CSE 4733/6733 - Operating System 1

Stephen A. Torri, Ph.D.

Mississippi State University

CSE 4733/6733 - Operating System 1

Stephen A. Torri, Ph.D.

Mississippi State University

Introduction

Introduction

Introduction

Sharing resources amongst processes (Ref: M. Vutukuru)

- **Importance:** Understanding OS process management is key to computer science.
- **Topics Covered:**
  - Process Execution
  - System Calls
  - Context Switching
  - Scheduling Policies
- **Prerequisites:** Assumes basic knowledge of computer architecture, programming, and operating systems.
- **Objective:** To explore mechanisms and policies of process management in modern operating systems.

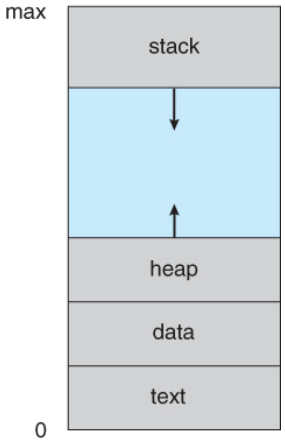
Introduction
Sharing resources amongst processes (Ref: M. Vutukuru)
WHAT WILL I LEARN?
- Importance: Understanding OS process management is key to computer science.
- Topics Covered:
  - Process Execution
  - System Calls
  - Context Switching
  - Scheduling Policies
- Prerequisites: Assumes basic knowledge of computer architecture, programming, and operating systems.
- Objective: To explore mechanisms and policies of process management in modern operating systems.

1. The operating system enables efficient resource allocation, multitasking, security, and more.

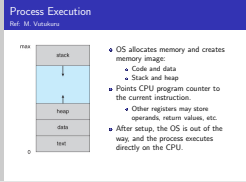
Mechanism of process Execution

Process Execution

Ref: M. Vutukuru



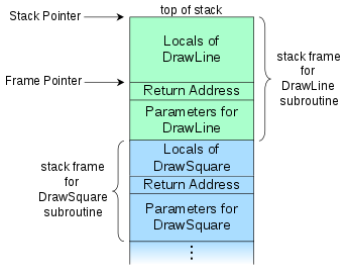
- OS allocates memory and creates memory image:
  - Code and data
  - Stack and heap
- Points CPU program counter to the current instruction.
  - Other registers may store operands, return values, etc.
- After setup, the OS is out of the way, and the process executes directly on the CPU.



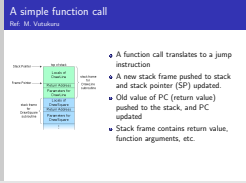
1. Introduction to Process Execution:
   - Explain that the Operating System (OS) is responsible for managing the execution of processes.
   - Highlight the importance of understanding how the OS allocates memory and controls the CPU.
2. Memory Allocation and Creation:
   - Describe how the OS allocates memory and creates a memory image for a process.
   - Explain the memory components, including code, data, stack, and heap.
3. Audience Engagement: How does the OS ensure a process has the resources it needs?

A simple function call

Ref: M. Vutukuru



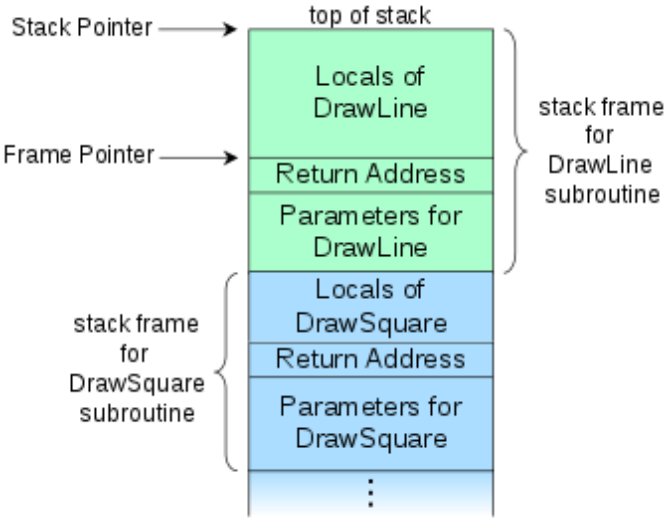
- A function call translates to a jump instruction
- A new stack frame pushed to stack and stack pointer (SP) updated.
- Old value of PC (return value) pushed to the stack, and PC updated
- Stack frame contains return value, function arguments, etc.

