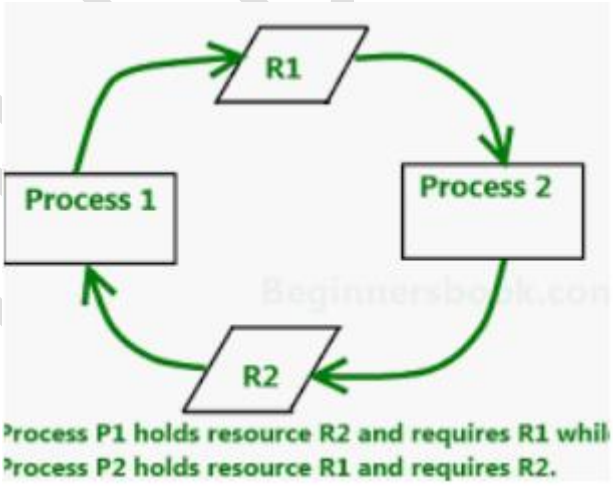
**ANDROID OPERATING SYSTEM – PERSONALISED SHORT NOTES FOR END SEMESTER EXAMS**

1. **Deadlock – conditions, prevention and prevention schemes.**
    a. Permanent blocking
    b. Set of processes
    c. Compete for the system resources / communication
    d. Process is blocked
    e. Waiting for event
    f. Can be only triggered by another blocked process
    g. None of the process complete
    h. **Conditions.**
        i. Mutual exclusion – one resource for one process (*necessary condition*)
        ii. Hold and wait – hold resources; await for assignment of other resources (*necessary condition*)
        iii. No pre-emption – no forced removal of resource from a process (*necessary condition*)
        iv. Circular wait – shown below (*not necessary condition*)
    i. **Prevention and Prevention Scheme.**
        i. Avoid any of three conditions
        ii. Allocation Policy – Conservative, under commits resources
        iii. **Requesting all resources at once.**
            1. *Advantages* – well for process of single burst activity; no pre-emption necessary
            2. *Disadvantages* – inefficient, delays process initiation, future requirements of resources must be known
        iv. **Pre-emption.**
            1. *Advantages* – convenient, resources' state can be saved and restored
            2. *Disadvantages* – pre-empts are often more than necessary
        v. **Resource ordering.**
            1. *Advantages* – no run time computation, problem solved in system design
            2. *Disadvantages* – disallows incremental resource requests



2. **Definitions.**
    a. **Atomic operation.**
        i. Function / action
        ii. Sequence of one / more instructions
        iii. Appears to be indivisible
        iv. No other process can interrupt
        v. Executes as a group
        vi. Guarantees isolation from concurrent processes
    b. **Live-lock.**
        i. Situation
        ii. No useful work
        iii. Two / more process continuously change their states
        iv. In response to changes in other processes
    c. **Critical section.**
        i. Section of code
        ii. Within a process
        iii. Requires access to shared resources
        iv. Will not be executed if another process is in corresponding section of code
    d. **Race condition.**
        i. Situation
        ii. Multiple threads / processes
        iii. Read / write shared data item
        iv. Final result depends on relative timing of their execution
    e. **Thrashing.**
        i. State
        ii. System spends most time
        iii. Swapping process pieces
        iv. Instead of executing instructions
    f. **Multi – threading.**
        i. Ability of OS
        ii. To support multiple, concurrent paths of execution
        iii. Within a single process
    g. **Semaphore.**
        i. Counting / general semaphore
        ii. Integer value
        iii. Signalling among processes
        iv. 3 atomic processes – increment, decrement, initialise
        v. Increment – unblocking of process
        vi. Decrement – blocking of process
    h. **Monitor.**
        i. Programming language construct
        ii. Encapsulates variables, access procedures, initialisation code
        iii. Within abstract data type
        iv. Variables can be accessed by access procedures
        v. Access procedures are critical sections
        vi. One process accessing the monitor at one time
        vii. Queue of process waiting to access monitor
    i. **Program counter.**
        i. Register in computer processor
        ii. Has address of the instruction currently being executed
        iii. Points to the next instruction in sequence
        iv. Reverts to 0 when computer restarts

3. **Phases of Virus.**
    a. **Dormant Phase.**
        i. Idle
        ii. Not all viruses have this state
        iii. Waiting for activation event
    b. **Propagation Phase.**
        i. Identical copies
        ii. Attaches to other areas on disk
    c. **Triggering Phase.**
        i. Activated
        ii. Performs intended function
        iii. Caused by system events
    d. **Execution Phase.**
        i. Function performed
        ii. Maybe harmless
        iii. Maybe damaging – destruction of programs / files / both

