1 Virtualizing CPU

- Turns a single CPU into a seemingly infinite number of CPUS, and allows many programs to seemingly run at once
- To implement CPU virtualization, the OS needs low-level machinery called **mechanism** and high level intelligence called **policies**
- Steps

  1. Involve OS to setup hardware hardwre to limit what the process can do without OS assistance (**Limited Direct Execution**)

     This is done so by

     1. Setting up trap handler
     2. Starting an interrupt timer (so process won't last forever)

  2. Involve OS to intervene at key points to perform previleged operations or switch out operations when they have monopolized the CPU too long

2 Limited Direct Execution

- Idea: Just run the program you want to run on the CPU, but first make sure to set up the hardware so as to limit what process can do without OS assistance
- baby proofs the CPU by

  1. Setting up trap handlers
  2. Starts an interrupt timer
  3. Run processes in a restricted mode

  Example

  Baby proofing a room:

  – Locking cabinets containing dangerous stuff and covering electrical sockets.
  – When room is readied, let your baby roam free in knowledge that all the dangerous aspect of the room is restricted

3 Trap Handlers

- Is instruction that tells the hardware what to run when certain exceptions occur

Example

What code to run when

1. Hard disk interrupt occurs
2. Keyboard interrupt occrs
3. Program makes a system call?

4 Timer Interrupt

- Is a hardware mechanism that ensures the user program does not run forever
- Is emitted at regular intervals by a timer chip [6]

5 Response Time

- **Formula** $T_{response} = T_{firstrun} - T_{arrival}$
- measures the interactive performance between users and the system

6 Turnaround Time

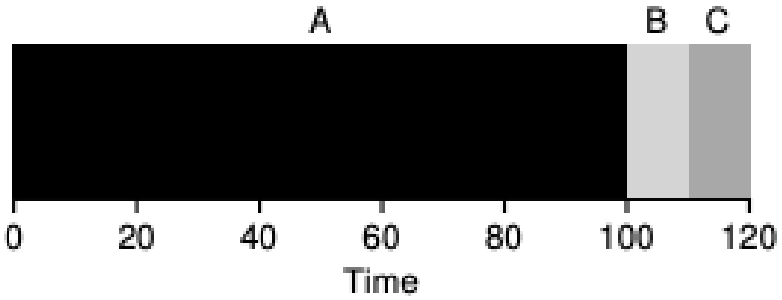
- **Formula** $T_{turnaround} = T_{completion} - T_{arrival}$
- measures the amount of time taken to complete a process

7 Starvation

- Is the problem that occurs when high priority processes keep executing and low priority processes get blocked for indefinite time [1]

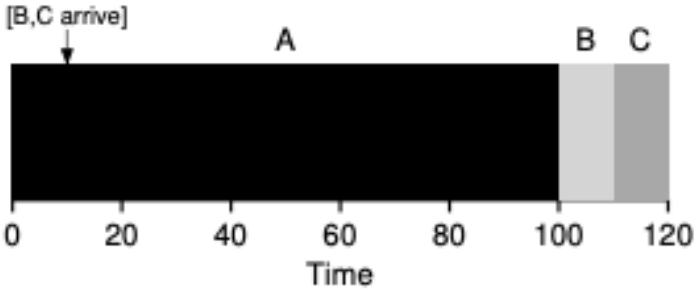
8 Convoy Effect

- Is the problem where number of relatively-short potential consumers of a resource get queued behind a heavy weight consumer

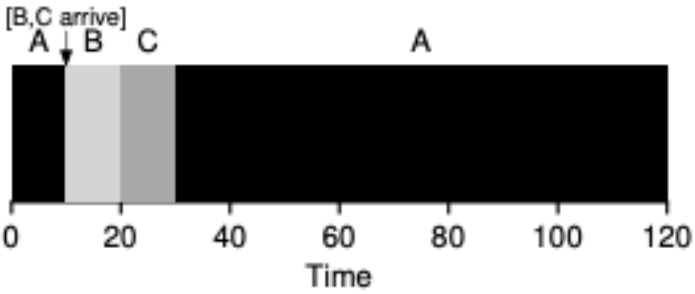


9 Scheduling policies

- Are algorithms for allocating CPU resources to concurrent tasks deployed on (i.e., allocated to) a processor (i.e., computing resource) or a shared pool of processors [5]
- Are sometimes called **Discipline**
- Covers the following algorithms in textbook
  - **First In First Out**
    - Is the most basic scheduling algorithm
    - Is vulnerable to **convoy effect**
    - No **starvation** as long as every process eventually completes
  - **Shortest Job First**
    - Improves average **turnaround time** given processes of uneven length
    - Is a general scheduling principle useful in situation where turnaround time per process matters
    - Is vulnerable to **convoy effect**