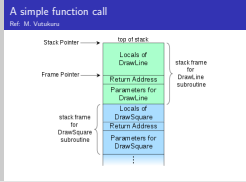


1. Introduction to Function Calls:
   - Explain what a function call is and why it's an essential concept in programming and computer science.
2. Explaining the Visual Aid:
   - Walk the audience through the images, explaining each part of the call stack layout and how a function call translates into a jump instruction.
   - Explain the role of the stack frame, return value, function arguments, etc., in a function call.
   - Discuss how the stack pointer is updated and what happens to the program counter (PC).
3. Audience Engagement: Can anyone explain what happens to the stack pointer during a function call?"

A simple function call

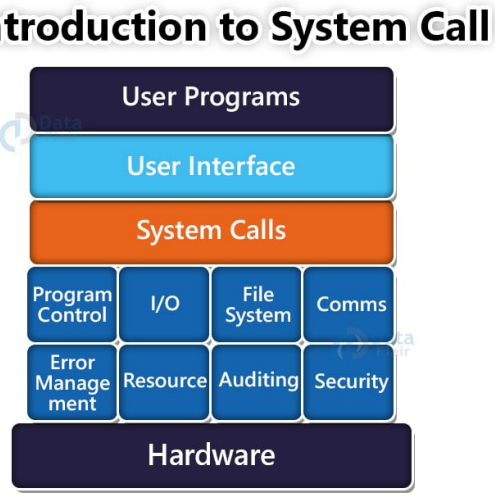
Ref: M. Vutukuru





How is a system call different?

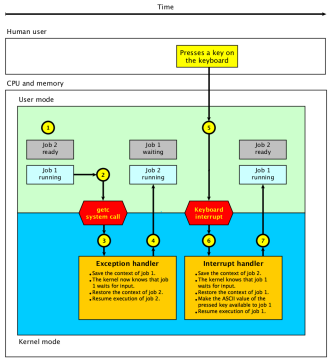
Ref: M. Vutukuru



How is a system call different?
Ref: M. Vutukuru
Introduction to System Call
User Programs
User Interface
System Calls
Program Control
I/O
File System
Comms
Error Manage ment
Resource
Auditing
Security
Hardware

How is a system call different?

Ref: M. Vutukuru



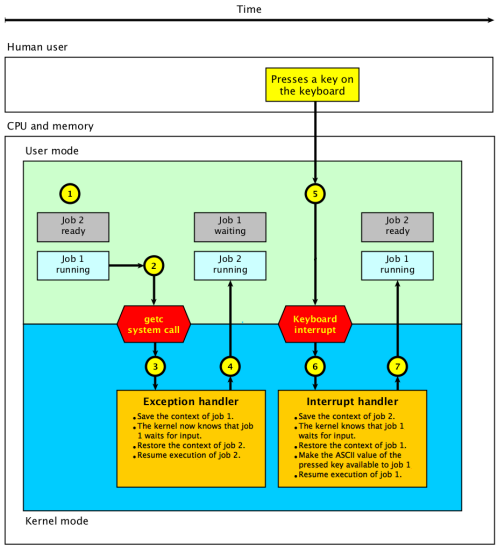
- CPU hardware has multiple privilege levels – One to run user code: user mode
- To keep the kernel space protection, a separate stack is used when in kernel mode.
- Kernel does not trust user-provided addresses to jump into its space.

How is a system call different?
Ref: M. Vutukuru
CPU hardware has multiple privilege levels – One to run user code: user mode
To keep the kernel space protection, a separate stack is used when in kernel mode.
Kernel does not trust user-provided addresses to jump into its space.

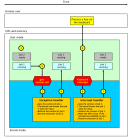
1. Privilege evels.
   - One to run OS code like system calls: kernel mode. Some instructions execute only in kernel mode.
   - Most applications will work in user mode.
2. - Kernel sets up Interrupt Descriptor Table (IDT) at boot time.
   - IDT contains addresses of kernel functions to run for system calls and other events.

How is a system call different?

Ref: M. Vutukuru



How is a system call different?
Ref: M. Vutukuru



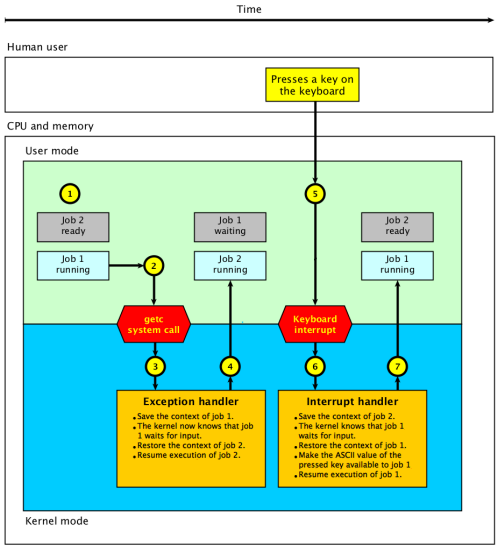
Step 1: Job 2 is ready to run, and Job 1 is running.

