Page size $=2^n=1024$ B$=2^{10}$ B
Of bits in offset part (n) =10

Assuming a 1-KB page size, what are the page numbers and offsets for the following address references (provided as decimal numbers)

Solution steps:

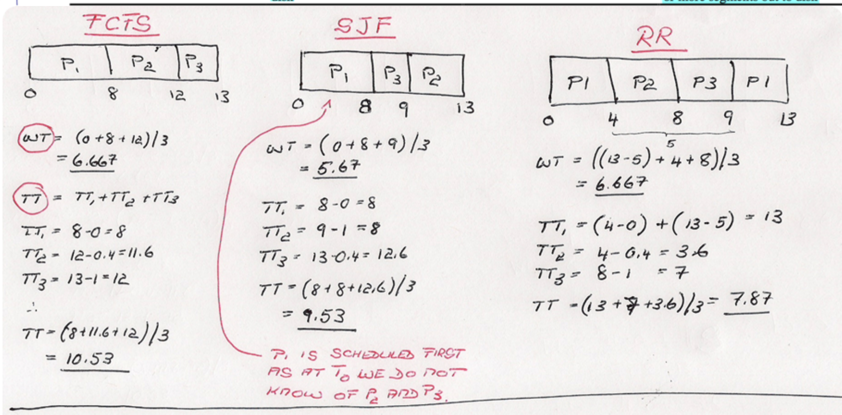
1. Convert logical address: Decimal → Binary
2. Split binary address to 2 parts (page #, Offset), offset : *n* digits
3. Convert offset & page# : Binary→ Decimal

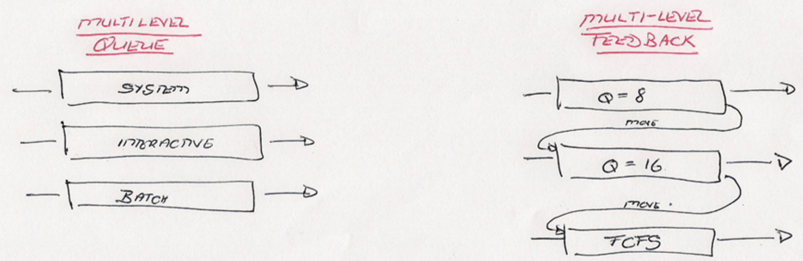
Logical address (decimal)	*Logical address (binary)*	*Page # (6 bits) (binary)*	*Offset (10 bits) (binary)*	*Page # (decimal)*	*Offset (decimal)*
2375	0000 1001 0100 0111	0000 10	01 0100 0111	2	327
19366	0100 1011 1010 0110	0100 10	11 1010 0110	18	934
30000	0111 0101 0011 0000	0111 01	01 0011 0000	29	304
256	0000 0001 0000 0000	0000 00	01 0000 0000	0	256
16385	0100 0000 0000 0001	0100 00	00 0000 0001	16	1

Technique	Description	Strengths	Weaknesses
Fixed Partitioning	Main memory is divided into a number of static partitions at system generation time. A process may be loaded into a partition of equal or greater size.	Simple to implement; little operating system overhead.	Inefficient use of memory due to internal fragmentation; maximum number of active processes is fixed.
Dynamic Partitioning	Partitions are created dynamically, so that each process is loaded into a partition of exactly the same size as that process.	No internal fragmentation; more efficient use of main memory.	Inefficient use of processor due to the need for compaction to counter external fragmentation.
Simple Paging	Main memory is divided into a number of equal-size frames. Each process is divided into a number of equal-size pages of the same length as frames. A process is loaded by loading all of its pages into available, not necessarily contiguous, frames.	No external fragmentation.	A small amount of internal fragmentation.
Simple Segmentation	Each process is divided into a number of segments. A process is loaded by loading all of its segments into dynamic partitions that need not be contiguous.	No internal fragmentation.	weakness: c… Improved memory utilization and reduced overhead compared to dynamic partitioning.
Virtual-Memory Paging	As with simple paging, except that it is not necessary to load all of the pages of a process. Nonresident pages that are needed are brought in later automatically.	No external fragmentation; higher degree of multiprogramming; large virtual address space.	Overhead of complex memory management.
Virtual-Memory Segmentation	As with simple segmentation, except that it is not necessary to load all of the segments of a process. Nonresident segments that are needed are brought in later automatically.	No internal fragmentation, higher degree of multiprogramming; large virtual address space; protection and sharing support.	Overhead of complex memory management.