4. **Internal and External Fragmentation.**
    a. **Internal Fragmentation.**
        i. Fixed sized memory block
        ii. Allocated to process
        iii. Memory given > memory requested
        iv. Free spaces causes internal fragmentation
        v. *Solution* – partitioned into variable sized blocks, best fit to process
    b. **External Fragmentation.**
        i. Variable sized blocks
        ii. Dynamically
        iii. Removal of process causes free space
        iv. This is external fragmentation
        v. *Solution* – compaction, paging, segmentation

5. **CIA Triad.**
    a. **Confidentiality.**
        i. Preserving restricted access
        ii. To authorised users
        iii. On information access; disclosure
        iv. Protects personal privacy; proprietary (exclusive, patented) information
        v. *Loss* – unauthorised disclosure of information
    b. **Integrity.**
        i. Guarding against improper / false info
        ii. Keeping the info / data consistent and authentic
        iii. *Loss* – unauthorised modification or destruction of info
    c. **Availability.**
        i. Ensuring timely and reliable access to info and use info
        ii. *Loss* – disruption of access to (or use) info; info system
    d. Three fundamental concepts which embody security objectives for data, information and computation services.

6. **Different memory management techniques.**

| Technique | Description | Strengths | Weakness |
|---|---|---|---|
| Fixed partitioning | • Main memory division<br>• Fixed size<br>• Static partitions<br>• Process loaded into equal / greater size partition<br>• OS can swap a terminated process with a new process if all partitions are loaded | • Simple to implement<br>• Less OS overhead | • Inefficient<br>• Internal fragmentation<br>• Max no. of active fixed processes<br>• Program may be too big to fit in partition |
| Dynamic partitioning | • Process sized blocks<br>• Created dynamically | • Efficient<br>• No internal fragmentation | • External fragmentation<br>• Inefficient use of processor |
| Simple paging | • Memory into frames<br>• Process into pages<br>• Same size of pages and frames<br>• Whole process (in terms of pages) is loaded into frames<br>• Not necessary continuous frames | • No external fragmentation | • Small amount of internal fragmentation |
| Simple segmentation | • Process into segments<br>• Loaded into dynamic partitions<br>• Not necessary continuous | • No internal fragmentation<br>• Reduced overhead as compared to dynamic partitioning<br>• Improved memory utilisation | • External fragmentation |
| Virtual memory paging | • Not necessary to load all pages of process<br>• Can be loaded later automatically | • No external fragmentation<br>• Large virtual address space<br>• Higher degree of multiprogramming | • Overhead of complex memory management |
| Virtual memory segmentation | • Not necessary to load all segments of process<br>• Can be loaded in the partitions | • No internal fragmentation<br>• Protection; sharing support<br>• Large virtual address space | • Overhead of complex memory management |

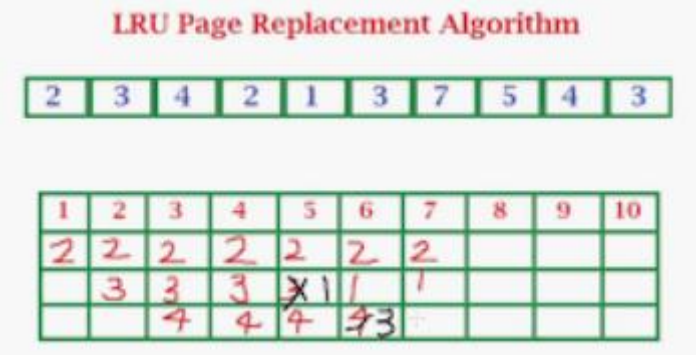
| | | | |
|---|---|---|---|
| | automatically later | • Same last point | |

**To check for internal and external fragmentation in each technique.**

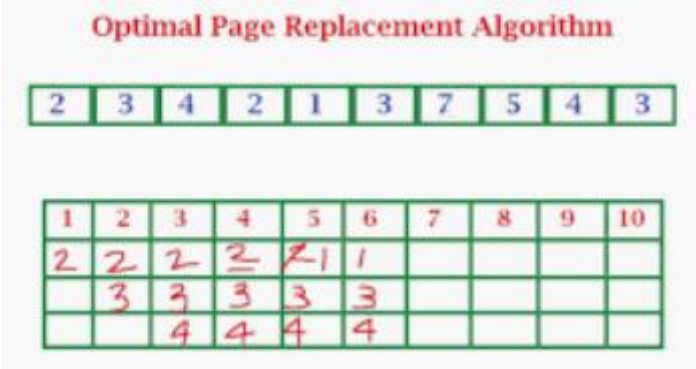
| Technique | Internal fragmentation | External fragmentation |
|---|---|---|
| Fixed partitioning | YES | - |
| Dynamic partitioning | NO | YES |
| Simple paging | YES | NO |
| Simple segmentation | NO | YES |
| Virtual memory paging | YES | NO |
| Virtual memory segmentation | NO | YES |

