CS252: Systems Programming (Summer 2026)

People Lab Sessions Help Hours Textbook Course Info
Class Slides Class Schedule Lab Handouts Homeworks Exams Class Slides

TEACHER

Dr. Gustavo Rodriguez-Rivera
E-mail: grr@purdue.edu
Office Hours: You may talk to me immediately after the in-person lectures
Office: DSAI 1139C

Lecture:
10:30 AM - 11:30 AM | Felix Haas Hall | Room G066

GTAS

Sikang	Sun	sun1017@purdue.edu

UTAS

West Lafayette:		
Benjamin	Lobos Lertpunyaroj	blobosle@purdue.edu
Lillith	Erickson	ericks63@purdue.edu

LAB SESSIONS

		Lab Session	Head TA	Associate TA
L01		TR 02:00 PM - 03:50 PM: Felix Haas Hall Room: G040	Sikang Sun	Benjamin Lobos Lertpunyaroj, Lillith Erickson
L02		TR L02 12:00 PM - 01:50 Felix Haas Hall Room: G040	Sikang Sun	Benjamin Lobos Lertpunyaroj, Lillith Erickson

Help Hours

IMPORTANT: Also consider attending any of the labs to ask questions and get help. Students registered in the lab have seating priority, but most of the time TAs will have slots available.

After Hours Lab Help Hours (Walk-Ins)

BoilerQ: https://courses.cs.purdue.edu/queue/queue/719

Date and Time	Room (or link)	TAs
Friday 2 - 4 PM	HAAS G072	Sikang Sun (no session first week)

Open Lab Help Hours -
Students can work in a lab and get help from TAs

Date & Time (EST)	Location (Lab)	TA(s)

Date & Time (EST)	Location (Lab)	TA(s)

Note: TAs will allocate the first half of the session to common questions in a shared room. The second half will be for private sessions. You may request a private session during the first half of the help hours.

Syllabus

West Lafayette: HTML
Indianapolis: on Brightspace

Class Homepage

An updated syllabus, class handouts, lecture notes, and other information can be found at:
http://www.cs.purdue.edu/homes/cs252

Textbook

- Book Online: *"Introduction to Systems Programming: a Hands-on Approach" by Gustavo Rodriguez-Rivera and Justin Ennen*
- Recommended: Advanced Programming in the UNIX Environment by W. Richard Stevens. (Useful for the shell. Good as a reference book.)

Goal

After successfully completing this course:

- You will consolidate the programming skills from the previous core courses.
- You will understand how programs run in user space and how they interact with the OS.
- You will be able to write large programs (>1000 lines).
- You will be able to use tools like IDEs, debuggers, profilers, and source control to write good and maintainable code.
- You will learn how to work in teams.
- You will learn how to use Scripting Languages.
- You will learn to write multi-process and multi-threaded programs.

  Note: This course does not cover OS internals. That will be covered in the Operating Systems Course.

Course Organization

The course is organized in lecture sessions and lab sessions. You are required to attend the labs every week, as that is where instructors will explain details of the class projects and answer questions. Lab attendance will count toward part of the attendance grade.

Lists and Announcements:

All announcements will be sent via email and/or posted in Ed Discussion.

Grade Distribution

The final grade will be 30% midterm 30% final exam, 30% projects and homeworks, 10% attendance.

Syllabus

- Address space. Structure of a Program. Text, Data, BSS, Stack Segments.
- Assembly Language Review
- Review of Pointers, double pointers, pointers to functions
- Use of an IDE and debugger to program in C and C++.
- Memory allocation. Problems with memory allocation. Memory Leaks, Premature Frees, Memory Smashing, Double Frees.
- Executable File Formats. ELF, COFF, a.out.
- Development Cycle, Compiling, Assembling, Linking. Static Libraries
- Loading a program, Runtime Linker, Shared Libraries.
- Scripting Languages. sh, bash, basic UNIX commands.
- File creation, read, write, close, file mode.
- IO redirection, pipes
- Fork, wait, waitpid, signals.
- Directories, creating, directory list.
- Project: Writing your own shell.
- Programming with Threads, thread creation.
- Race Conditions, Mutex locks.
- Socket Programming.
- Iterative and concurrent servers.
- Introduction to SQL
- Source Control Systems (CVS, SVN) and distributed (GIT, Mercurial)
- Introduction to Software Engineering
- Design Patterns
- Execution Profiling.