V.S

	Simple Paging	Virtual Memory Paging	Simple Segmentation	Virtual Memory Segmentation
Physical Structure	Main memory partitioned into small fixed-size chunks called frames	Main memory partitioned into small fixed-size chunks called frames	Main memory not partitioned	Main memory not partitioned
Logical Structure	Program broken into pages by the compiler or memory management system	Program broken into pages by the compiler or memory management system	Program segments specified by the programmer to the compiler (i.e., the decision is made by the programmer)	Program segments specified by the programmer to the compiler (i.e., the decision is made by the programmer)
Internal fragmented?	Internal fragmentation within frames	Internal fragmentation within frames	No internal fragmentation	No internal fragmentation
External fragmented?	No external fragmentation	No external fragmentation	External fragmentation	External fragmentation
Required info per process?	Operating system must maintain a page table for each process showing which frame each page occupies	Operating system must maintain a page table for each process showing which frame each page occupies	Operating system must maintain a segment table for each process showing the load address and length of each segment	Operating system must maintain a segment table for each process showing the load address and length of each segment
Unused space?	Operating system must maintain a free frame list	Operating system must maintain a free frame list	Operating system must maintain a list of free holes in main memory	Operating system must maintain a list of free holes in main memory
Calculate abs address	Processor uses page number, offset to calculate absolute address	Processor uses page number, offset to calculate absolute address	Processor uses segment number, offset to calculate absolute address	Processor uses segment number, offset to calculate absolute address
All page/segment required in memory?	All the pages of a process must be in main memory for process to run, unless overlays are used	Not all pages of a process need be in main memory frames for the process to run. Pages may be read in as needed	All the segments of a process must be in main memory for process to run, unless overlays are used	Not all segments of a process need be in main memory frames for the process to run. Segments may be read in as needed
Swapping required?		Reading a page into main memory may require writing a page out to disk		Reading a segment into main memory may require writing one or more segments out to disk





- **Long term scheduler** control the degree of programming
- **Sync on OSes**: - Solaris: Adaptive Mutex, Reader-Writer Locks, Turnstile (Q); XP:singleproc:disable interrupt,multi:spinlocks & DispatcherObject; Linux: same with no Dispatcher
- **Effective Access Time** = ∑ prob x accesstime of each case
- **On UMA systems, accessing RAM takes the same amount of time from any CPU**. On NUMA systems, accessing some parts of memory may take longer than accessing other parts of memory, thus creating a performance penalty for certain memory accesses.
- **Paging arithmetic laws**: Page size = frame size

Logical address space (/size) = 2m | Page size= frame size= 2n
Physical address space (/size) = 2x (x: bits in physical address
Logical address space (/size) = # of pages × page size
Physical address space (/size) = # of frames × frame size
# of pages= 2m-n # of entries (records)in page table = # of pages

- **predict next CPU burst**: Tn+1= α * tn + (1- α) * τn
- to ensure that **Circular Wait** condition never holds is to impose a total ordering of all resource types, and to require that each process requests resources in an increasing order of enumeration.
- The **benefits of multithreaded programming** fall into the categories: responsiveness, resource sharing, economy, and utilization of multiprocessor architectures.
- **Blocking and Nonblocking I/O**

➢**Blocking** process suspended until I/O completed
➢**Nonblocking** Returns quickly with count of bytes read or written
➢**Asynchronous** I/O subsystem signals process when I/O completed

- **System Call Parameter Passing:** methods used to pass parameters to the OS

1. Simplest: pass the parameters in **registers**
2. Parameters stored in a **block**, or table, in memory, and ***address*** of block passed as a parameter in a register. This approach taken by Linux and Solaris
3. Parameters placed, or ***pushed***, onto the **stack** by the program and ***popped*** off the stack by the operating system