    - Is vulnerable to **starvation**
      - When only short-term jobs come in while a long term job is in queue
  - **Shortest Time-to-completion First**
    - Addresses **convoy effect** in **Shortest Job First**
    - Determines which of the remaining+new jobs has least time left, and schedule accordingly at any time

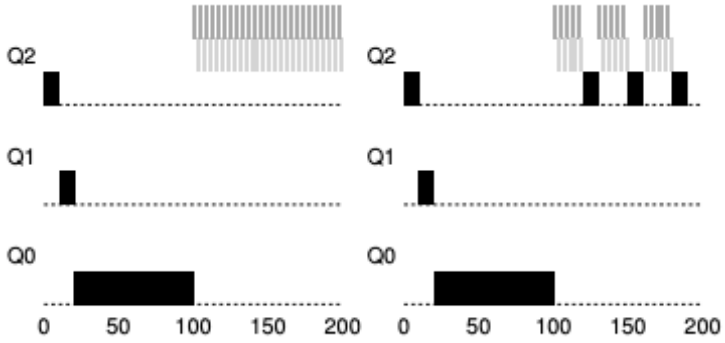


    - Is vulnerable to **starvation**
        - When only short-term jobs come in while a long term job is in queue

- **Round Robin**
    - Has good **response time** but terrible **turnaround time**
    - Runs job for a **time slice** or **quantum**
    - Each job gets equal share of CPU time
    - Is clock-driven [6]
    - Is starvation-free [7]
    - Must have the length of a time slice (**quantum**) as multiple of timer-interrupt period

```
void release(boolean *lock) {
        *lock = false;
}
```

- **Multi-level Feedback Queue**
    - Is the most well known approaches to shceduling
    - Optimizes **turnaround time**, and minimizes **response time**
    - Observees the execution of a job and priortizes accordingly without prior knowledge
    - Rules
        - **Rule 1:** If Priority(A) > Priority(B), A runs (B doesn't)
        - **Rule 2:** If Priority(A) = Priority(B). A & B run in round-robin fashion using the time slice (quantum length) of the given queue
        - **Rule 3:** When a job enters the system, it is placed at the highest priority(the top most queue)
        - **Rule 4:** Once a job uses up its time allotment at a given level (regardless of how many times it has given up the CPU), its priority is reduced (it moves down on queue)
        - **Rule 5:** After some time period S, move all the jobs in the system to the topmost queue.



10 User Mode

- Is restricted
- Executing code has no ability to *directly* access hardware or reference memory [1]
- Crashes are always recoverable [1]
- Is where most of the code on our computer / applications are executed [3]

11 Kernel Mode

- Is previleged (non-restricted)
- Executing code has complete and unrestricted access to the underlying hardware [3]
- Is generally reserved for the lowest-level, most trusted functions of the operating system [1]
- Is fatal to crash; it will halt the entire PC (i.e the blue screen of death) [3]

12 Interrupt

- i is a signals are sent by hardware (keyboardm mouse, etc.), or software (page fault, protection violation, system call)
- Tells the CPU to stop its current activities and execute the appropriate part of the operating system (**Interrupt Handler**). [2]
- Has three different types [2]

  1) **Hardware Interupts**
     - Are generated by hardware devices to signal that they need some attention from the OS.

- May be due to receiving some data

  Examples

  * Keystrokes on the keyboard
  * Receiving data on the ethernet card

- May be due to completing a task which the operating system previous requested

  Examples

  Transfering data between the hard drive and memory

2) **Software Interupts**

- Are generated by programs when a system call is requested

3) **Traps**

- Are generated by the CPU itself
- Indicate that some error or condition occured for which assistance from the operating system is needed

13 Trap

- Is an exception in a user process [1]
- Is an interrupt caused by an exceptional condition (breakpoint, division by zero, invalid memory access) [2]
- Is called by user
- Usually results in kernel mode by invoking **trap instruction**[2]

14 Trap Instruction

- when **Trap** or **System call** is invoked, **Trap instruction** simultaneously jumps into the kernel and raise the previlege level to kernel mode

15 Context Switch

- Is switching from running a user level process to the OS kernel and often to other user processes before the current process is resumed
- Happens during a timer interrupt or system call
- Saves the following states for a process during a context switch
  - Stack Pointer
  - Program Counter
  - User Registers
  - Kernel State
- May hinder performance

