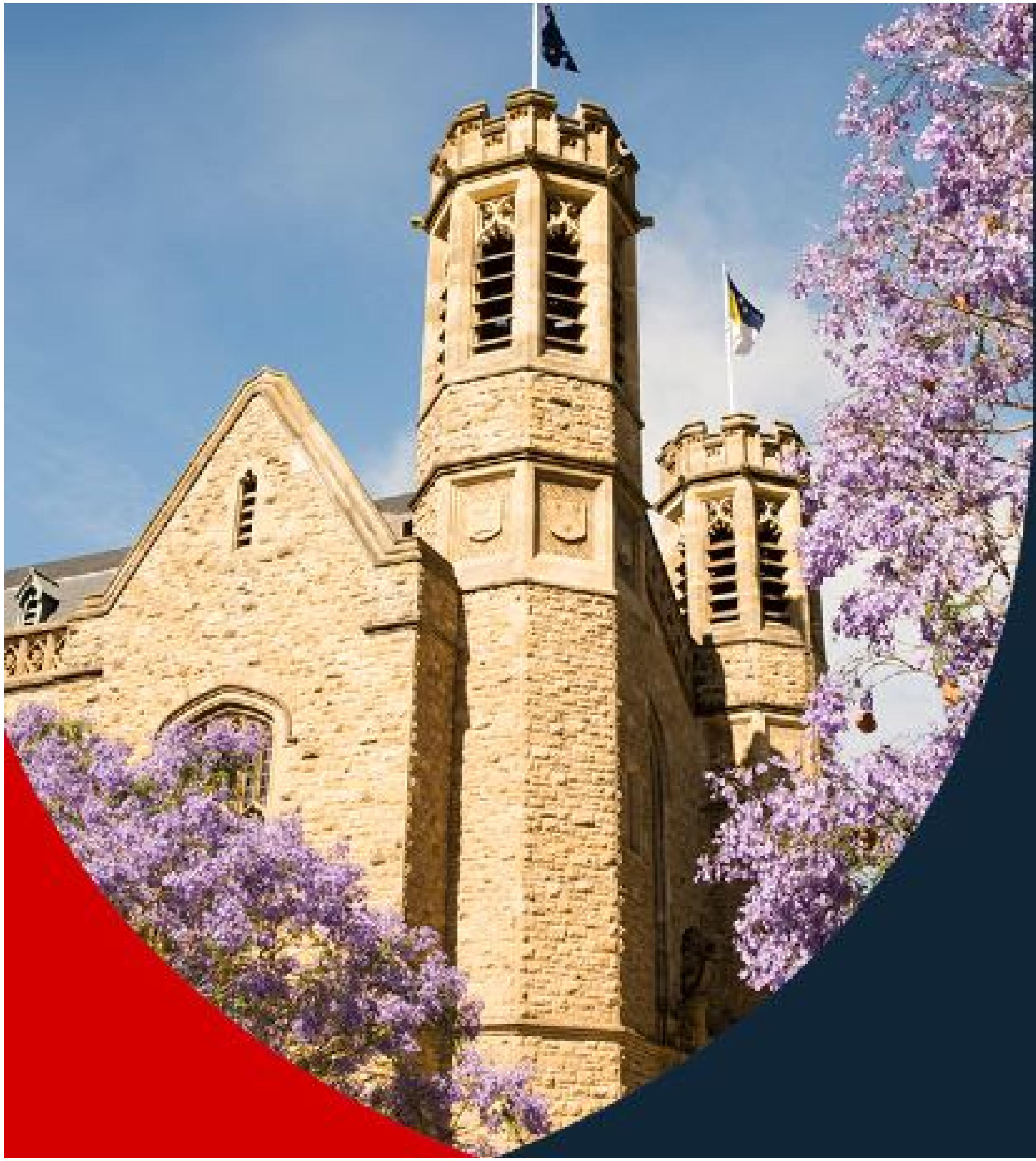


Operating Systems

COMP SCI 3004

COMP SCI 7064

Week 1 – Introduction

make history.

THE UNIVERSITY of ADELAIDE

We acknowledge and pay our respects to the Kaurna people, the traditional custodians whose ancestral lands we gather on.

We acknowledge the deep feelings of attachment and relationship of the Kaurna people to country and we respect and value their past, present and ongoing connection to the land and cultural beliefs.

Lecturers

(Course Coordinator) **Cruz Izu**

Olaf Maennel (Course Lecturer)

TA's: Deepak Bhargavan Pillai, Linet Maria Cherian, Kartikeya Mishra

THE UNIVERSITY of ADELAIDE

Learning Outcomes

- Explain the role of the operating system as a high-level interface to the hardware.
- Use OS as a resource manager that supports multiprogramming.
- Explain the low-level implementation of CPU dispatch.
- Explain the low-level implementation of memory management.
- Explain the performance trade-offs inherent in OS implementation.

THE UNIVERSITY of ADELAIDE

Assessment Criteria

Passing the course

- **Written Exam 50%**
- **Hurdle: You need at least 40% of the mark of the written examination to pass this course.**
- If your mark for any component with a hurdle is less than 40% of the allocated marks for that component, and your overall mark is greater than 45 F, your overall mark will be reduced to 45 F.
- To pass the course, you must obtain a passing mark overall and achieve at least 40% of the available marks in components with a hurdle.

THE UNIVERSITY of ADELAIDE

Passing the course

- Three Practical Assignments (38%)
  - “System calls” 08% DUE: *11 Aug 2023* (Week 3)
  - “Memory” 15% DUE: *15 Sep 2023* (Week 8)
  - “Concurrency” 15% DUE: *27 Oct 2023* (Week 12)

THE UNIVERSITY of ADELAIDE

Passing the course

- Five Class Quizzes (6%)
  - In-class: 11 Aug, 25 Aug, 08 Sep, 06 Oct, 20 Oct.
  - The final mark will be out of the best 4 quizzes.

- Five Tutorial Sessions (6%)
  - Every odd week – points for attendance.

You are expected to participate in all activities, attend lectures and submit your assignments on time.

THE UNIVERSITY of ADELAIDE

Late submission policy

- You should hand your coursework in on time.
- If you hand in your work late, your mark will be capped based on how many days late it is.
  - up to 1 day late — mark reduced to 75%, marks below 75% not affected.
  - up to 2 days late — mark reduced to 50%, marks below 50% not affected.
  - up to 3 days late — mark reduced to 25%, marks below 25% not affected.
  - More than 3 days late — mark is reduced to 0.
- If you handed in something on time, and it is worth more than something that you handed in late, you will get the higher mark.

THE UNIVERSITY of ADELAIDE

Repeating Students

- Students who repeat a course are expected to attempt all of the aspects of the course again. This includes making fresh attempts at all coursework assessment items.
- You may apply to the course coordinator to have your previous work counted but this is not usually granted.
- Make sure that you attend all of the lectures, do all of the work and study hard for the exam – you don't want to get stuck repeating the same course over and over.

THE UNIVERSITY of ADELAIDE

Assignment Extensions 1/2

A student may be eligible for assignment extensions based on medical, compassionate, extenuating circumstances.

A student will be ineligible if their Circumstances:

- Were avoidable, and the student had a reasonable opportunity to make alternative arrangements;
- Relate to balancing workloads from other units of study, disciplines or faculties
- Were personal commitments or events such as international travel, holidays or weddings;
- Relate to temporary minor ailments such as colds, minor respiratory infections, headaches or minor gastric upsets;
- Relate to stress or anxiety normally associated with examinations, required assessment tasks or any aspect of coursework;
- Are a result of misreading or misunderstanding the examination timetable.

THE UNIVERSITY of ADELAIDE

Assignment Extensions 2/2

For extensions, please contact the course coordinator

Cruz Izu <cruz.izu@adelaide.edu.au>

- If you think your situation is exceptional, contact your course coordinator ASAP, who will then consult the Head of School.
- Students who deliberately submit false or fraudulent documentation may be referred to the Student Misconduct Tribunal.
- You will normally only receive an extension equivalent to the number of days covered by your documentation. Don't expect to get an extra week because you lost a day.

THE UNIVERSITY of ADELAIDE

Exam Information (including additional / replacement exams)

University Examinations Site for information on Additional/Replacement exams:

- http://www.adelaide.edu.au/student/exams/

For the full policy on Modified Arrangements, see:

- https://www.adelaide.edu.au/policies/3303

THE UNIVERSITY of ADELAIDE

Academic Integrity

- **Useful link: https://www.adelaide.edu.au/student/success/academic-integrity-for-students**
- **Academic integrity values**
  - **Honesty** - being honest about where your ideas come from
  - **Respect** - giving credit when you use other people's ideas
  - **Responsibility** - taking ownership of your work
  - **Fairness** - treating other students and scholars fairly
  - **Trust** - doing the right thing, even when nobody is watching
  - **Courage** - standing up for what is right
- **The University takes academic integrity very seriously. For the most serious types of misconduct students can be suspended or completely excluded from the University.**

THE UNIVERSITY of ADELAIDE

Types of Academic Misconduct 1/2

- Plagiarism, where students present Work for assessment or publication that is not their own, without attribution or reference to the original source.
- Collusion, where students present Work as independent Work when it has in fact been produced in whole or in part with others (including persons external to the University) unless prior permission for joint or collaborative Work has been given by the Course coordinator, as specified in the Course Outline.
- Copying, where a student acts in such a way as to seek to gain unfair advantage or assist another student to do so.

THE UNIVERSITY of ADELAIDE

Types of Academic Misconduct 2/2

- Cheating in Examinations means engaging in dishonest practice or breaching the rules during or in relation to Examinations
- Contract Cheating, where a student submits completed or partially completed Work that a third party has completed for them, regardless of the relationship between the student and the third party or whether the third party is paid or unpaid. This includes the submission of work completed by an AI agent, without the explicit consent of your course coordinator.
- Misrepresentation, where a student presents untrue information with the intention of deceiving or misleading the assessor.
- Solicitation, where a student offers or gives money or any item or service to a University staff member or any other person to gain academic advantage for the student or another person.

THE UNIVERSITY of ADELAIDE

How to avoid plagiarism/collusion

- If you get stuck, seek help from the lecturer, tutor or prac demonstrator rather than copying from someone else.
- Starting your work early will help you to avoid getting stuck at the last minute.

When in doubt, ask your lecturer.

THE UNIVERSITY of ADELAIDE

Guidelines for using AI generated content

- **There are many content generating AI (text, code, solutions):**
  - use AI cautiously, understanding its bias and limitations
  - be wary that AI generated content can be dramatically wrong
  - submit content that you have created yourself
  - make sure that you cite where you have used content generated by AI

When in doubt, ask your lecturer.

THE UNIVERSITY of ADELAIDE

Assignment 1: "System calls"

- DUE: *11 Aug 2023* (Week 3)

08%

Objectives:

- Refresh/practice some basic C programming
- Familiarise yourself with the Unix system call mechanism
- Learn about processes and signals
- Get you thinking about parallel processes, as well as what the Unix shell really does.

