Operating Systems Design (CS 423)

Elsa L Gunter
2112 SC, UIUC
http://www.cs.illinois.edu/class/cs423/

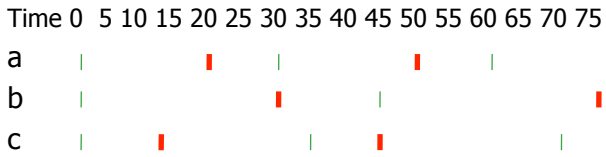
Based on slides by Roy Campbell, Sam King, and Andrew S Tanenbaum

Definitions

- Tasks - unit of work
- Periodic and aperiodic tasks
- Arrival time - time when task can begin execution
- Deadline – time by which task must be done
  - Can be relative or absolute
- Execution time - amount of time task needs to finish work

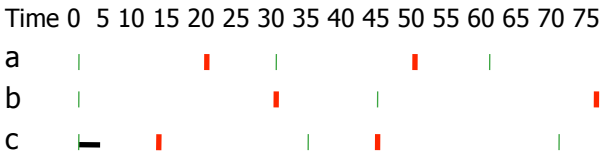
EDF example

- Job a: e(a) = 15, D(a) = 20 (period 30)
- Job b: e(b) = 10, D(b) = 30 (period 45)
- Job c: e(c) = 5, D(c) = 10 (period 35)



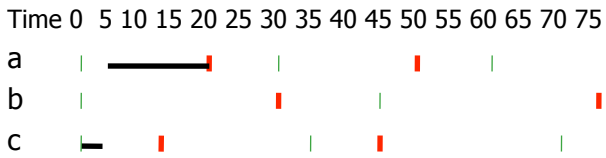
EDF example

- Job a: e(a) = 15, D(a) = 20 (period 30)
- Job b: e(b) = 10, D(b) = 30 (period 45)
- Job c: e(c) = 5, D(c) = 10 (period 35)



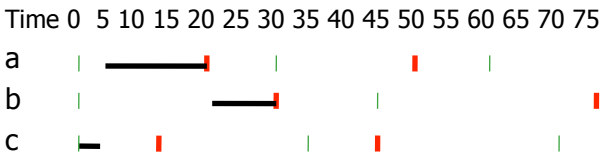
EDF example

- Job a: e(a) = 15, D(a) = 20 (period 30)
- Job b: e(b) = 10, D(b) = 30 (period 45)
- Job c: e(c) = 5, D(c) = 10 (period 35)



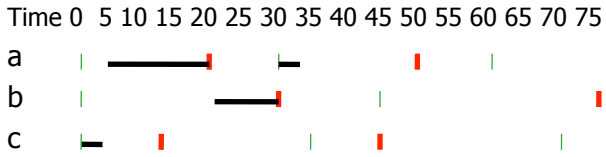
EDF example

- Job a: e(a) = 15, D(a) = 20 (period 30)
- Job b: e(b) = 10, D(b) = 30 (period 45)
- Job c: e(c) = 5, D(c) = 10 (period 35)



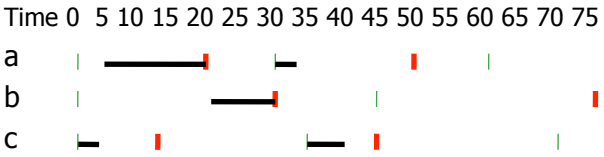
EDF example

- Job a: e(a) = 15, D(a) = 20 (period 30)
- Job b: e(b) = 10, D(b) = 30 (period 45)
- Job c: e(c) = 5, D(c) = 10 (period 35)



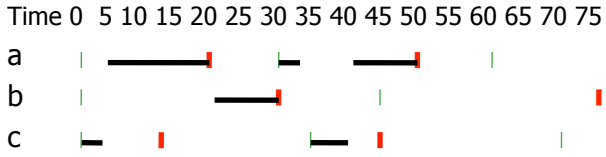
EDF example

- Job a: e(a) = 15, D(a) = 20 (period 30)
- Job b: e(b) = 10, D(b) = 30 (period 45)
- Job c: e(c) = 5, D(c) = 10 (period 35)