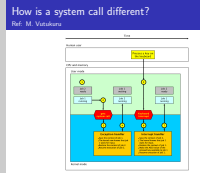
Step 2: Job 1 uses the "getc" system call to read a character from the keyboard.

Step 3: The system call uses a system call exception to enter the kernel.

How is a system call different?

Ref: M. Vutukuru

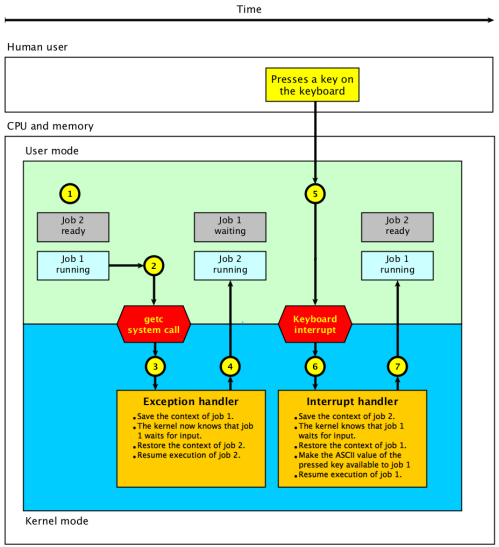


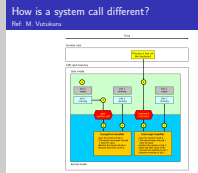


Step 4: The kernel resumes execution of Job 2. Job 1 changes its state from ready to "running."

How is a system call different?

Ref: M. Vutukuru





Step 5: The human presses a key on the keyboard.Step 6: The key press causes a keyboard interrupt. The kernel handles the interrupt.

- The kernel saves the context of Job 2 and changes its state from running to ready.
- The kernel knows that job 1 waits for the "getc" system call to complete.
- The kernel restores the context of Job 1 and changes its state from waiting to running.
- The ASCII value of the pressed key is made available to Job 1.

The kernel resumes the execution of Job 1.

Mechanism of system call: trap instruction

Ref: M. Vutukuru

- When a system call must be made, a special trap instruction is run (usually hidden from the user by the "libc" library)
- Trap instruction execution
  - Move CPU to a higher privilege level
  - Switch to kernel stack – Save context (old PC, registers) on the kernel stack
  - Look up address in IDT and jump to trap handler function in OS code

Mechanism of system call: trap instruction
Ref: M. Vutukuru
- When a system call must be made, a special trap instruction is run (usually hidden from the user by the "libc" library)
- Trap instruction execution
  - Move CPU to a higher privilege level
  - Switch to kernel stack – Save context (old PC, registers) on the kernel stack
  - Look up address in IDT and jump to trap handler function in OS code

Definition: Context Switch

Definition

A **Context Switch** is when the OS saves the contents of the Central Process Unit (CPU) registers to the stack of a process and then setups the CPU registers for another process.

Definition: Context Switch
Definition
A **Context Switch** is when the OS saves the contents of the Central Process Unit (CPU) registers to the stack of a process and then setups the CPU registers for another process.

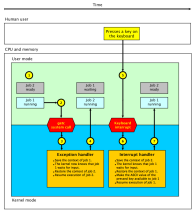
1. The CPU registers that are saved are:
   - General purpose registers
   - Stack pointer (ESP/RSP): Address location where data can be written on the stack.
   - Segment Registers: Holds addresses to memory blocks for use by a process.
   - Base Pointer (EBP/RBP): Address to the starting address of the current stack frame.
   - EFLAGS: Program status and control registers
   - Instruction Pointer (EIP)

Scheduling Policies

Scheduling Policies

What is a scheduling policy?

Ref: M. Vutukuru

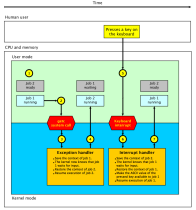


- On a context switch, which process to run next from a set of ready processes?
- OS scheduler schedules the CPU requests (bursts) of processes
  - CPU burst = the CPU time used by a process in a continuous stretch
  - If a process comes back after I/O wait, it counts as a fresh CPU burst.