16 System Call

- Is the programmatic way in which a computer program requests a previleged service from the kernel of the operating system
- i.e. Reading from disk
- traps into the kernel so that priviliged instruction can be run
- Provides services via API or Application Program Interface
- Is the only entry points into the kernel system
- Is strictly a subset of software interrupts
- Steps
  1) Setup **trap tables** on boot
  2) Execute system call
  3) Save *Program Counter*, *CPU registers*, *kernal stack* (so process can resume after **return-from-trap** or **context switch**)
  4) Switch from **user mode** to **kernel mode**
  5) Perform previleged operations
  6) Finish and execute **return-from-trap** instruction
  7) Return from **kernel mode** to **user mode** and resume user program
- Has five different types

Types of System Calls	Windows	Linux
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Management	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Management	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()

Example

- `yield()`
  - Is a system call
  - Causes the calling thread to relinquish the CPU
  - Places the current thread at the end of the run queue
  - Schedules another thread to run

Example

`open()`, `read()`, `write()`, `close()`, `mkdir()` are other examples of system calls

17 Signals

- Provides a way to communicate with the process
- Can cause job to stop, continue, or terminate
- Can be delivered to an application
  - Stops the application from whatever its doing
  - Runs Signal handler (some code in application to handle the signal)
  - When finished, the process resumes previous behavior

18 CPU-bound process

[8]

- CPU Bound processes are ones that are implementing algorithms with a large number of calculations
- Programs such as simulations may be CPU bound for most of the life of the process.
- Users do not typically expect an immediate response from the computer when running CPU bound programs.
- They should be given a lower priority by the scheduler.

19 I/O-bound process

[8]

- Processes that are mostly waiting for the completion of input or output (I/O) are I/O Bound.
- Interactive processes, such as office applications are mostly I/O bound the entire life of the process. Some processes may be I/O bound for only a few short periods of time.
- The expected short run time of I/O bound processes means that they will not stay the running the process for very long.
- They should be given high priority by the scheduler.

20 Memory API

- Has two types of memory
  1. **Stack**
     - Is also called **automatic memory**
     - Allocations and deallocations are managed by compiler
     - Deallocates memory by the end of function call
  2. **Heap**
     - Is long-lived
     - Allocation and deallocation are managed by user
     - Creates **memory leak** if memory not freed
     - **valgrind** is a useful heap memoery debugging tool link
- `malloc()`

  - Is a C library call
  - **Syntax:** `void *malloc(size_t size)`
  - Allocates a block of `size` bytes to **heap memory** and if successful, returns a pointer to it
  - Returns `NULL` if memory allocation is unsuccessful

Example

```
int *x = malloc(10 * sizeof(int));
```

- `free()`
  - Is a C library call
  - Frees heap memory that is no longer in use

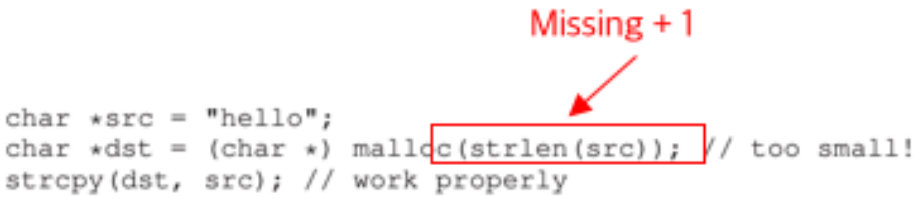
Example

```
int *x = malloc(10 * sizeof(int));
...
free(x);
```

- `brk(), sbrk(), mmap()`
  - Are system calls for memory management

21 Buffer overflow

- is an error that occurs when not enough heap memory is allocated



22 Virtualizing Memory

- Basic Idea: For the most part, let the program run directly on the hardware; however, at certain key points in time (e.g. system call, timer interrupt), arrange so that the OS gets involved and make sure the 'Right' thing happens.
- Like CPU, many programs are sharing the memory at the same time
- Like CPU, the goal is to create an illusion that it has its own code and data
- Like CPU, the memory also needs low-level machinery called **mechanism**, and high level intelligence called **policies**
- Steps

  1. Use **address translation** to transform each memory access, changing **virtual address** provided by instruction to **physical address**
     - Memory access includes instruction fetch, load, or store
     - Is done using hardware
  2. Invove OS at key points to **manage memory**

     Memory management includes

     1. Setting up hardware so correct translations take place
     2. Keeping track of which locations are free and which are in use
     3. Judiciously intervening to maintain control over how memory is used

23 Address Translation

- Is also called **hardware-based address translation**
- Is a mechanism of memory virtualization
- Is the technique of transforming virtual address to physical address