Lecture Videos

- Lectures are available in Brightspace
- Course primer videos
- Course lab videos

Books

- *"Introduction to Systems Programming: a Hands-on Approach" by Gustavo Rodriguez-Rivera and Justin Ennen*

Labs

Lab0: GDB and Introduction to x86-64 Assembly Programming Lab 0 Quiz Example Question	Week 1: due Thursday .6/18/2026 during your lab time. You will show gdbtokens.txt Week 2 due Thursday 6/25/2026 before your lab time. See the grading form.
Lab1: Implementing your own malloc	Milestone 1: Test_simple0 and test_simple1 Due Tuesday 6/30/2026 before your Lab time. Milestone 2: At least 50pts in tests. Due Thursday 7/2/2026 before your lab time. Milestone 3. Final submission. Tuesday 7/7/2026 before your lab time

Late Policy

Intermediate milestones of a lab do not have late submissions. Show during your lab time what you have the day of your lab for partial grade, milestones typically count only for 10% of the total grade of a lab.

The late penalty for a final submission is 10 points per day. You will be graded during your lab time the following week using the git commit of the day you completed your work. You may not be graded at labs other than your assigned lab section.

Any extension for health reasons or other extraordinary reasons have to be discussed with Gustavo after lecture or by email. Attach any doctor's note to your email.

Homeworks Exams

Class Slides

- All Class Slides

Other Slides

- What happened on Mars (slides 40 - 50)
- Emergency Preparedness

Week to Week Schedule (Approximated)

Summer session: Every 2 weeks corresponds to one week of the Summer session.

Week	Topic	Slides	Textbook
Week 1	Sections of a Program C/C++ Review	1-60	Chapter 0, 1, 2
Week 2	Explicit	61-124	Chapter 0, 1, 2

	Memory Management Using a Debugger (Presentation of Lab1 Handout)	125-138	
Week 3	The UNIX Operating System	139-187	Chapter 0, 1, 2
Week 4	Common UNIX Commands and Shell Scripting (Presentation of Lab2 Handout)	188-213	Chapter 3
Week 5	Unix System Programming and The Shell Project (Command Table and Parsing) (Presentation of Lab3 Handout)	214-238	Chapter 3, 4
Week 6	Unix System Programming and The Shell Project (System Calls and Execution)	238-273	Chapter 4,5
Week 7	Wildcards, Signals, Subshells	274-300	Chapter 4,5
Week 8	User Mode, Kernel Mode, Interrupts and System Calls	301-358	Chapter 4,5

Week 9	Threads and Thread Synchronization Mutex Locks (Presentation of Lab4 Handout)	359-404	
Week 11	Midterm Exam - (Includes Everything before Threads) Semaphores	405-441	
Week 11	Deadlock Prevention, Condition Variables	442-487	
Week 12	The Internet and Socket Programming. (Presentation of Lab5 Handout)	488-538	
Week 13	The Internet and Socket Programming. Programming with Sockets,	539-568	
Week 14	SQL and relational Databases Software Development in Teams	569-588 589-617	Chapter 9
Week 15	Software Patterns, Program Optimization	618-634 635-641	Chapter 10
Week 16	Final Review		