Assignment 1: “System calls”

This assignment consists of two tasks:

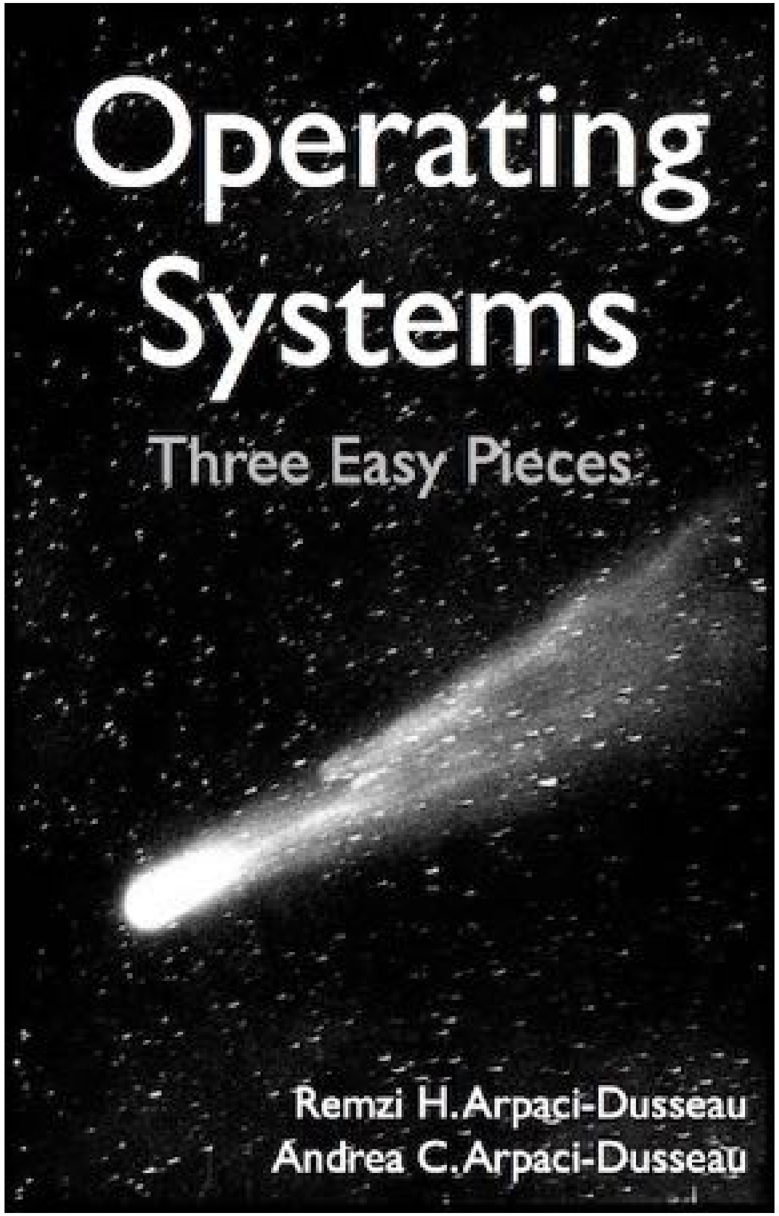
Task 1 (30%) Modify a program to handle HUP and INT signals.

Task 2 (70%) Improve a basic command line shell

You need to submit your code to Gradescope by Friday, 11^{th} August.

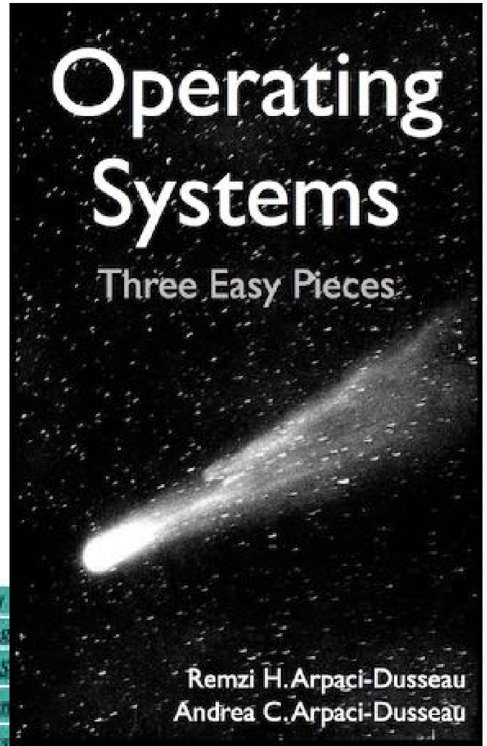
Course reading

Textbook



THE UNIVERSITY of ADELAIDE

Textbook



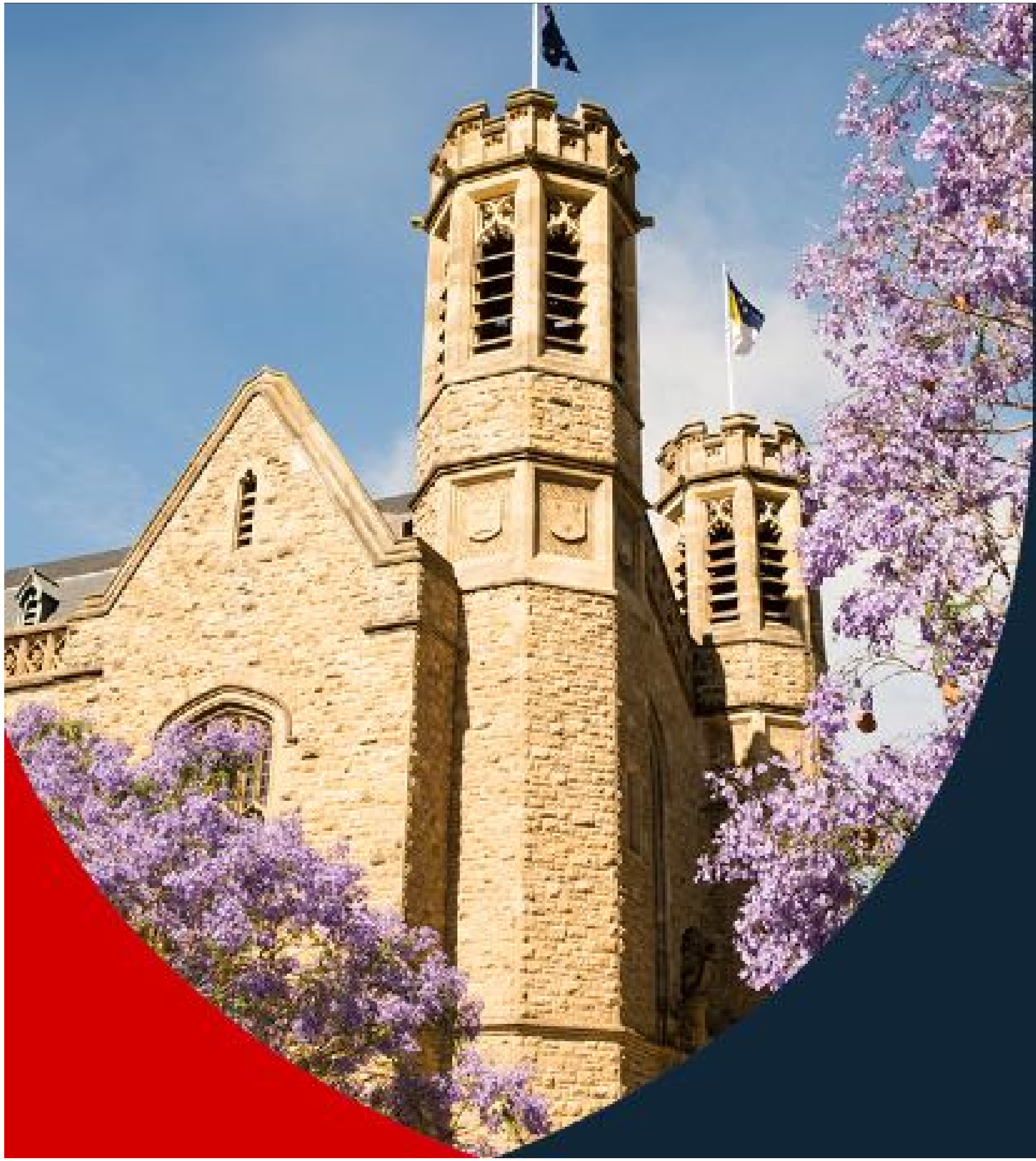
Operating Systems: Three Easy Pieces *by* Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau

https://pages.cs.wisc.edu/~remzi/OSTEP/

Intro	Virtualization		Concurrency	Persistence	Security
Preface	3 *Dialogue*	12 *Dialogue*	25 *Dialogue*	35 *Dialogue*	52 *Dialog*
TOC	4 Processes	13 Address Spaces code	26 Concurrency and Threads code	36 I/O Devices	53 *Intro S*
1 *Dialogue*	5 Process API code	14 Memory API	27 Thread API code	37 Hard Disk Drives	54 *Authen*
2 Introduction code	6 Direct Execution	15 Address Translation	28 Locks code	38 Redundant Disk Arrays (RAID)	55 *Access*
	7 CPU Scheduling	16 Segmentation	29 Locked Data Structures	39 Files and Directories	56 *Cryptography*
	8 Multi-level Feedback	17 Free Space Management	30 Condition Variables code	40 File System Implementation	57 *Distributed*
	9 Lottery Scheduling code	18 Introduction to Paging	31 Semaphores code	41 Fast File System (FFS)	
	10 Multi-CPU Scheduling	19 Translation Lookaside Buffers	32 Concurrency Bugs	42 FSCK and Journaling	**Appendices**
	11 *Summary*	20 Advanced Page Tables	33 Event-based Concurrency	43 Log-structured File System (LFS)	*Dialogue*
		21 Swapping: Mechanisms	34 *Summary*	44 Flash-based SSDs	Virtual Machines
		22 Swapping: Policies		45 Data Integrity and Protection	*Dialogue*
		23 Complete VM Systems		46 *Summary*	Monitors
		24 *Summary*		47 *Dialogue*	*Dialogue*
				48 Distributed Systems	Lab Tutorial
				49 Network File System (NFS)	Systems Labs
				50 Andrew File System (AFS)	xv6 Labs
				51 *Summary*	

THE UNIVERSITY of ADELAIDE

Questions
about course administration?

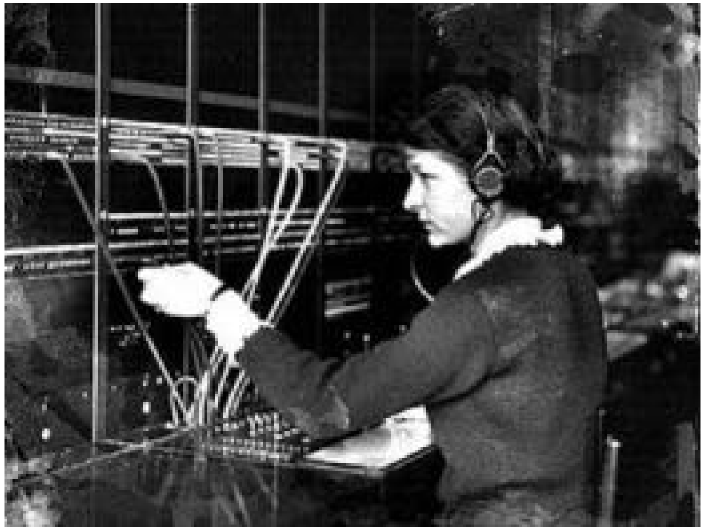


What is an Operating System (OS)?

make history.