What is a scheduling policy?
Ref: M. Vutukuru
On a context switch, which process to run next from a set of ready processes?
OS scheduler schedules the CPU requests (bursts) of processes
CPU burst = the CPU time used by a process in a continuous stretch
If a process comes back after I/O wait, it counts as a fresh CPU burst.

What are we trying to optimize?

Ref: M. Vutukuru



- Maximize (utilization = fraction of time CPU is used)
- Minimize average (turnaround time = time from process arrival to completion)
- Minimize average (response time = time from process arrival to first scheduling)
- Fairness: all processes must be treated equally
- Minimize overhead: process runs long enough to amortize the cost of context switch (1 microsecond)

What are we trying to optimize?
Ref: M. Vutukuru
Maximize (utilization – fraction of time CPU is used)
Minimize average (turnaround time – time from process arrival to completion)
Minimize average (response time – time from process arrival to first scheduling)
Fairness: all processes must be treated equally
Minimize overhead: process runs long enough to amortize the cost of context switch (1 microsecond)

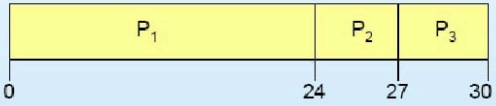
First come, first served

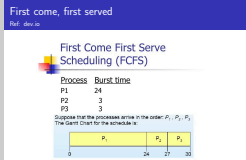
Ref: dev.io

First Come First Serve Scheduling (FCFS)

Process	Burst time
P1	24
P2	3
P3	3

Suppose that the processes arrive in the order: P_1 , P_2 , P_3
The Gantt Chart for the schedule is:





1. If the processes arrive in the order P1, P2, and P3 are served in FCFS order, we get the result shown above. The waiting time is 0 milliseconds for process P1, 24 milliseconds for process P2, and 27 milliseconds for process P3. Thus, the average waiting time is $(0+ 24 + 27)/3 = 17$ milliseconds.
2. Advantage: The code for FCFS scheduling is simple to write and understand.

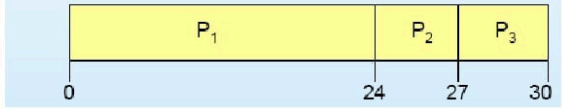
First come, first served

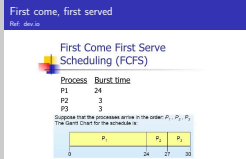
Ref: dev.io

First Come First Serve Scheduling (FCFS)

Process	Burst time
P1	24
P2	3
P3	3

Suppose that the processes arrive in the order: P_1 , P_2 , P_3
The Gantt Chart for the schedule is:

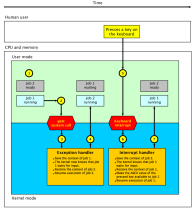




1. Disadvantages:
   - The process will run to the completion, and there are high chances of starvation, as it is non-preemptive.
   - There is a convoy effect as all the other processes wait for the one big process to get off the CPU. This effect would result in lower CPU and device utilization than possible if the shorter processes were allowed to go first.
   - It is poor in performance since the average waiting time is higher compared to other scheduling algorithms.

Shortest Job First

Ref: M. Vutukuru

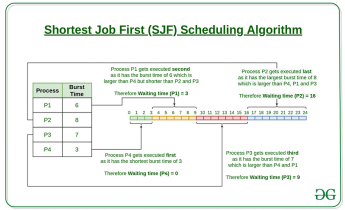


The shortest job first (SJF) or shortest job next is a scheduling policy that selects the waiting process with the shortest execution time to execute next. SJN, also known as Shortest Job Next (SJN), can be preemptive or non-preemptive.

Shortest Job First
Ref: M. Vutukuru
The shortest job first (SJF) or shortest job next is a scheduling policy that selects the waiting process with the shortest execution time to execute next. SJN, also known as Shortest Job Next (SJN), can be preemptive or non-preemptive.

Shortest Job First

Ref: Program for Shortest Job First (or SJF) CPU Scheduling - Geek for Geeks



Characteristics of SJF Scheduling:

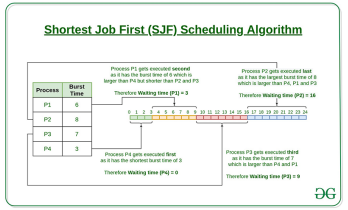
- Shortest Job first has the advantage of having a minimum average waiting time among all scheduling algorithms.
- It is a Greedy Algorithm.
- Can cause starvation.
- SJF can have bursty behavior.
- SJF can be used in specialized environments where accurate running time estimates are available.

