**CS 354 - Machine Organization & Programming**

**Tuesday Feb 13th and Thursday Feb 15th, 2024**

**Midterm Exam - Thursday, February 22nd, 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**

**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, Feb 16

**Project p2B:** Due on or before next week Friday, Feb 23

**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**

|  |  |
| --- | --- |
| 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

*format string* contains format specifiers and characters to skip

*format specifiers %d %f %p %s % (must mastch corresponding destination variable)*

*whitespace* input separator (space, tab, newline)

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* format specifiers and chars to display (Use \n to flush buffer)

Standard Error

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

writes formatted error output to the console terminal window

**String I/O can read from and write to a string instead of terminal and keyboard**

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 (a.out < input\_file >(rewrites) output\_file or >>(append) output\_file)**

**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 = fopen(\*(argv+1), “r”);

if (ifp == NULL) {

fprintf(stderr, "Can't open input file %s!\n", argv[1]); exit(1);

}

FILE \*ofp = fopen(argv[2], “w”);

if (ofp == NULL) {

fprintf(stderr, "Can't open output file %s!\n", argv[2]);

fclose(ifp);

exit(1); }

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

while(fgets(buffer,bufsize, ifp) != NULL){

fputs(buffer,ofp);

if(fclose(ifp)==NULL){print(“error msg”);exit(1);} if (fclose(ofp)==NULL){printf(“err msg”); exit(1);}

}

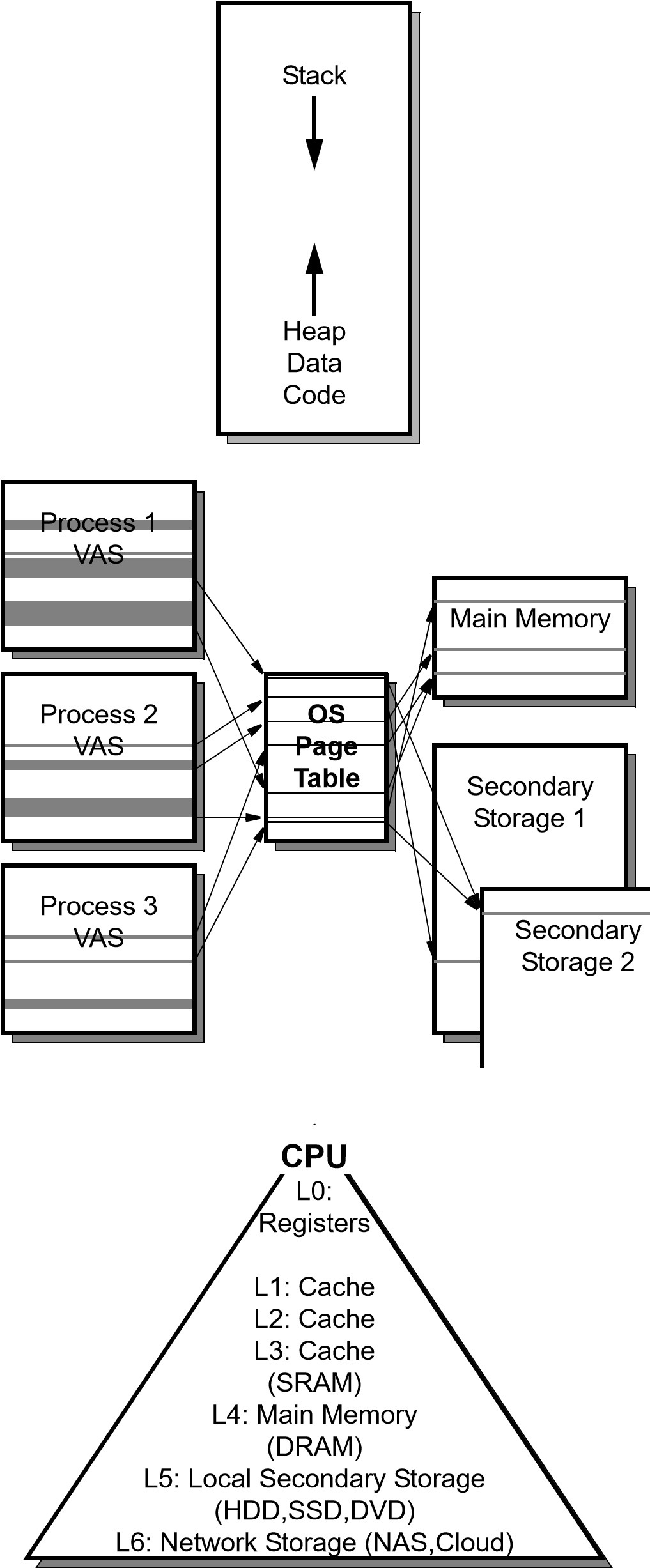
return 0;