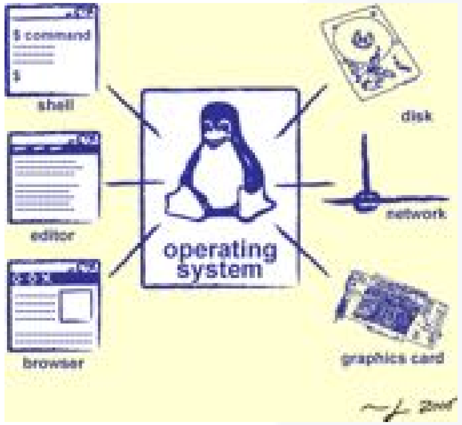
THE UNIVERSITY of ADELAIDE

Operating System



Switchboard Operator

Computer Operators



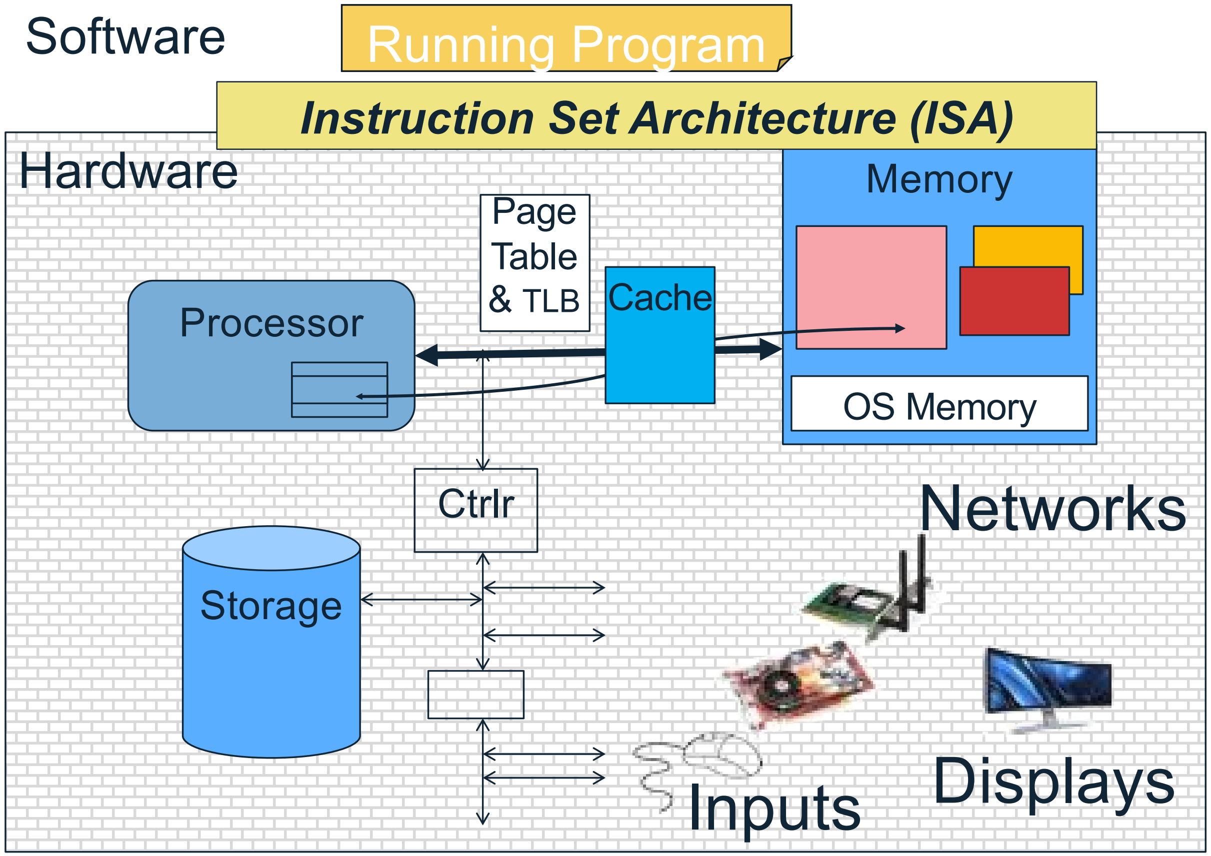
Operating System

“Questions”

- Does the programmer need to write a single program that performs many independent activities?
- Does every program have to be altered for every piece of hardware?
- Does a faulty program crash everything?
- Does every program have access to all hardware?

Hopefully, no!

Hardware/Software Interface



The OS *abstracts* these hardware details from the application

What is an Operating System (OS)?

- **Is software that converts hardware into a useful form for applications**
- **A program that acts as an intermediary between a user of a computer and the computer hardware**
- **It is a resource allocator and control program making efficient use of HW and managing the execution of user programs.**
- **OS goals are to**
  - make the computer system convenient to use
  - use the computer hardware in an efficient manner, and
  - manage resources, provide efficient and fair resource sharing

THE UNIVERSITY of ADELAIDE

OS *Abstracts* the Underlying Hardware

Processor → Thread
Memory → Address Space
Disks, SSDs, … → Files
Networks → Sockets
Machines → Processes

Application

Abstract Machine Interface

Operating System

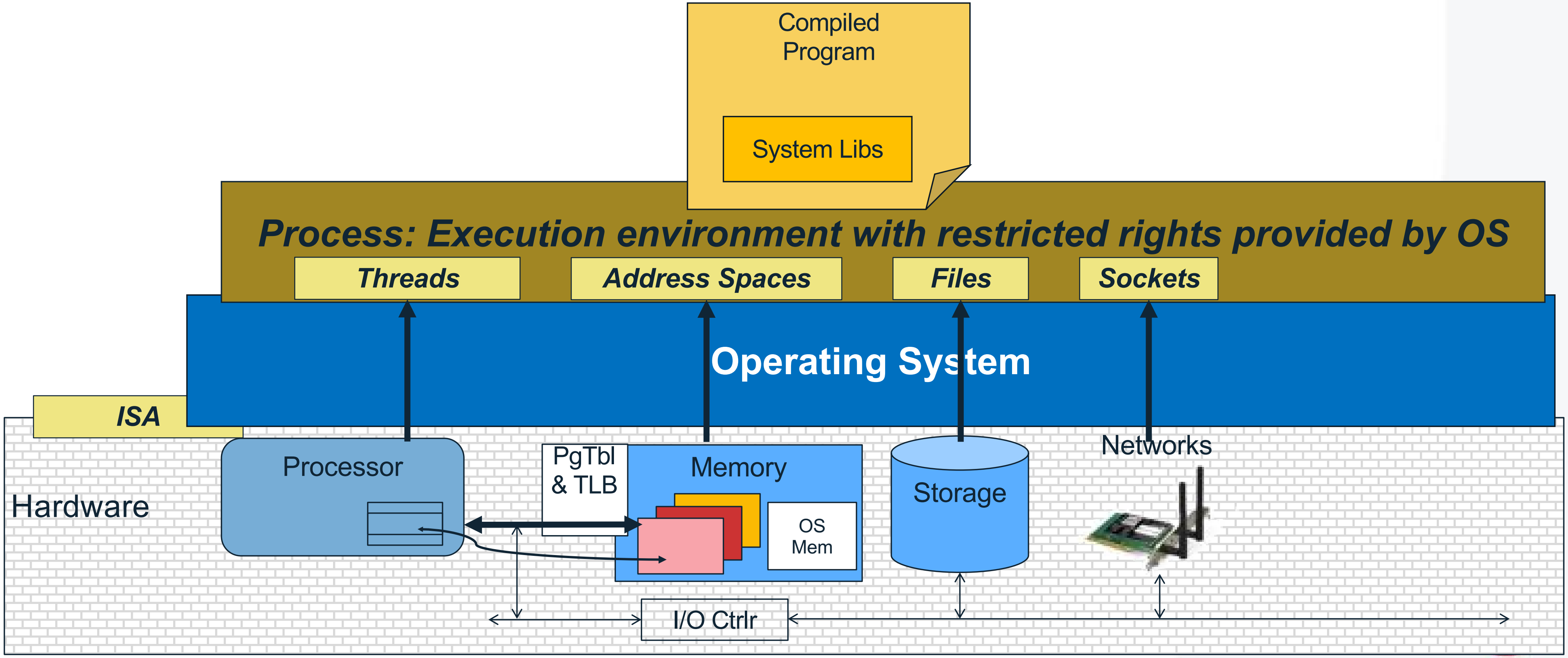
Physical Machine Interface

Hardware

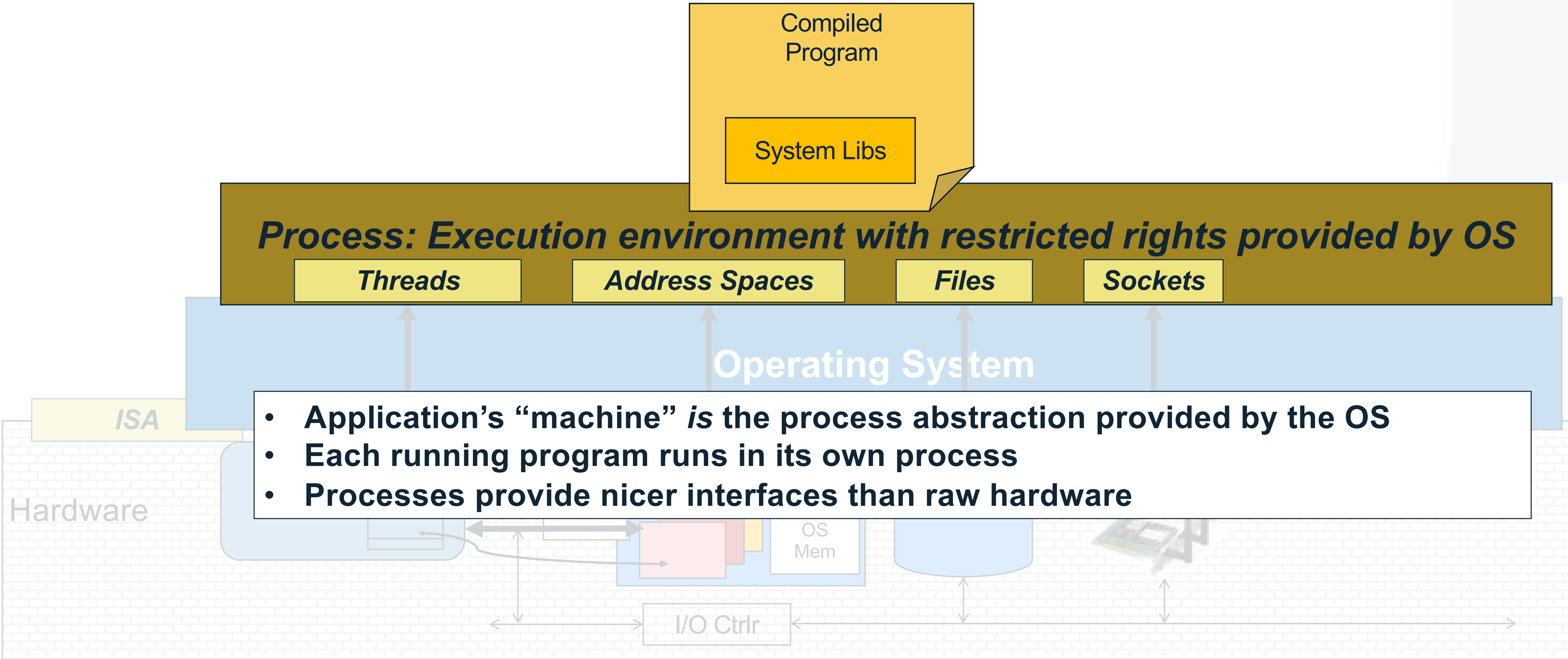
- OS as an *Illusionist*:
  - Remove software/hardware quirks (*fight complexity)*
  - Optimise for convenience, utilisation, reliability, … *(help the programmer)*
- For any OS area (e.g. file systems, virtual memory, networking, scheduling):
  - What hardware interface to handle? (physical reality)
  - What's a software interface to provide? (nicer abstraction)

THE UNIVERSITY of ADELAIDE

OS Basics: Virtualizing the Machine



Compiled Program's View of the World



- Application's "machine" *is* the process abstraction provided by the OS
- Each running program runs in its own process
- Processes provide nicer interfaces than raw hardware

What does OS provide?

