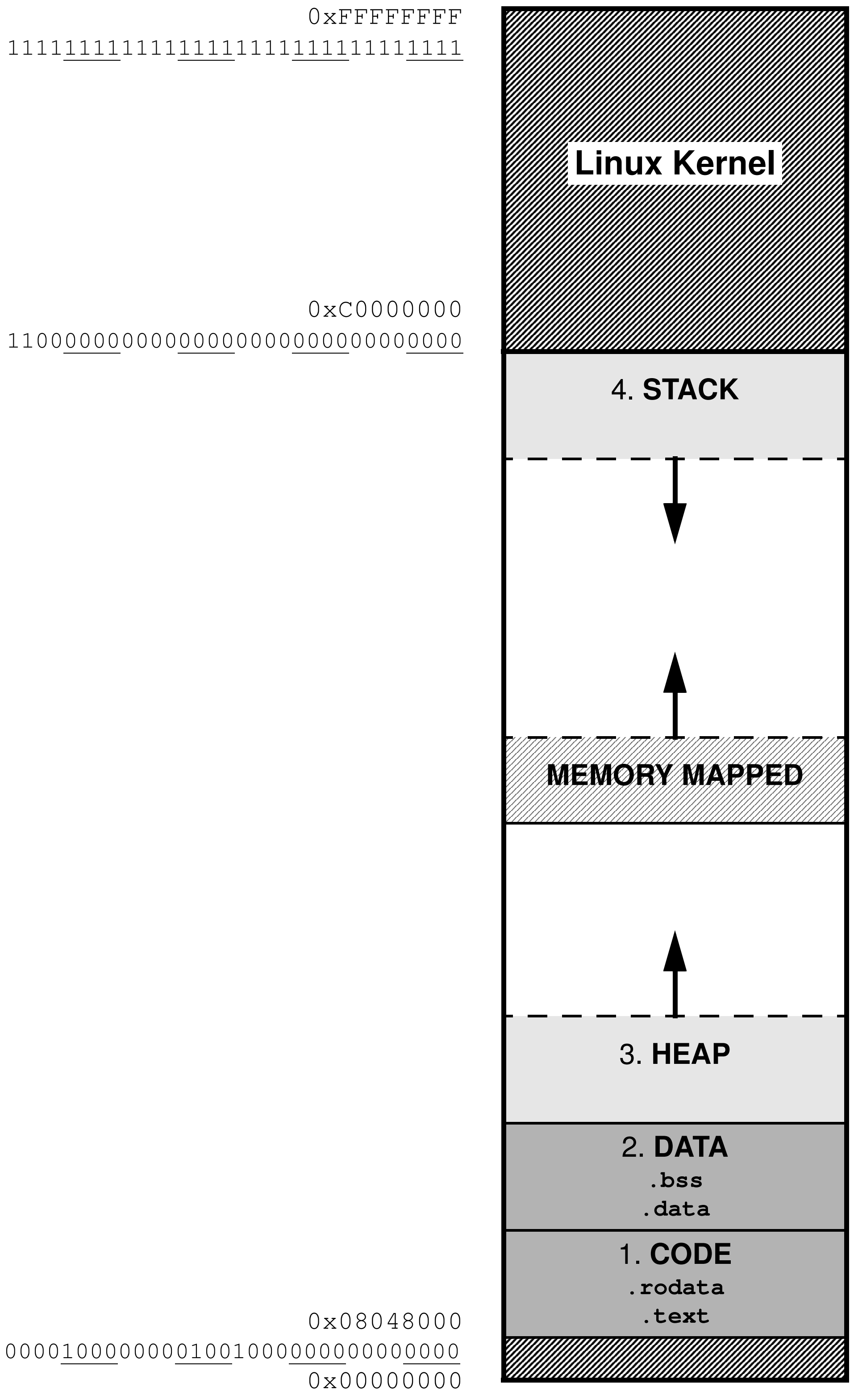


A Process' Virtual Address Space (IA-32/Linux)

32-bit Processor = 32-bit Addresses

recall: *byte addressability*: each address accesses 1 byte
max addressable bytes: 2^{32} = 4,294,967,296 = 4GB

Kernel:



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:

3. HEAP (AKA Free Store)

Contains:

Lifetime:

Initialization:

Access:

4. STACK (AKA Auto Store)

Contains:

<u>*stack frame*</u> (AKA activation record)

Lifetime:

Initialization:

Access:

Where do I live?

```
#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, pointers*

- *Pointer variables can store any address but*