Shortest Job First
Ref: Program for Shortest Job First (or SJF) CPU Scheduling - Geek for Geeks
Characteristics of SJF Scheduling:
- Shortest Job first has the advantage of having a minimum average waiting time among all scheduling algorithms.
- It is a Greedy Algorithm.
- Can cause starvation.
- SJF can have bursty behavior.
- SJF can be used in specialized environments where accurate running time estimates are available.

1. It may cause starvation if shorter processes keep coming. This problem can be solved using the concept of aging.
2. It is practically infeasible as Operating System may not know burst times and, therefore, may not sort them. While it is impossible to predict execution time, several methods can be used to estimate the execution time for a job, such as a weighted average of previous execution times.

Shortest Job First

Ref: Program for Shortest Job First (or SJF) CPU Scheduling - Geek for Geeks



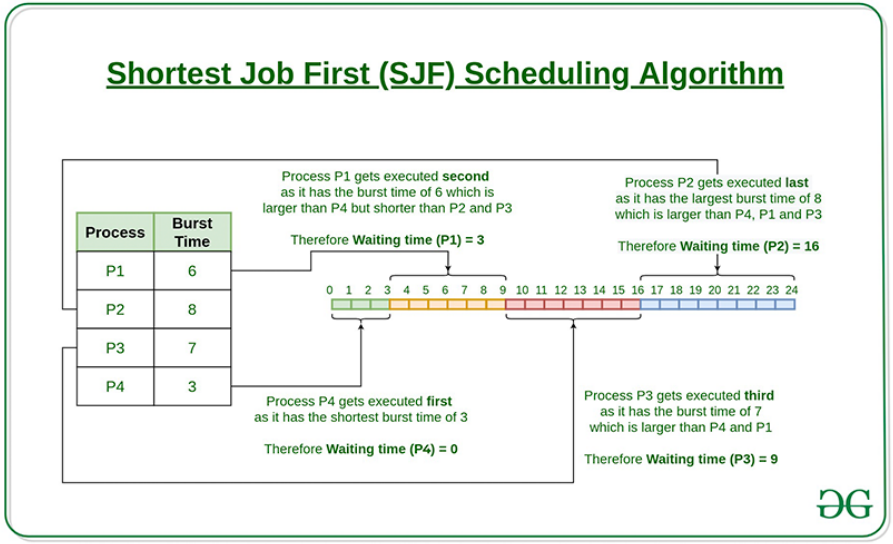
Algorithm:

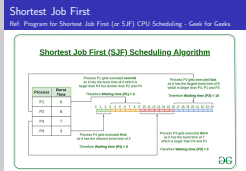
1. Sort all the processes according to the arrival time.
2. Select that process with minimum arrival and burst times.
3. After completion of the process, make a pool of processes that arrives after the completion of the previous process and select that process among the pool with minimum burst time.

Shortest Job First
Ref: Program for Shortest Job First (or SJF) CPU Scheduling - Geek for Geeks
Algorithm:
1. Sort all the processes according to the arrival time.
2. Select that process with minimum arrival and burst times.
3. After completion of the process, make a pool of processes that arrives after the completion of the previous process and select that process among the pool with minimum burst time.

Shortest Job First

Ref: Program for Shortest Job First (or SJF) CPU Scheduling - Geek for Geeks





Round Robin

Ref: Geek for Geeks

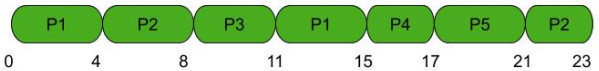
TIME SLICE = 4

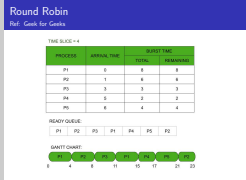
PROCESS	ARRIVAL TIME	BURST TIME	
		TOTAL	REMAINING
P1	0	8	8
P2	1	6	6
P3	3	3	3
P4	5	2	2
P5	6	4	4

READY QUEUE:

P1	P2	P3	P1	P4	P5	P2

GANTT CHART:





1. The Round Robin scheduling algorithm is preemptive. It uses the concept of time slice or time quantum. The process at the beginning of the ready queue gets a chance to be executed first but only for a span of one-time quantum. As new and more processes get added to the ready queue, the ongoing process gets preempted and added to the end of the ready queue. The next process gets the chance, again for a span of one-time quantum. This algorithm is designed for time-sharing systems.