Role #1: Abstraction - Provides standard library for resources

- **What is a resource?**
  - Anything valuable (e.g., CPU, memory, disk)
- **What abstraction does modern OS typically provide for each resource?**
  - CPU: process and/or thread
  - Memory: address space
  - Disk: files

THE UNIVERSITY of ADELAIDE

Advantages and challenges of Abstraction

- **Advantages of OS providing abstraction?**
  - Allow applications to reuse common facilities
  - Make different devices look the same
  - Provide higher-level or more useful functionality
- **Challenges**
  - What are the correct abstractions?
  - How much of hardware should be exposed?

THE UNIVERSITY of ADELAIDE

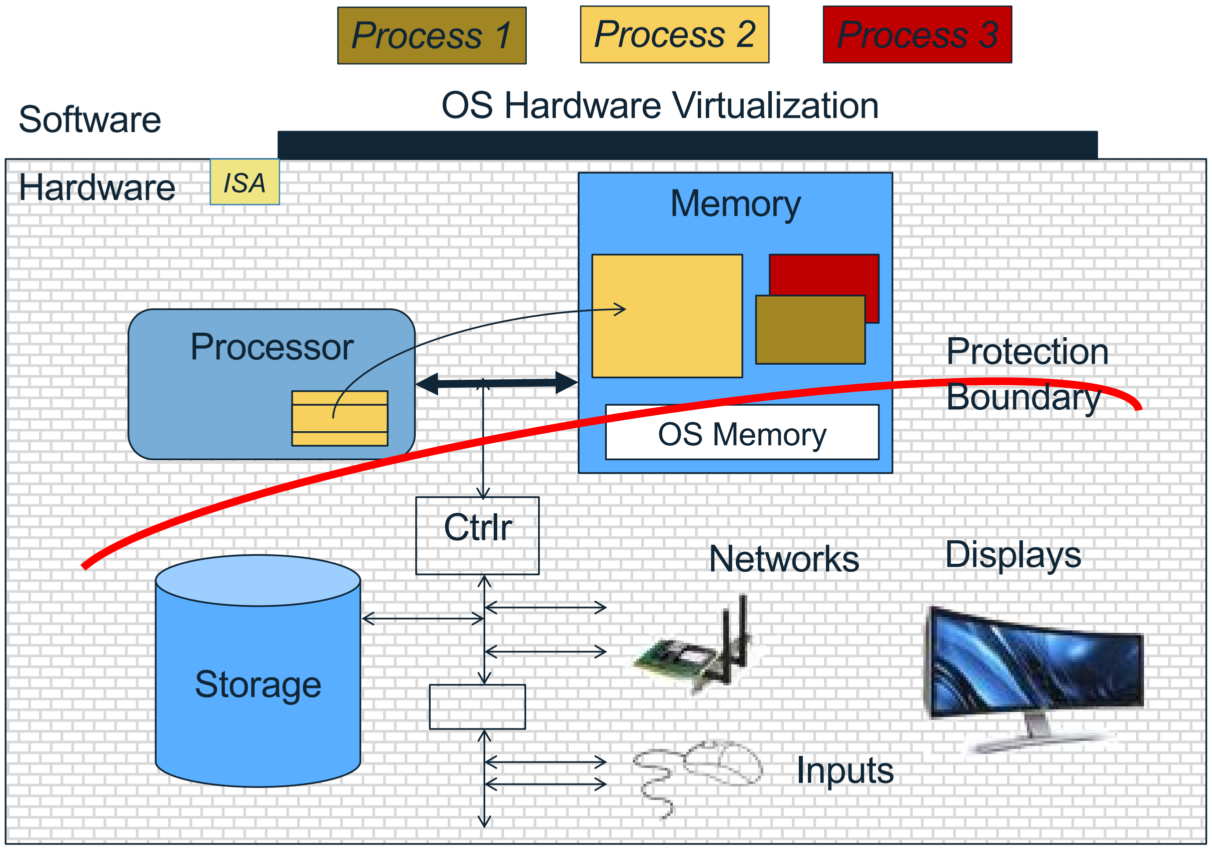
What does OS provide?

Role #2: Resource management – Share resources well

- **Advantages of OS providing resource management?**
  - Protect applications from one another
  - Provide efficient access to resources (cost, time, energy)
  - Provide fair access to resources
- **Challenges**
  - What are the correct mechanisms?
  - What are the correct policies?

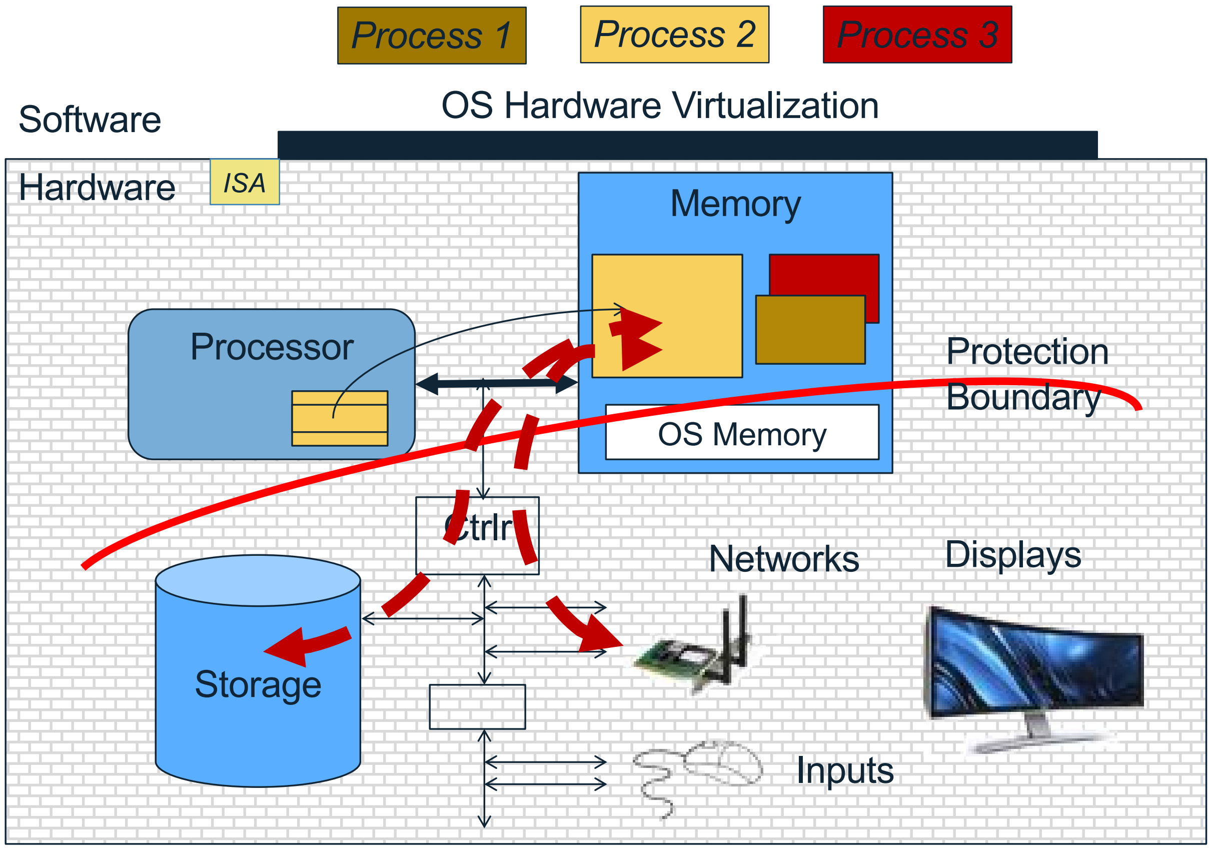
THE UNIVERSITY of ADELAIDE

OS Basics: Protection



- OS *isolates* processes from each other
- OS *isolates* itself from other processes
- … even though they are actually running on the same hardware!

OS Basics: I/O



- OS provides common services in the form of I/O

Why study Operating Systems?

Why study Operating Systems?

Some of you will actually design and build operating systems or components of them.

- Perhaps more now than ever

Many of you will create systems that utilise the core concepts in operating systems.

- Whether you build software or hardware
- The concepts and design patterns appear at many levels

All of you will build applications, etc., that utilise operating systems

- The better you understand their design and implementation, the better use you'll make of them.

THE UNIVERSITY of ADELAIDE

Operating Systems: Three Easy Pieces

Virtualisation

make history.

THE UNIVERSITY of ADELAIDE

Virtualization: The CPU

Abstraction:

- What is a process?

Mechanism

- Why is limited direct execution a good approach for virtualizing the CPU?
- What execution state must be saved for a process?
- What 3 modes could a process in?

THE UNIVERSITY of ADELAIDE

What is a process?

Process: An execution stream in the context of a process state

What is an execution stream?

- Stream of executing instructions
- Running piece of code
- “thread of control”

What is a process?

Process: An execution stream in the context of a process state

What is process state?

- Everything that the running code can affect or be affected by
- Registers
- General purpose, floating point, status, program counter, stack pointer
- Address space
- Heap, stack, and code
- Open files

Processes vs. Programs

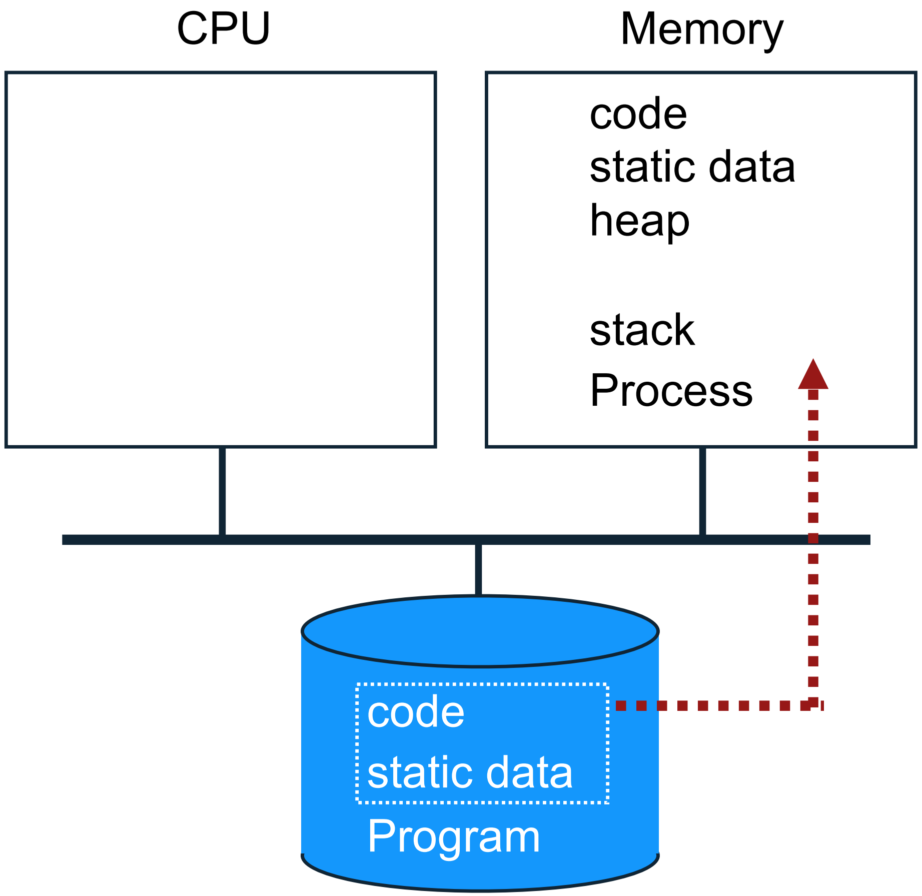
A process is different than a program

- Program: Static code and static data
- Process: Dynamic instance of code and data

Can have multiple process instances of the same program

- Can have multiple processes of the same program
- Example: many users can run “sort” at the same time

Process creation



Processes vs. Threads

A process is different than a thread

Thread: “Lightweight process” (LWP)

- An execution stream that shares an address space
- Multiple threads within a single process

Example:

- Two **processes** examining same memory address 0xffe84264 see **different** values (I.e., different contents)
- Two **threads** examining memory address 0xffe84264 see **same** value (I.e., same contents)

Virtualizing the CPU

Goal

- Give each process impression it alone is actively using CPU
- Resources can be shared in time and space

Assume single uniprocessor

- Time-sharing (multi-processors: advanced issue)

Memory?