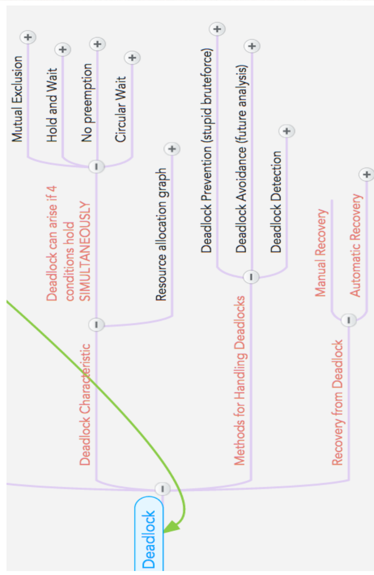
Types of System Calls:
Process control - File management - Device management - Information maintenance- Communications - Protection

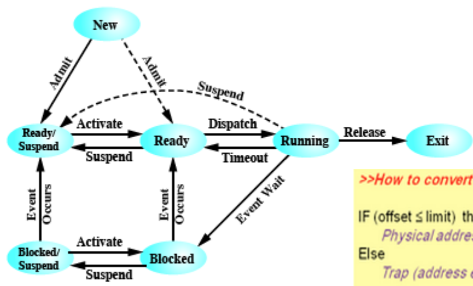
CPU scheduling decisions may take place when a process:
o Switches from **running** *to* **waiting** state o Switches from **running** *to* **ready** state o Switches from **waiting** *to* **ready**
o **Terminates**

Scheduling Criteria

· **CPU utilization** – keep the CPU as busy as possible
· **Throughput** – # of processes that complete their execution per time unit
· **Turnaround time** – amount of time to execute a particular process
· **Waiting time** – amount of time a process has been waiting in the ready queue
· **Response time** – amount of time from when a request was submitted until the first response is produced.

- **fork** retun 0 indicates the child process, >0 is parent.

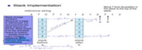




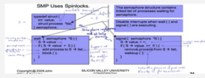
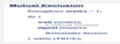
>>How to convert from logical address to physical address in segmentation???

IF (offset ≤ limit) then
Physical address= base address+ offset
Else
Trap (address error)

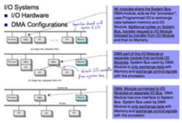
Page Replacement Algorithms

- FIFO
- Optimal
  - Replace page that WILL NOT be used for longest period of time
  - Lowest possible Page-Fault Rate
  - Difficult to implement as it needs future referencing knowledge
  - Impractical, not widely used.
- LRU
  - Replace page that HAS NOT been used for the longest time
    - Similar to FIFO, except recently used page is move to the top of the queue (so it will pop out last)
  - 
    Stack implementation of LRU
    - keep a stack of page numbers and move recently referenced page to the top
  - 
    Counter implementation of LRU
    - Every page has a counter, everytime a page is referenced move the counter to this page and copy the clock into the counter. When a page need to replace, find page that has the smallest clock value.
- LRU-Approximation
  - Use Counter Implementation and adding a Reference Bit and a Modified Bit (x,y) to offer 2nd chance algorithm
    - (0,0) <-- Replace
    - (0,1) <-- Give 1 chance
    - (1,0) <-- Give 2 chances
    - (1,1) <-- Give 3 chances
- LFU
  - Using the counter implementation – replace page with LEAST count
- MFU
  - Using the counter implementation – replace with MOST count

Semaphores

- Semaphore: A semaphore S is a variable or abstract data type that is used to control access, by multiple processes, to a common resource in parallel programming or a multi-user environment.
  -
  - two standard operations modify S
- Semaphore Implementation with No Busy Waiting (Block/Sleep)
  - No more looping
  - 
  - Each semaphore has an associated waiting queue and
- Counting Semaphores
  - Control access to a finite instances of shared resources
  - Also use Wait (S) and Signal (S)
- Binary Semaphores (Mutexes)
  - Semaphore S is either "0" or "1"
  - is used for locking resource
  - 
  - Mutual Exclusion: Semaphore implementation must guarantee that no two processes can execute wait () and signal () on the same semaphore at the same time.
- Spinlocks
  - Semaphore with Busy Waiting
  - Thread "spins" on a processor for lock
  - Only for short waiting time for lock
- 3 potential problems when using semaphore
  - 
  - Deadlock - Two or more processes are waiting indefinitely for an event that is caused by one of the waiting processes.
  - Starvation or Infinite Blocking - Process wait indefinitely within Semaphore
  - Priority Inversion - Higher priority task waiting on a semaphore held by a lower priority task, which is interrupted by another task (with medium priority). And thus the higher task has to wait.