}

copy src\_filename to dst\_filename

# Three Faces of Memory

 *Abstraction: manage complexity by focusing on relevant details*

**Process View = Virtual Memory**

Goal: Provide a simple view

*virtual address space (VAS)*:

Illusion by O.S. that each process has its own continuous memory block

*virtual address*: a simulated address a process generates

**System View = Illusionist (CS 537)**

Goal: make memory shareable and secure

*pages*: 4 kb units of storage, fixed size, contiguous

*page table*: OS data structure, maps virtual pages to physical pages, keeps processes from interfering with other



**Hardware View = Physical Memory**

Goal: keep CPU busy

*physical address space (PAS)*:

*multi*-level hierarchy

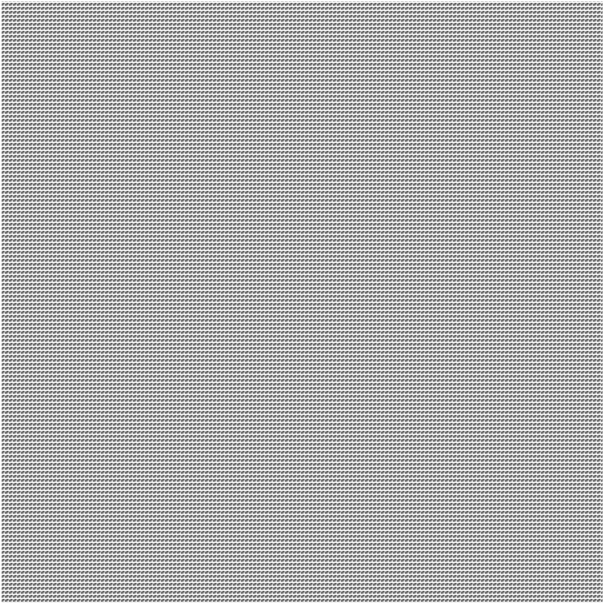
*freq accessed data in mem closest to cpu*

*physical address*:actual addr used to access memory

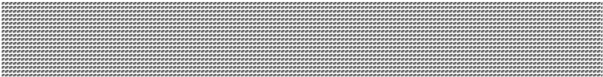
**Virtual Address Space (IA-32/Linux)**

**32-bit Processor = 32-bit Addresses => 232 = 4,294,967,296 = 4GB Address Space**

11111111111111111111111111111111 = 0xFFFFFFFF *address space*: range of valid mem addr for process



**Linux Kernel**



1

.

**CODE**

**.rodata**

**.text**

.

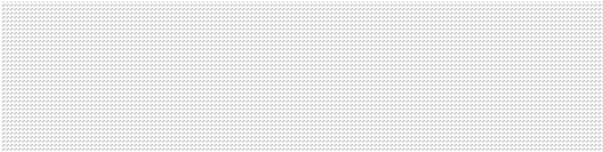
3

**HEAP**

4

.

**STACK**



**MEMORY MAPPED**

2

.