- Space-sharing (later)

Disk?

- Space-sharing (later)

How to provide good CPU performance?

Direct execution

- Allow user process to run directly on hardware
- OS creates process and transfers control to starting point (i.e., main())

Problems with direct execution of process?

- Restricted (access file, read/write other process data)
- Run forever (slow, buggy, or malicious)
- Slow (like I/O)

Solution

- **Limited direct execution** – OS and hardware maintain some control

THE UNIVERSITY of ADELAIDE

Problem 1: Restricted OPS

How can we ensure user process cannot harm others?

Solution: privilege levels supported by hardware (bit of status)

- User processes run in user mode (restricted mode)
- OS runs in kernel mode (not restricted)
  - Instructions for interacting with devices
  - Could have many privilege levels (advanced topic)

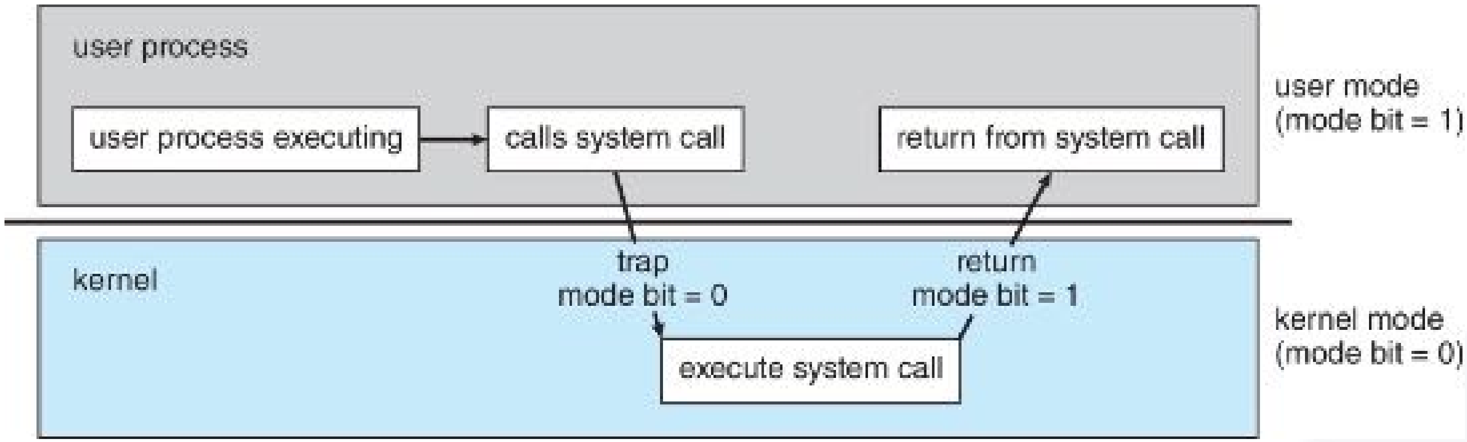
How can process access device?

- System calls (function call implemented by OS)
- Change privilege level through system call (trap)

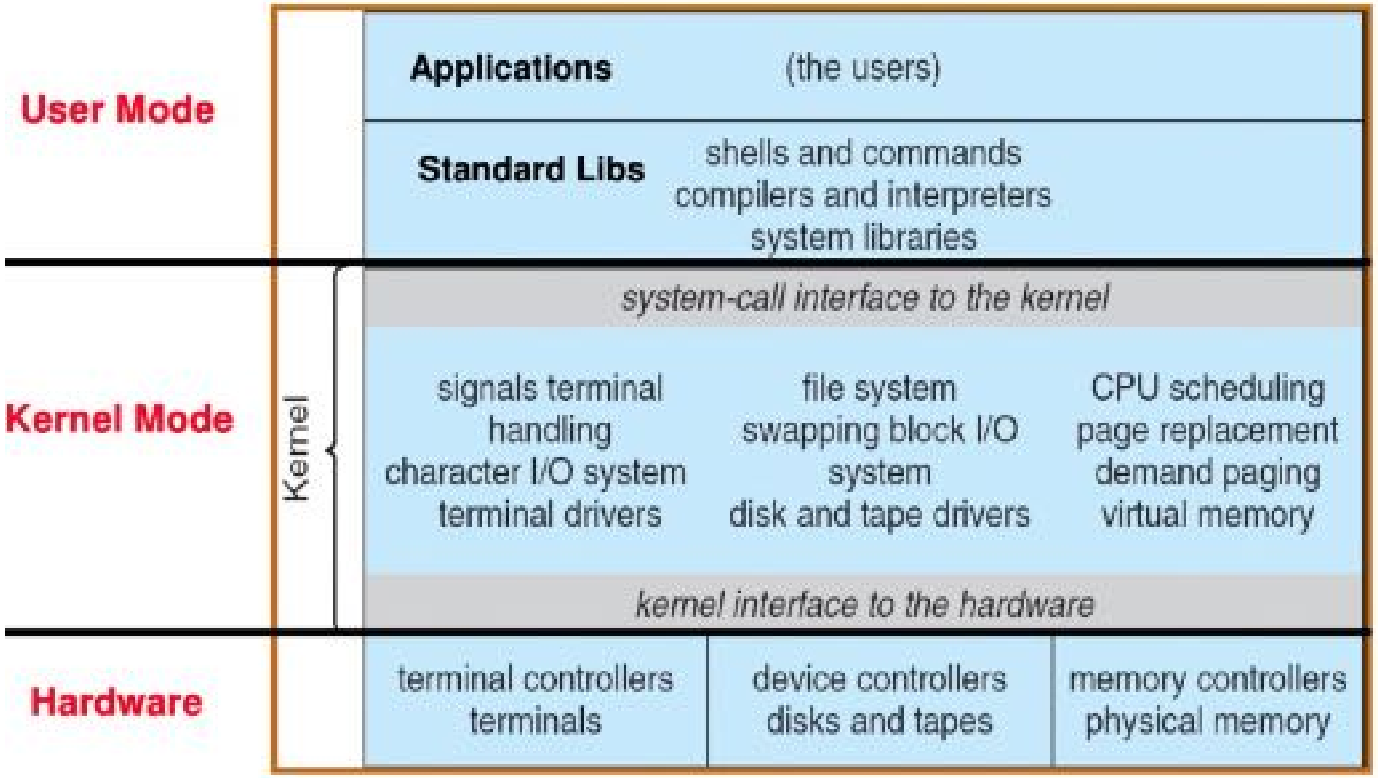
THE UNIVERSITY of ADELAIDE

Transition from user to kernel mode

- When the system starts executing it is in kernel mode
- When control is given to a user program the mode-bit changes to “user mode”.
- When a user issues a system call it results in an interrupt, which trap to the operating system. At that time, the mode–bit is set to “kernel mode”.

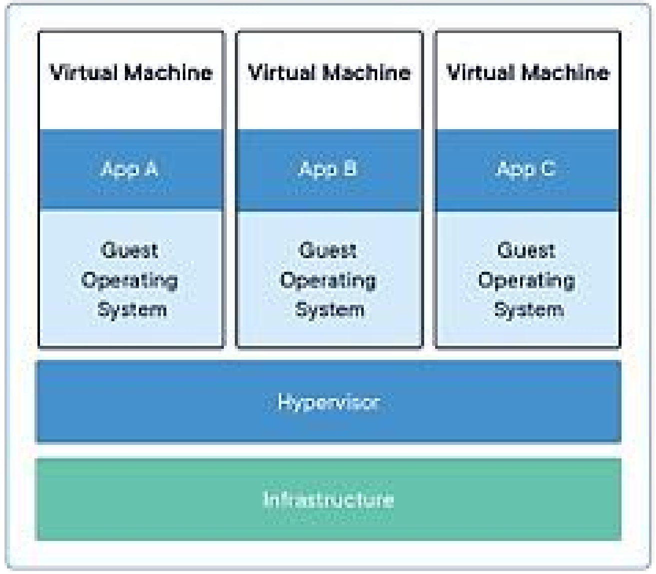


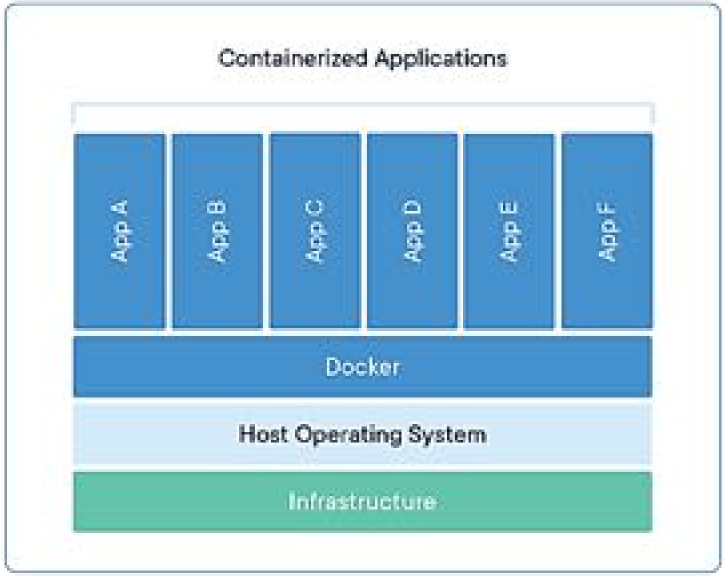
UNIX System Structure



THE UNIVERSITY of ADELAIDE

Virtualization: Execution Environments for Systems





Additional layers of protection and isolation can help further manage complexity

System Call – How a process access devices?