Suggested Videos and Materials

- History of Node.js
- Steve Jobs on Why Sometimes Big Companies Make Bad Products.
- Linus Torvalds Talking about the C Programming Language
- The "C" Programming Language: Brian Kernighan
- CEI-CERT C coding standard
- GIT Techtalk by Linus Torvalds
- Linux TED Talk
- History of GNU/LINUX explained by Richard Stallman and Linus Torvalds
- James Gosling on the Creation of Java
- James Gosling Interview
- Bill Atkinson, creator of Quickdraw and MacPaint in the first Mac.
- The Origins of Linux—Linus Torvalds
- Bjarne Stroustrup: Why I Created C++
- Old Video on UNIX - shows Kernighan, Ritchie and Thompson. - AT&T Archives film from 1982, Bell Laboratories
- Joel Spolsky Co-creator of Stack-Overflow Interview
- Steve Jobs The Lost Interview (in Netflix) - Very good business and technology insights.
- The Story of the Sputnik Moment
- Design philosophy - Sandy Munro
- Linus Torvalds Interview 1998
- Linus Torvalds on Design vs. Prototyping
- The Struggle of Building the Original iPhone - The Untold Story
- The Creation of TurboPascal
- Writing a simple device technology driver for the Raspberry Pi
- Steve Wozniak and the Genesis of Video Games.in PCs
- John Carmack, video game developer and creator of Doom, and Quake and founder of Oculus VR, on why you should use a debugger when programming. And the whole interview (5hrs with lots of good advice for programmers)
- Good Explanation of pointers
- History of Computer Graphics 1990
- Premature optimization
- Linus vs Linus
- Git will make sense after this
- Exploiting buffer overflows
- Aho on the creation of Yacc
- History of Chrome
- Roadmap of Computer Languages
- History of turbopascal and typescript

- Carl Sagan on the Encyclopedia of Life
- One rule in the kernel: don't break user space.
- If it is a bug people rely on, it is a feature.
- Ffmpeg history
- Xerox Park history

Suggested Links

- Dennis Ritchie Homepage at Bell Labs
- "The Development of the C Language" by Dennis M. Ritchie
- Coding Standards

Course Policies

Academic Integrity

Instruction		Effect	Description
`movq`	S, D	$D \leftarrow S$	Move quad word
`movabsq`	I, R	$R \leftarrow I$	Move quad word
`movslq`	S, R	$R \leftarrow$ SignExtend(S)	Move sign-extended double word
`movsbq`	S, R	$R \leftarrow$ SignExtend(S)	Move sign-extended byte
`movzbq`	S, R	$R \leftarrow$ ZeroExtend(S)	Move zero-extended byte
`pushq`	S	R[`%rsp`] $\leftarrow$ R[`%rsp`] $- 8$; M[R[`%rsp`]] $\leftarrow S$	Push
`popq`	D	$D \leftarrow$ M[R[`%rsp`]]; R[`%rsp`] $\leftarrow$ R[`%rsp`] $+ 8$	Pop

Instruction		Effect	Description
`leaq`	S, D	$D \leftarrow$ `&`S	Load effective address
`incq`	D	$D \leftarrow D$ `+ 1`	Increment
`decq`	D	$D \leftarrow D$ `- 1`	Decrement
`negq`	D	$D \leftarrow$ `-`D	Negate
`notq`	D	$D \leftarrow$ `~`D	Complement
`addq`	S, D	$D \leftarrow D$ `+` S	Add
`subq`	S, D	$D \leftarrow D$ `-` S	Subtract
`imulq`	S, D	$D \leftarrow D$ `*` S	Multiply
`xorq`	S, D	$D \leftarrow D$ `^` S	Exclusive-or
`orq`	S, D	$D \leftarrow D$ `|` S	Or
`andq`	S, D	$D \leftarrow D$ `&` S	And
`salq`	k, D	$D \leftarrow D$ `<<` k	Left shift
`shlq`	k, D	$D \leftarrow D$ `<<` k	Left shift (same as `salq`)
`sarq`	k, D	$D \leftarrow D$ `>>` k	Arithmetic right shift
`shrq`	k, D	$D \leftarrow D$ `>>` k	Logical right shift

Instruction		Effect	Description
`imulq`	S	R[`%rdx`]:R[`%rax`] $\leftarrow S \times$ R[`%rax`]	Signed full multiply
`mulq`	S	R[`%rdx`]:R[`%rax`] $\leftarrow S \times$ R[`%rax`]	Unsigned full multiply
`cltq`		R[`%rax`] $\leftarrow$ SignExtend(R[`%eax`])	Convert `%eax` to quad word
`cqto`		R[`%rdx`]:R[`%rax`] $\leftarrow$ SignExtend(R[`%rax`])	Convert to oct word
`idivq`	S	R[`%rdx`] $\leftarrow$ R[`%rdx`]:R[`%rax`] $\bmod S$; R[`%rax`] $\leftarrow$ R[`%rdx`]:R[`%rax`] $\div S$	Signed divide
`divq`	S	R[`%rdx`] $\leftarrow$ R[`%rdx`]:R[`%rax`] $\bmod S$; R[`%rax`] $\leftarrow$ R[`%rdx`]:R[`%rax`] $\div S$	Unsigned divide