**DATA**

**.bss**

**.data**

*process*: a running program

*Windows IA*-32 2GB of O.S., ser gets 2GB

11000000000000000000000000000000 = 0xC0000000

*kernel*: mem resident portion of O.S

*user process*: process that is not kernel

 *Every user process has this simple view of memory –Makes coding easier*

Section

Section



00001000000001001000000000000000 = 0x08048000

00000000000000000000000000000000 = 0x00000000

# C’s Abstract Memory Model

1. **CODE Segment**

Contains: program code, binary machine code

**.text** section machine code

**.rodata** section string literals

Lifetime: entire program’s execution

Initialization: by loader from executable object file before execution begins

Access: READ – ONLY

1. **DATA Segment**

Contains: global variables and static locals

Lifetime: entire program’s execution

Initialization: by loader from executable object file before execution

**.data** section vars that are init to non-zero

**.bss** section (Block Started by Symbol) vars that are not initialized or initialized to zero

Access: read/write

1. **HEAP** (AKA Free Store)

Contains: memory allocated and freed by programmer during execution

Lifetime: managed by programmer, malloc/calloc/realloc free

Initialization: none be default

Access: read/write

1. **STACK** (AKA Auto Store)

Contains: memory in stack frames, auto allocated and freed by code generated by compiler

*stack frame* (AKA activation record) non-static local variable, parameters

Lifetime: from time declared until end of scope

Initialization: none by default

Access: read/write

# Meet Globals and Static Locals

**What?**

A *global variable* is

 declared outside of any function

 accessible to all functions in src file

 allocated in the data segment

A *static local variable* is

 declared inside a function with static modifier

 accessible only within the function

 allocated in DATA segment

**Why?** For storage that exists during entire runtime

 *In general, global variables should not be used*

*Instead use local variables that are passed to callee functions*

**How?**

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

void f1(int p) { static int x = 22; //Static local

x = x + p \* g; printf("%d\n", x);

}

int main(void) {

f1(g); g = 2;

int g = 1;

//Non static Local, “shadowed” global var g

f1(g); return 0; } *shadowing*: when local var blocks access to global variable

 *Avoid shadowing; don’t use the same identifier for local and global var*

**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; //Data .data

int guy; //Data .bss

int madison(int pam //Stack) {

static int max = 0; //Data .bss

int meg[] = {22,44,88}; //SAA, Stack

int \*mel = &pam; //Stack

max = gus--;

return max + meg[1] + \*mel;

}

int \*austin(int \*pat //Stack ){

static int amy = 33; //Data .data

int \*ari = malloc(sizeof(int)\*44); //Ari on stack, blue on heap

gus--;

\*ari = \*pat;

return ari;

}

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

int vic[] = {33,66,99}; //Stack

int \*wes = malloc(sizeof(int)); //Wes on stack, blue on heap

\*wes = 55;

guy = 66;

free(wes);

wes = vic;

wes[1] = madison(guy);

wes = austin(&gus);

free(wes);

printf("Where do I live?"); //Code

return 0;

}

 *Arrays, structs, and variables can live in Data, Heap, or Stack segments*

*Ptrs can store any addr, but SEG Fault to dereference to addr without permission*

# Linux: Processes and Address Spaces

**Process and Job Control**  Linux is multitasking OS

ps lists snapshot of user processes

jobs

& ctrl+z

bg fg

ctrl+c

**Program Size**

size <executable or object\_file> displays size of programs mem segment

$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 VAS (mem map) of each process

$pmap <pid\_of\_process>

$cat /proc/<pid\_of\_process>/maps magic numbers, stack, libraries

$cat /proc/self/maps notice heap

/proc: virtual file system that reveals kernel data in ascii

|  |  |  |
| --- | --- | --- |
| 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).**
   1. LAST NAME field - left align last name as given in room email
   2. FIRST NAME field - left align first five letters of your first name as given in email
   3. IDENTIFICATION NUMBER your UW Student WiscCard ID number
   4. SPECIAL CODES E - write and fill-in bubble for your exam version number 1
   5. 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**.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Number of | Question | Possible |
| Parts | Questions | Format | Points |
| I | 10 | 2 pt Simple Choice | 20 |
| II | 12 | 3 pt Multiple Choice (+bonus) | 36 |
| III | 4 | Written | 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

25 = 32, 26 = 64, 27 = 128, 28 = 256, 29 = 512, 210 = 1024

210 = K, 220 = M, 230 = G

2A \* 2B = 2A+B, 2A / 2B = 2A-B

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

©2023 Deppeler CS354 Exam 1 P Page 1 of 7