Types of Kernel Mode Transfer

Syscall

- Process requests a system service, e.g., exit
- Like a function call, but “outside” the process
- Does not have the address of the system function to call
- Marshall the syscall id and args in registers and exec syscall

Interrupt

- External asynchronous event triggers context switch
- eg. Timer, I/O device
- Independent of user process

Trap or Exception

- Internal synchronous event in process triggers context switch
- e.g., Protection violation (segmentation fault), …

System call

P wants to call read()

System call

P can only see its own memory because of user mode
(other areas, including kernel, are hidden)

System call

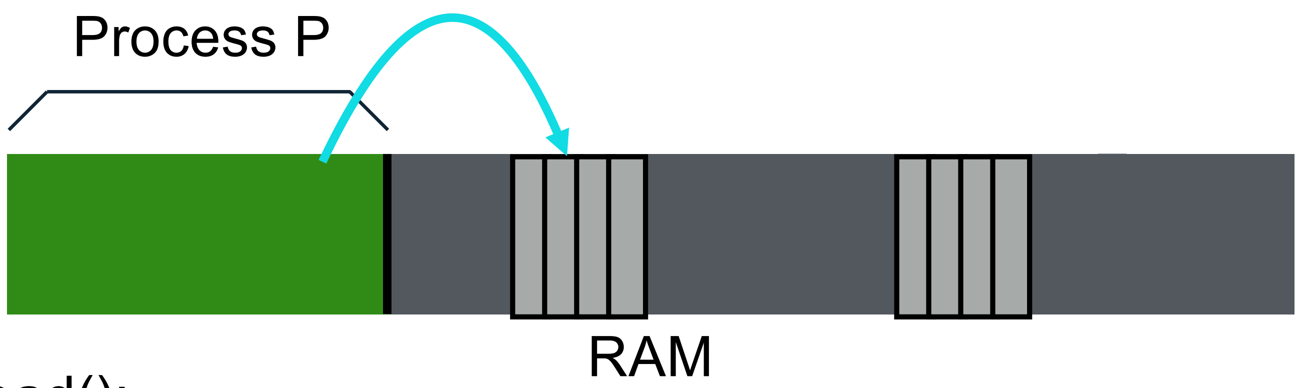
P wants to call read() but no way to call it directly

System call

read():

```
movl $6, %eax;   int $64
```

System call



read():

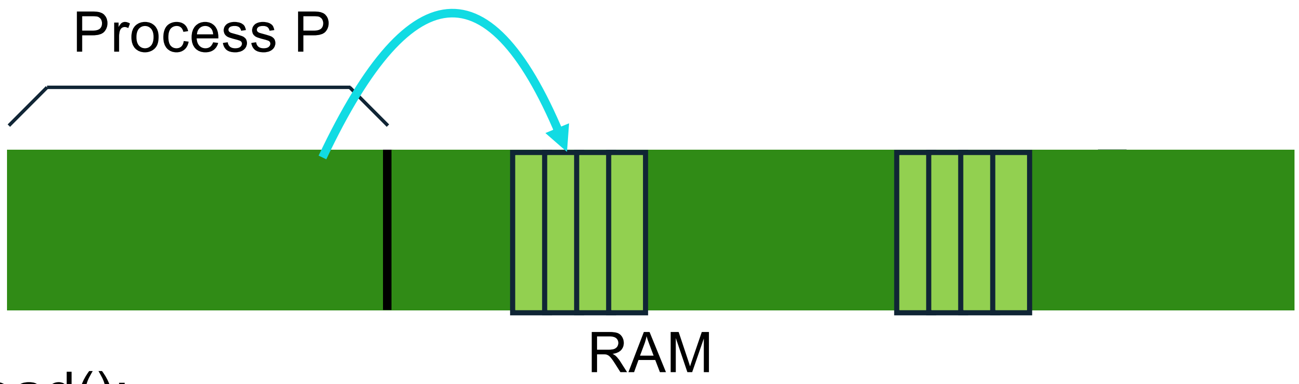
```
movl $6, %eax;   int $64
```

syscall-table index

trap-table index

System call

Kernel mode: we can do anything!

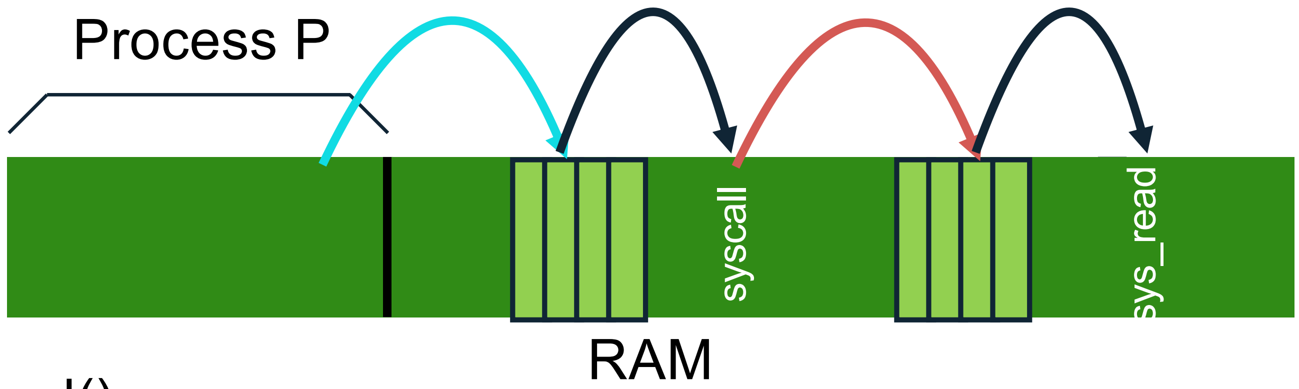


read():

```
movl $6, %eax;    int $64
```

syscall-table index   trap-table index

System call



read():

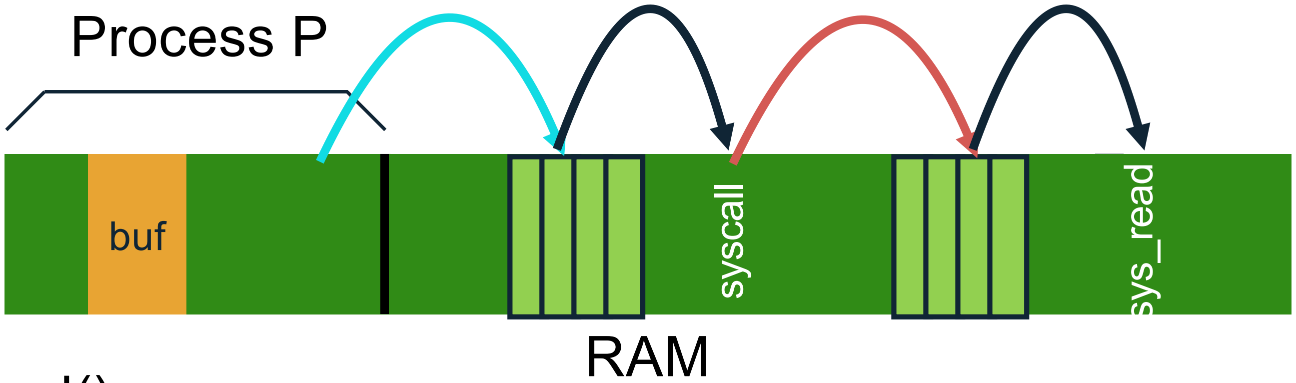
```
movl $6, %eax;   int $64
```

syscall-table index

trap-table index

Follow entries to correct system call code

System call



read():

```
movl $6, %eax;   int $64
```

syscall-table index   trap-table index

Kernel can access user memory to fill in user buffer return-from-trap at end to return to Process P

What to limit?

User processes are not allowed to perform:

- General memory access
- Disk I/O
- Special x86 instructions like lidt

THE UNIVERSITY of ADELAIDE

Problem 2: How to take CPU away?

OS requirements for multitasking: Separation of policy and mechanism

- Policy: Decision-maker to optimise some workload performance metric
  - Which process when?
  - Process Scheduler: Later
- Mechanism: Low-level code that implements the decision
  - How?
  - Process Dispatcher: Next in today’s lecture

THE UNIVERSITY of ADELAIDE

Dispatch Mechanism

OS runs dispatch loop

```
while (1) {
        run process A for some time-slice
        stop process A and save its context
        load context of another process B
    }
```

Context-switch

Question 1: How does dispatcher gain control?

Question 2: What execution context must be saved and restored?

THE UNIVERSITY of ADELAIDE

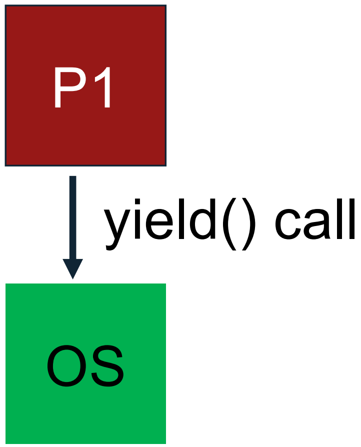
Q1: How does Dispatcher get control?

Option 1: Cooperative Multi-tasking

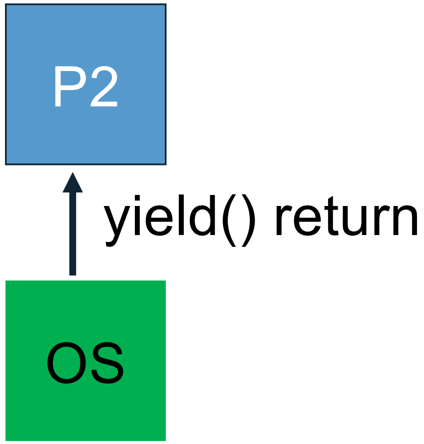
- Trust process to relinquish CPU to OS through traps
  - Examples: System call, page fault (access page not in main memory), or error (illegal instruction or divide by zero)
  - Provide special yield() system call

THE UNIVERSITY of ADELAIDE

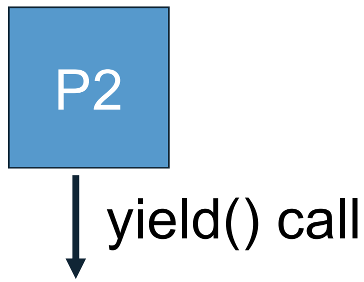
Cooperative Approach



Cooperative Approach



Cooperative Approach



Q1: How does dispatcher run?

Problem with cooperative approach?

Disadvantages: Processes can misbehave or have bugs

- By avoiding all traps and performing no I/O, can take over entire machine
- Only solution: Reboot!

Not performed in modern operating systems

THE UNIVERSITY of ADELAIDE

Q1: How does dispatcher run?

Option 2: True Multi-tasking

- Guarantee OS can obtain control periodically
- Enter OS by enabling periodic alarm clock
  - Hardware generates timer interrupt (CPU or separate chip)
  - Example: Every 10ms
- User must not be able to mask timer interrupt
- Dispatcher counts interrupts between context switches
  - Example: Waiting 20 timer ticks gives 200 ms time slice
  - Common time slices range from 10 ms to 200 ms

Q2: What context must be saved?

Dispatcher must track the context of the process when not running

What information is stored in PCB (Process Control Block)?

- PID
- Process state (I.e., running, ready, or blocked)
- Execution state (all registers, PC, stack ptr)
- Scheduling priority
- Accounting information (parent and child processes)
- Credentials (which resources can be accessed, owner)
- Pointers to other allocated resources (e.g., open files)

THE UNIVERSITY of ADELAIDE

Operating System	Hardware	Program
		Process A …

Operating System	Hardware	Program
		Process A …
	timer interrupt save regs(A) to k-stack(A) move to kernel mode jump to trap handler	

Operating System	Hardware	Program
		Process A …
	timer interrupt save regs(A) to k-stack(A) move to kernel mode jump to trap handler	
Handle the trap Call switch() routine save regs(A) to proc-struct(A) restore regs(B) from proc-struct(B) switch to k-stack(B) return-from-trap (into B)		

Operating System	Hardware	Program
		Process A …
	timer interrupt save regs(A) to k-stack(A) move to kernel mode jump to trap handler	
Handle the trap Call switch() routine save regs(A) to proc-struct(A) restore regs(B) from proc-struct(B) switch to k-stack(B) return-from-trap (into B)		
	restore regs(B) from k-stack(B) move to user mode jump to B's IP	

Operating System	Hardware	Program
		Process A …
	timer interrupt save regs(A) to k-stack(A) move to kernel mode jump to trap handler	
Handle the trap Call switch() routine save regs(A) to proc-struct(A) restore regs(B) from proc-struct(B) switch to k-stack(B) return-from-trap (into B)		
	restore regs(B) from k-stack(B) move to user mode jump to B's IP	
		Process B …

Problem 3: Slow OPS such as I/O?