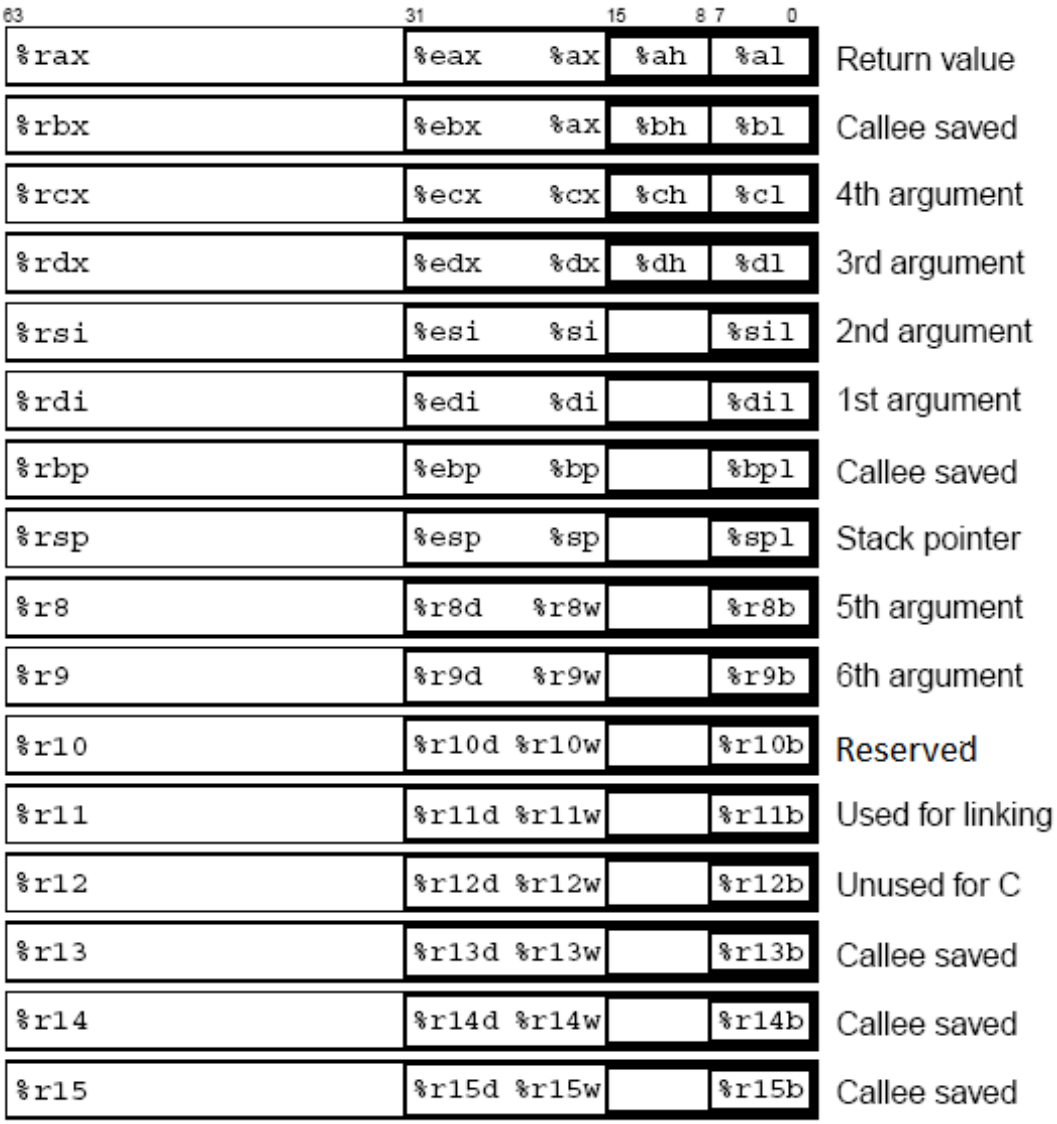
-

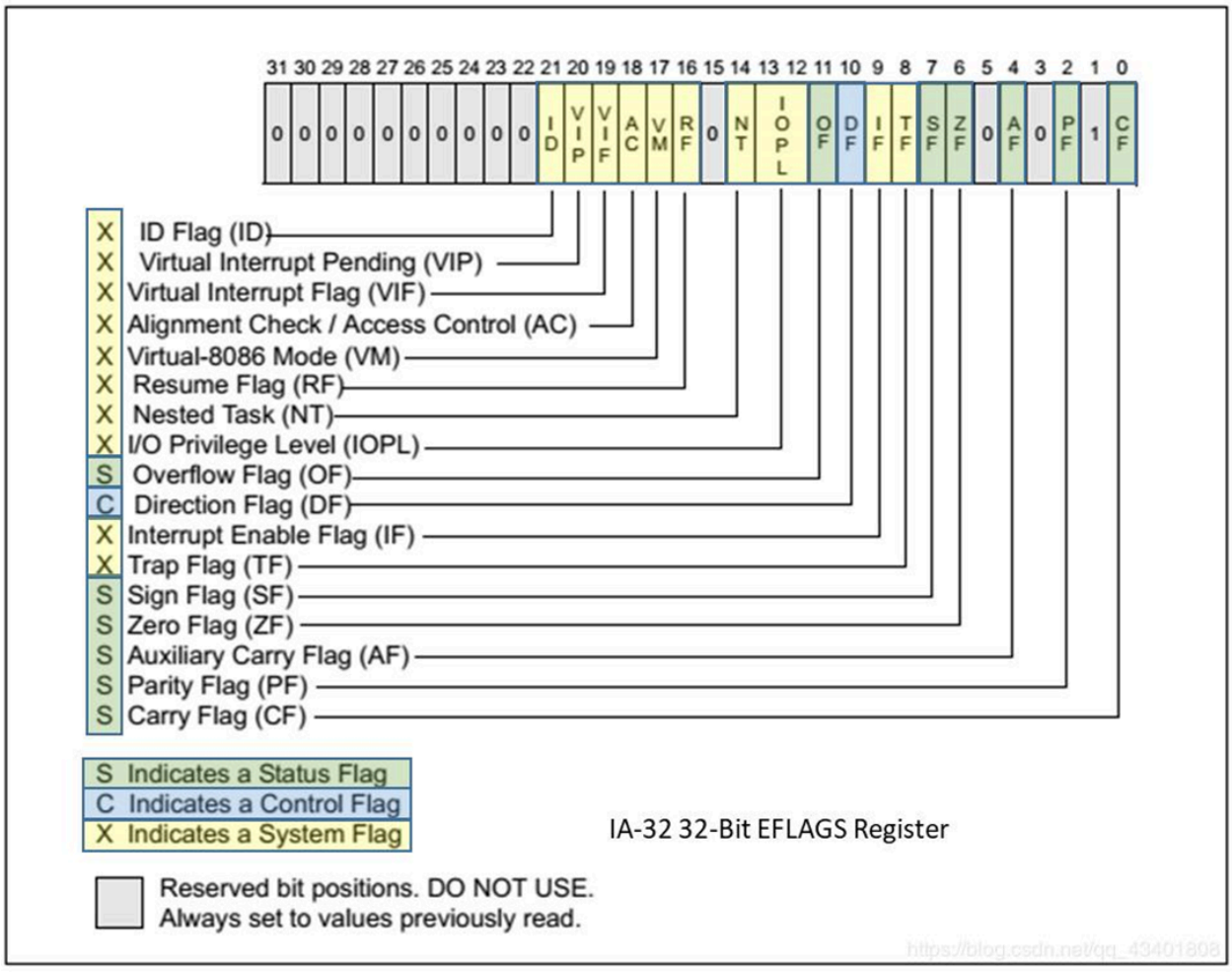
Instruction		Based on	Description
`cmpq`	S_2, S_1	S_1 - S_2	Compare quad words
`testq`	S_2, S_1	S_1 & S_2	Test quad word