3 DMA configurations

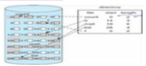
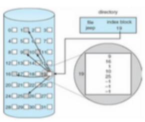
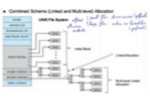


- Single-bus, detached DMA
- Single-bus, integrated DMA-I/O
- IO Bus

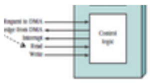
Frame Allocation Algorithm

- Each process needs a minimum number of pages, there are 2 major allocation schemes
  - FIXED equal allocation
    - Equal or proportional allocation base on the process SIZE
  - PRIORITY allocation
    - Use a proportional allocation scheme base on PRIORITY (not size)
- Global v/s Local Frame Allocation
  - GLOBAL Replacement – process selects a replacement frame from the set of all frames, including from other processes
  - LOCAL Replacement – process selects from its own allocated frames

File Block Allocation Methods

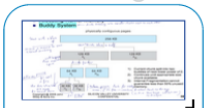
- Contiguous Allocation
  - 
  - data occupy set of contiguous blocks
  - Pros:
    - minimal head movement, seek minimal
  - Cons
    - finding contiguous space for new file
    - externel fragmentation
- Linked Allocation
  - 
    Each file is a linked list of disk blocks, scattered on disk.
  - Each block contains 4 Byte pointer to the next block
  - Pros:
    - file size can grow freely as free blocks are available
    - efficient for Sequential Access
    - Directory contains offset to the start of the file in FAT --> does not need to go to disk to find a file / sector
  - Cons:
    - Inefficient for Direct Access
    - internal fragmentation problem
  - uses by FAT
- Indexed Allocation
  - 
    Each file has index block storing all pointers to each data block (similar concept: page table)
- Combine Scheme
  - 
    UNIX file system
    - small files: direct access
    - large files: index or linked list
- Performance
  - Sequential Access
    - use Linked Allocation
  - Random Access
    - Direct Access
    - Contiguous Allocation

DMA Controller



- Data Count
- Data Register
- Address Register
- Control Logic
- special processor for memory bus operation

Kernel Memory Allocation

- Characteristics
- 
  Solution 1: Buddy System
- Solution 2: Using SLAB

- Windows XP
  - Uses Working Set Minimum and Working Set Maximum
  - uses both Demand Paging + Clustering (load pages surrounding the faulting pages)
  - Page Replacement using Local Replacement Algorithm and other variation of FIFO algorithms
- Solaris
  - Scan rate changes according to amount of free pages

Disk Scheduling Methods

- Choose the next pending request to service
  - Random Access Time = Seek Time + Rotational Latency
  - Positioning time (Random Access Time) is time to move disk arm to desired cylinder (Seek Time) + rotating time of desired sector to disk head (Rotational Latency)
- FCFS (First Come First Serve)
  - large swing of disk head possible, can cause lengthy Access Time
    - not fastest
- SSTF (Shortest Seek Time First)
  - Select request with minimum seek time from the current head position
  - May cause starvation of some requests
- SCAN (aka elavator algorithm)
  - Disk arm starts from one end of disk, moves toward the other end and on the way it services request until it gets to the other end of the disk
- C-SCAN
  - Variant of SCAN, except that when it reaches the other end, it immediately returns to the beginning of the disk without servicing any requests on the return trip.
  - Provides a MORE UNIFORM WAIT TIME than SCAN
- C-LOOK
  - Variant of C-SCAN, except that arm only goes as far as the last request in each direction, and then reverses direction immediately.
- Selecting a disk-scheduling algorithm
  - SSTF is common, performs better than FCFS
  - SCAN and C-SCAN performs better for systems that place a heavy load on disk
  - SSTF or C-LOOK are most likely used as default