When running process performs op that does not use CPU, OS switches to process that needs CPU (policy issues)

OS must track mode of each process:

- Running: on the CPU (only one on a uniprocessor)
- Ready: waiting for the CPU
- Blocked: Asleep: Waiting for I/O or synchronization to complete

Problem 3: Slow OPS such as I/O?

OS must track every process in system

- Each process identified by unique Process ID (PID)

OS maintains queues of all processes

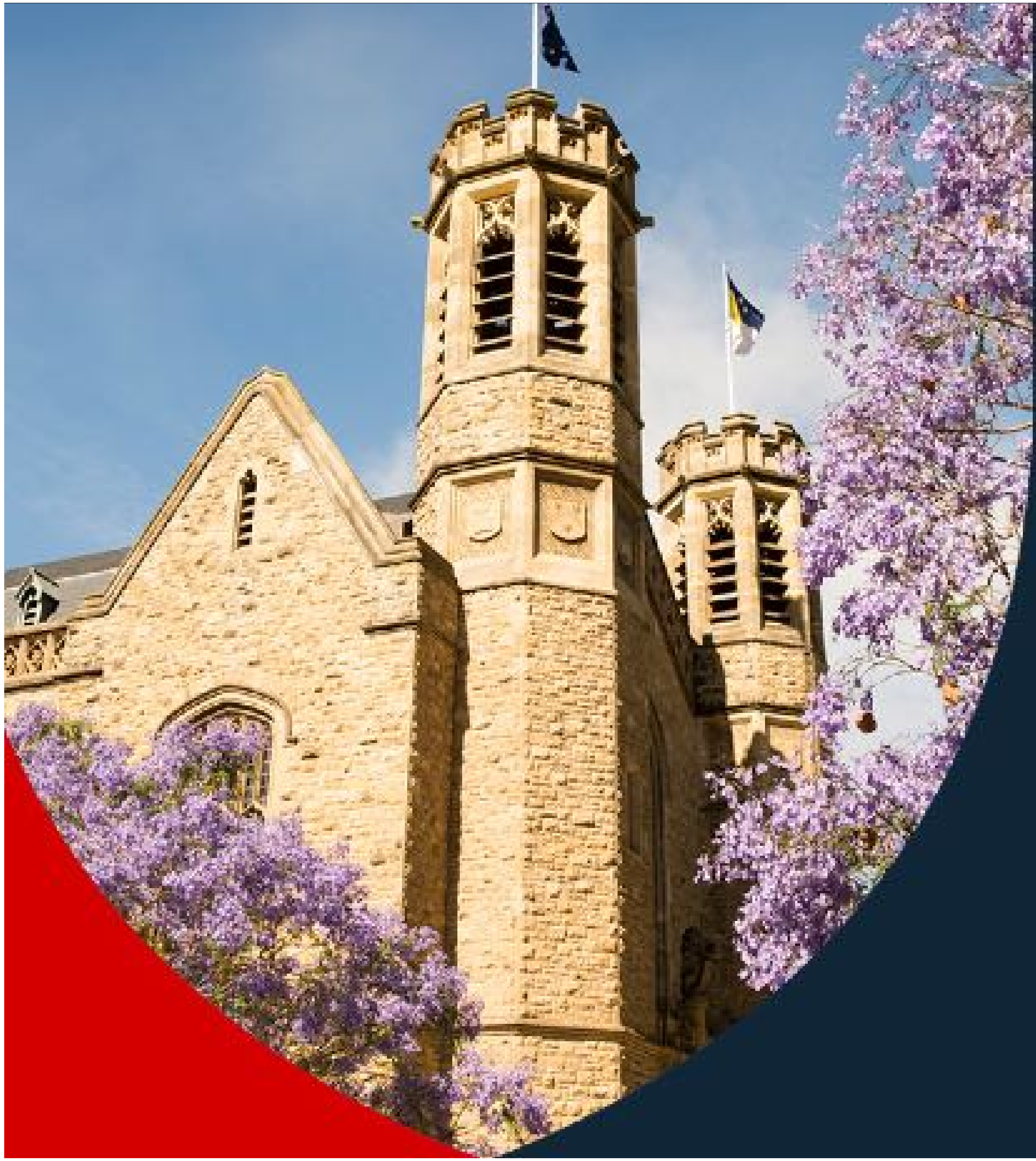
- Ready queue: Contains all ready processes
- Event queue: One logical queue per event
- e.g., disk I/O and locks
- Contains all processes waiting for that event to complete

Next Topic: Policy for determining which ready process to run

Summary

Virtualization

- Context switching gives each process impression it has its own CPU
- Direct execution makes processes fast
- Limited execution at key points to ensure OS retains control
- Hardware provides a lot of OS support
  - user vs kernel mode
  - timer interrupts
  - automatic register saving



COMP SCI 3004 Operating Systems

Week 2 – Scheduling

make history.

THE UNIVERSITY of ADELAIDE

CPU Virtualization: Scheduling

Questions answered in this lecture:

- What are different scheduling policies, such as:
  FCFS, SJF, STCF, RR and MLFQ?
- What type of workload performs well with each scheduler?

THE UNIVERSITY of ADELAIDE

CPU Virtualization: Two Components

Dispatcher

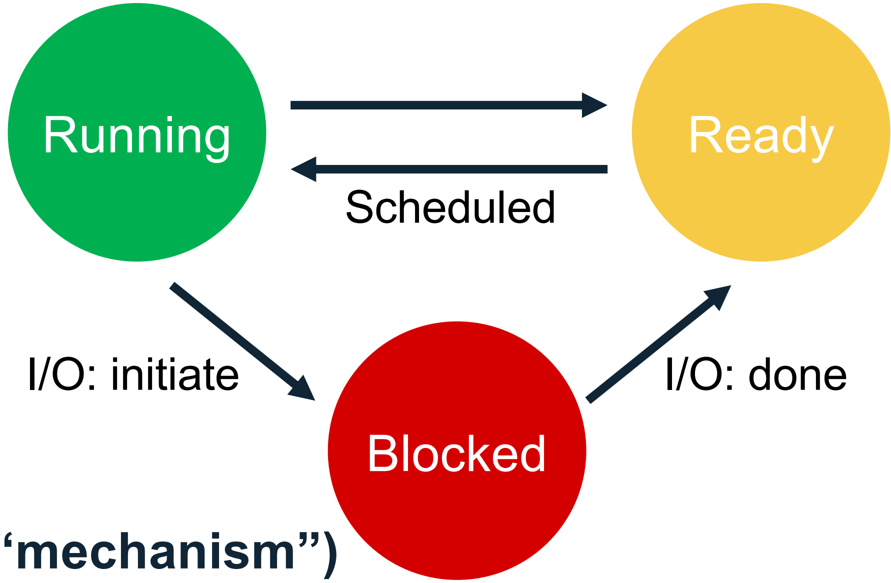
- **Low-level mechanism - performs context-switch**
  - Switch from user mode to kernel mode
  - Save execution state (registers) of old process in PCB
  - Insert PCB in ready queue
  - Load state of next process from PCB to registers
  - Switch from kernel to user mode
  - Jump to instruction in new user process

Scheduler

- Policy to determine which process gets CPU when

THE UNIVERSITY of ADELAIDE

State Transitions



How to transition? ("mechanism")

When to transition? ("policy")

Vocabulary

- **Workload:** set of job descriptions (arrival time, run_time)
- **Job:** View as current CPU burst of a process
  - process alternates between CPU and I/O
  - process moves between ready and blocked queues
- **Scheduler:** logic that decides which ready job to run
- **Metric:** measurement of scheduling quality

Scheduling Performance Metrics

Minimize turnaround time – Completion_time – arrival_time

Minimize response time – Initial_schedule_time – arrival_time

Minimize waiting time – Do not want to spend much time in Ready queue

Maximize throughput – Want many jobs to complete per unit of time

Maximize resource utilization – Keep expensive devices busy

Minimize overhead – Reduce number of context switches

Maximize fairness – All jobs get same amount of CPU over some time interval

Workload Assumptions

1. Each job runs for the same amount of time
2. All jobs arrive at the same time
3. All jobs only use the CPU (no I/O)
4. Run-time of each job is known

Scheduling Basics

Workloads:

- arrival_time
- run_time

Schedulers:

- FIFO
- SJF
- STCF
- RR

Metrics:

- turnaround_time
- response_time

THE UNIVERSITY of ADELAIDE

Example: workload, scheduler, metric

JOB	arrival_time (s)	run_time (s)
A	~0	10
B	~0	10
C	~0	10

- **FIFO: First In, First Out**
  - also called FCFS (first come first served)
  - run jobs in arrival_time order
- **What is our turnaround?:**
  - completion_time - arrival_time

FIFO: Event Trace

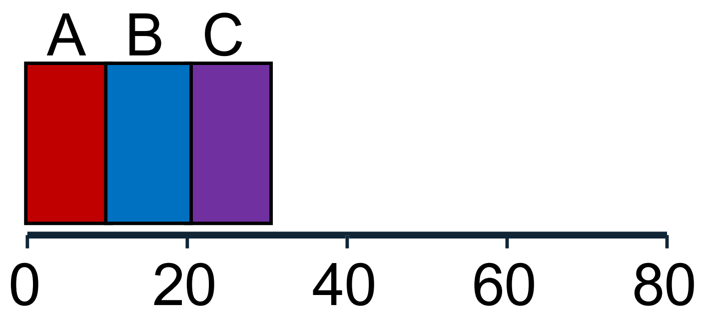
JOB	arrival_time (s)	run_time (s)
A	~0	10
B	~0	10
C	~0	10

Time	Event
0	A arrives
0	B arrives
0	C arrives
0	run A
10	complete A
10	run B
20	complete B
20	run C
30	complete C

THE UNIVERSITY of ADELAIDE

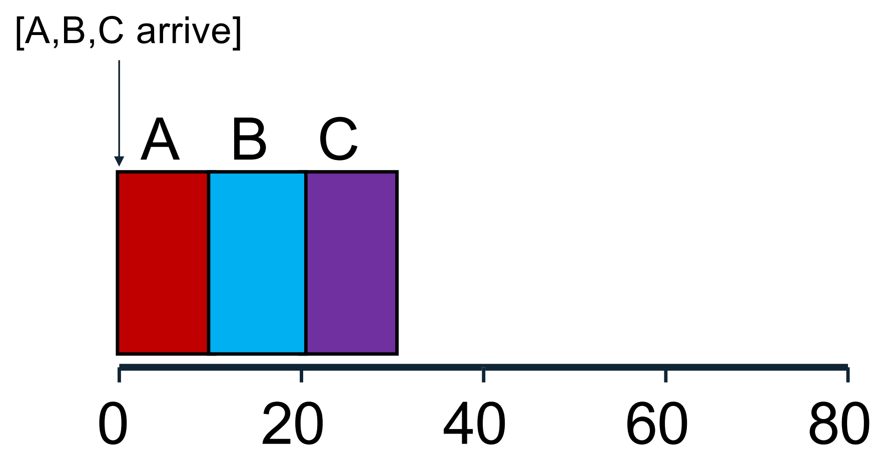
FIFO: (Identical JOBS)