Instruction		Synonym	Move condition	Description
`cmove`	S, D	`cmovz`	`ZF`	Equal / zero
`cmovne`	S, D	`cmovnz`	`~ZF`	Not equal / not zero
`cmovs`	S, D		`SF`	Negative
`cmovns`	S, D		`~SF`	Nonnegative
`cmovg`	S, D	`cmovnle`	`~(SF ^ OF) & ~ZF`	Greater (signed >)
`cmovge`	S, D	`cmovnl`	`~(SF ^ OF)`	Greater or equal (signed >=)
`cmovl`	S, D	`cmovnge`	`SF ^ OF`	Less (signed <)
`cmovle`	S, D	`cmovng`	`(SF ^ OF) | ZF`	Less or equal (signed <=)
`cmova`	S, D	`cmovnbe`	`~CF & ~ZF`	Above (unsigned >)
`cmovae`	S, D	`cmovnb`	`~CF`	Above or equal (Unsigned >=)
`cmovb`	S, D	`cmovnae`	`CF`	Below (unsigned <)
`cmovbe`	S, D	`cmovna`	`CF | ZF`	below or equal (unsigned <=)

Mnemonic	Required Flag State	Description
JGE JNL	SF = OF	Jump near if greater or equal Jump near if not less
JNG JLE	ZF = 1 or SF <> OF	Jump near if not greater Jump near if less or equal
JNLE JG	ZF = 0 and SF = OF	Jump near if not less or equal Jump near if greater

Stack Position	Register/Memory
0	rbx
1	r10
2	r13
3	r14
4	r15
>=5	Use the execution stack


63
31
15
8 7
0
%rax
%eax
%ax
%ah
%al
Return value
%rbx
%ebx
%ax
%bh
%bl
Callee saved
%rcx
%ecx
%cx
%ch
%cl
4th argument
%rdx
%edx
%dx
%dh
%dl
3rd argument
%rsi
%esi
%si
%sil
2nd argument
%rdi
%edi
%di
%dil
1st argument
%rbp
%ebp
%bp
%bpl
Callee saved
%rsp
%esp
%sp
%spl
Stack pointer
%r8
%r8d
%r8w
%r8b
5th argument
%r9
%r9d
%r9w
%r9b
6th argument
%r10
%r10d
%r10w
%r10b
Reserved
%r11
%r11d
%r11w
%r11b
Used for linking
%r12
%r12d
%r12w
%r12b
Unused for C
%r13
%r13d
%r13w
%r13b
Callee saved
%r14
%r14d
%r14w
%r14b
Callee saved
%r15
%r15d
%r15w
%r15b
Callee saved



Immediate Value

```
movq $0x501208,%rdi  #Put in register %rdi the
                       # constant 0x501208
                       #  %rdi = 0x501208
```

Direct Register Reference

```
movq %rax,%rdi        #Move the contents of
                      #register %rax to %rdi
                      #  %rdi = %rax
```

Indirect through a register

```
movq %rsi,(%rdi )     #Store the value in %rsi
                    #in the address contained in %rdi
                    # *(%rdi) = %rsi
```

Direct Memory Reference

```
movq 0x501208,%rdi   #Fetch the contents in memory
                     #at address 0x501208 and store it

                     #in %rdi
                      # %rdi = *(0x501208)
```

MALLOC FUNCTIONS AND DATA STRUCTURES

```
enum state { UNALLOCATED = 0, ALLOCATED = 1, FENCEPOST = 2 };

typedef struct header {
  size_t size_state;
  size_t left_size;
  union {
    // Used when the object is free
    struct {
      struct header * next;
      struct header * prev;
    };
    // Used when the object is allocated
    char data[0];
  };
} header;

extern void * base;
extern header freelistSentinels[];
extern char freelist_bitmap[];
extern header * osChunkList[];
extern size_t numOsChunks;
extern header * lastFencePost;
extern void * base;

size_t get_size(header * h);
void set_size(header * h, size_t size);
enum  state get_state(header *h);
void set_state(header * h, enum state s);
void set_size_and_state(header * h, size_t size, enum state s);
header * get_right_header(header * h);
header * get_header_from_offset(void * ptr, ptrdiff_t off);
header * get_left_header(header * h);
header * ptr_to_header(void * p);
void initialize_fencepost(header * fp, size_t left_size);
void insert_os_chunk(header * hdr);
void insert_fenceposts(void * raw_mem, size_t size);
header * allocate_chunk(size_t size);
void deallocate_object(void * p);
header * allocate_object(size_t raw_size);
```