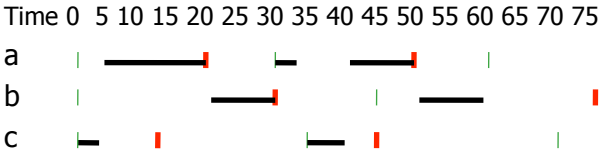
EDF example

- Job a: e(a) = 15, D(a) = 20 (period 30)
- Job b: e(b) = 10, D(b) = 30 (period 45)
- Job c: e(c) = 5, D(c) = 10 (period 35)



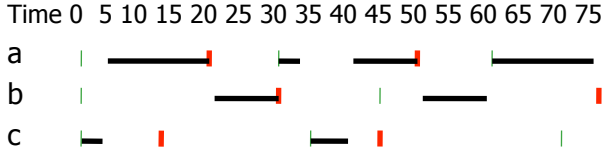
EDF example

- Job a: e(a) = 15, D(a) = 20 (period 30)
- Job b: e(b) = 10, D(b) = 30 (period 45)
- Job c: e(c) = 5, D(c) = 10 (period 35)



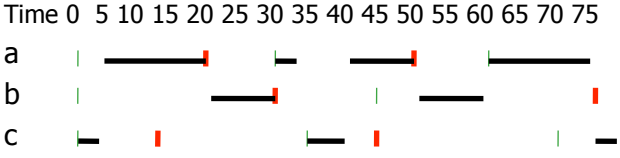
EDF example

- Job a: e(a) = 15, D(a) = 20 (period 30)
- Job b: e(b) = 10, D(b) = 30 (period 45)
- Job c: e(c) = 5, D(c) = 10 (period 35)



EDF example

- Job a: e(a) = 15, D(a) = 20 (period 30)
- Job b: e(b) = 10, D(b) = 30 (period 45)
- Job c: e(c) = 5, D(c) = 10 (period 35)



Address Spaces and Memory

- Process = one or more threads in an address space
- Thread: stream of execution
  - Unit of concurrency
- Address space: memory space that threads use
  - Unit of data

Address Space Abstrac2on

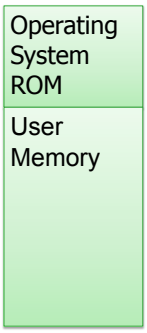
- Address space: all the memory data
  - Program code, stack, data segment
- Hardware interface (physical reality)
  - One memory, small, shared
- Application interface (illusion)
  - Each process has own memory, large

Illusions Provided by Address Apace

- Address independence
  - Same address in different processes not conflicting with each other
  - Eg Same address for stack
- Protection
  - One process cannot access the data of another
  - Secret data, protected code
- Virtual memory
  - 64 bit address space, memory many 4 G

Uni-programming

- One process runs at a time
- Always load process into the same spot
- How do you switch processes?
- What abstractions
- does this provide?
- Problems?



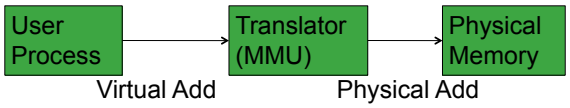
Multi-programming

- Multi-programming: more than one process in memory at a time
- Need address translation
  - Need protection
- Address translation
  - Avoid conflicting addresses
  - Static: before you execute
  - Dynamic: during execution, could change

Dynamic translation

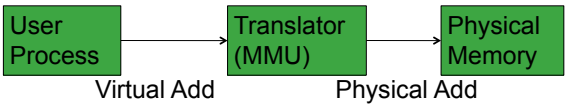
- Translate every memory reference from virtual address to physical address
- Virtual address: an address used by the user process to reference a location in memory
- Physical address: an address used by the physical memory

Dynamic Address Translation



- Translation enforces protection
  - One process can't even refer to another process's address space
- Translation enables virtual memory
  - A virtual address only needs to be in physical memory when it is being accessed
- Change translations on the fly as different virtual addresses occupy physical memory
- Do you need hardware support?

Address translation



- Lots of ways to implement, remember the big picture
- Tradeoffs:
  - Flexibility (e.g., sharing, growth, virtual memory)
  - Size of translation data
  - Speed of translation