JOB	arrival_time (s)	run_time (s)
A	~0	10
B	~0	10
C	~0	10



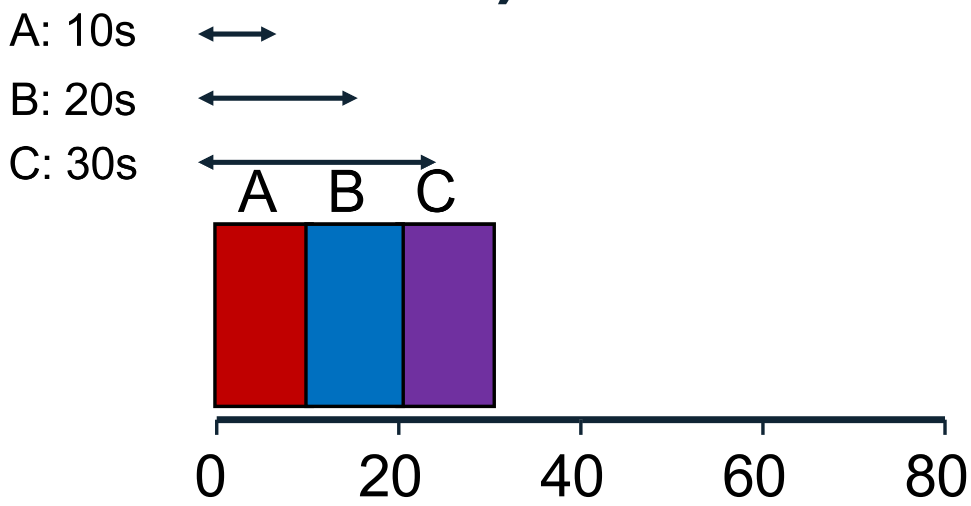
- **Gantt chart**
  - Illustrates how jobs are scheduled over time on a CPU

FIFO: (Identical JOBS)



- **What is the average turnaround time?**
  - Def: turnaround_time = completion_time - arrival_time

FIFO: (Identical JOBS)



- **What is the average turnaround time?**
  - Def: turnaround_time = completion_time - arrival_time
  - (10 + 20 + 30) / 3 = **20s**

THE UNIVERSITY of ADELAIDE

Scheduling Basics

Workloads:

- arrival_time
- run_time

Schedulers:

- FIFO
- SJF
- STCF
- RR

Metrics:

- turnaround_time
- response_time

Workload Assumptions

1. ~~Each job runs for the same amount of time~~
2. All jobs arrive at the same time
3. All jobs only use the CPU (no I/O)
4. Run-time of each job is known

THE UNIVERSITY of ADELAIDE

Any Problematic Workloads for FIFO?

Workload: ?

Scheduler: FIFO

Metric: turnaround is high

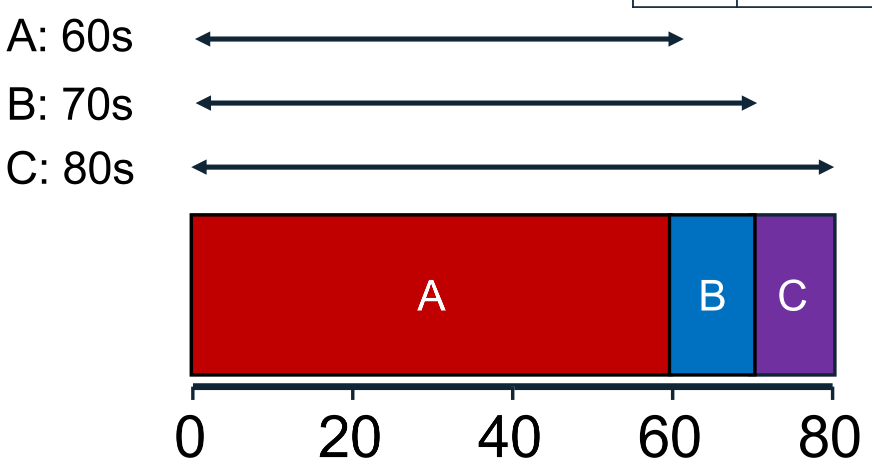
Example: Big First Job

JOB	arrival_time (s)	run_time (s)
A	~0	60
B	~0	10
C	~0	10

- **Draw Gantt chart for this workload and policy…**
- **What is the average turnaround time?**

Example: Big First Job

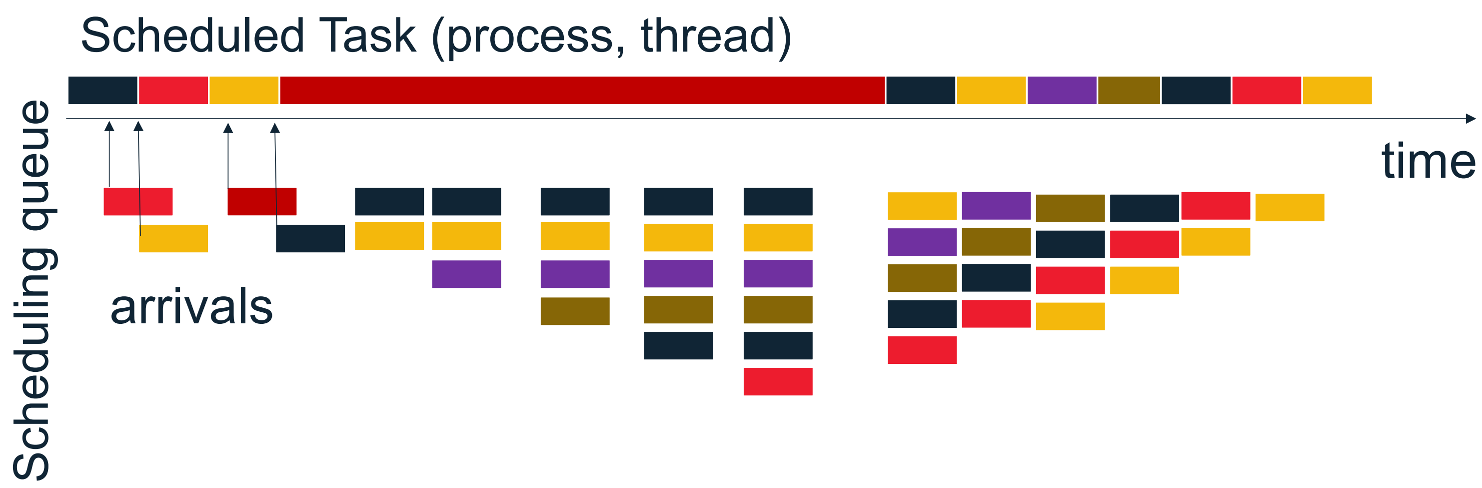
JOB	arrival_time (s)	run_time (s)
A	~0	60
B	~0	10
C	~0	10



Average turnaround time: **70s**

Convoy Effect

Convoy effect



With FCFS non-preemptive scheduling, convoys of small tasks tend to build up when a large one is running.

Passing the Tractor

- **Problem with Previous Scheduler:**
  - FIFO: Turnaround time can suffer when short jobs must wait for long jobs
- **New scheduler:**
  - SJF (Shortest Job First)
  - Choose job with smallest run_time

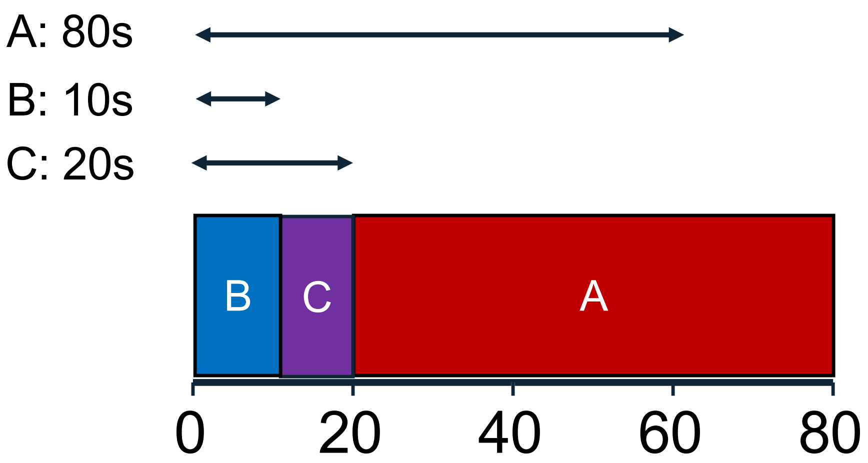
THE UNIVERSITY of ADELAIDE

Shortest Job First

JOB	arrival_time (s)	run_time (s)
A	~0	60
B	~0	10
C	~0	10

What is the average turnaround time with SJF?

SJF Turnaround Time



What is the average turnaround time with SJF?

- (80 + 10 + 20) / 3 = **~36.7s**
- **Average turnaround with FIFO: 70s**

SJF Turnaround Time

- **For minimizing average turnaround time (with no preemption):**
  - SJF is provably optimal
- **Moving shorter job before longer job improves turnaround time of short job more than it harms turnaround time of long job**

THE UNIVERSITY of ADELAIDE

Scheduling Basics

Workloads:

- arrival_time
- run_time

Schedulers:

- FIFO
- SJF
- STCF
- RR

Metrics:

- turnaround_time
- response_time

THE UNIVERSITY of ADELAIDE

Workload Assumptions

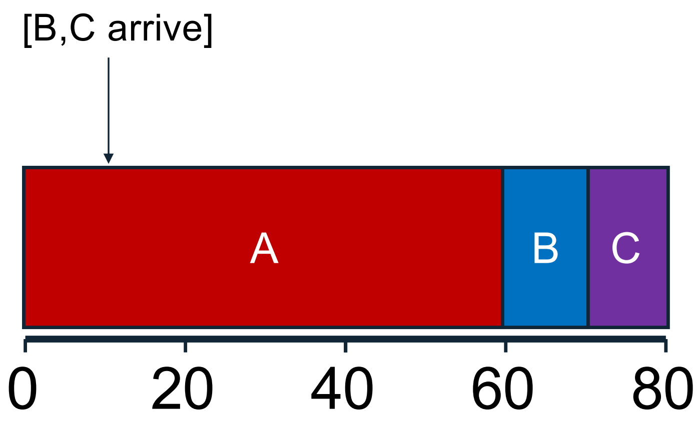
1. ~~Each job runs for the same amount of time~~
2. ~~All jobs arrive at the same time~~
3. All jobs only use the CPU (no I/O)
4. Run-time of each job is known

Shortest Job First (Arrival Time)

JOB	arrival_time (s)	run_time (s)
A	~0	60
B	~10	10
C	~10	10

What is the average turnaround time with SJF?

Stuck Behind a Tractor Again



JOB	arrival_time (s)	run_time (s)
A	~0	60
B	~10	10
C	~10	10

What is the average turnaround time with SJF?

- (60 + (70 – 10) + (80 – 10)) / 3 = **63.3s**

THE UNIVERSITY of ADELAIDE

Conclusion

We have covered today:

- Introduction
- Abstractions of resources.
  - Virtualising the CPU
  - The abstraction level of a process.
- Kernel and user mode
  - System calls via interrupts and traps
- OS mechanisms on how to take the CPU Away