7. **Types of replacement algorithms.**
   a. **First in first out (FIFO).**
      i. Page frames; process in circular buffer
      ii. Pages removed in round robin style
      iii. Pages in main memory kept as list
      iv. Easy to implement
      v. Page replaced which is in memory the longest
      vi. Easy to understand
      vii. Disadvantage – oldest block in the memory is most used


   b. **Least recently used (LRU).**
      i. Replaces a page which has not been referenced for longest time
      ii. Based on the logic, same page is least likely to be needed in future
      iii. Almost same as optimal replacement
      iv. Efficient
      v. Disadvantage – difficult to implement, high overhead and expensive



c. **Optimal replacement.**
   i. Page is replaced which will not be needed for longest time
   ii. OS keeps track of all pages referenced by program
   iii. Optimal; fewest no. of page faults
   iv. Disadvantage – impossible to implement; impossible to predict the future



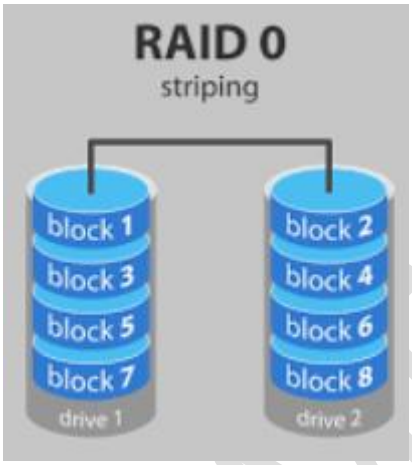

d. **Clock policy.**
   i. Called as second chance replacement algorithm
   ii. Basic logic of FIFO
   iii. Has ref and valid key columns
   iv. When new page enters frame, valid key = 1 and ref key = 0
   v. Same page referenced then valid = 1 and ref key = 1
   vi. New page enters and acc. to FIFO logic, above page needs to removed
   vii. It won't be removed since ref key = 1; given second chance; ref key set to 0
   viii. Visualise frames laid out in circle
   **(for more information, or diagram, please refer internet)**

8. **DMA.**
   a. Direct memory access
   b. Mimics processor
   c. Takes control of system bus
   d. Transfers data to and from memory over system bus
   e. Processor sends signal to DMA if it wishes to read / write block of data
   f. **Processor sends following to DMA.**
      i. Read / write operation using read / write control line
      ii. Address of I / O device involved (printer, keyboard etc.)
      iii. Starting location in memory (communicated on data lines; stored by DMA module in address register)
      iv. Number of words to be read / written (stored in data count register)
   g. Processor continues with other work
   h. Has passed on the operation to DMA module
   i. DMA module transfers full block of data
   j. One word at a time to or from memory
   k. Does not involve processor
   l. Sends interrupt signal to processor once transfer is complete
   m. Processor involved at beginning and end
   n. Saves time, saves cost, and efficient
   o. Processor can take up heavy tasks; or is free

**9. RAID and its types.**

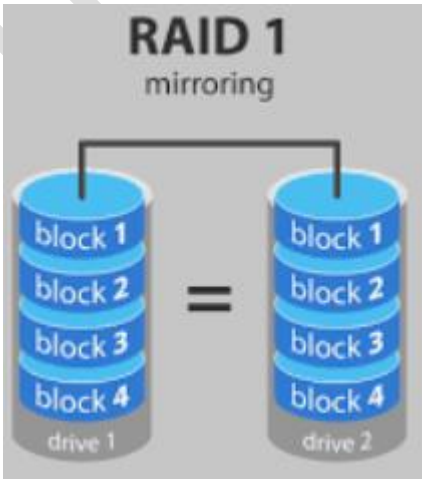
a. **RAID 0.**

i. Non redundant
ii. Not true RAID
iii. Data across all disks
iv. 2 unrelated I / O tasks can be done in parallel if on different disks
v. Reduces I / O transfer time if single I / O request consists multiple logically continuous strips and can be handled in parallel

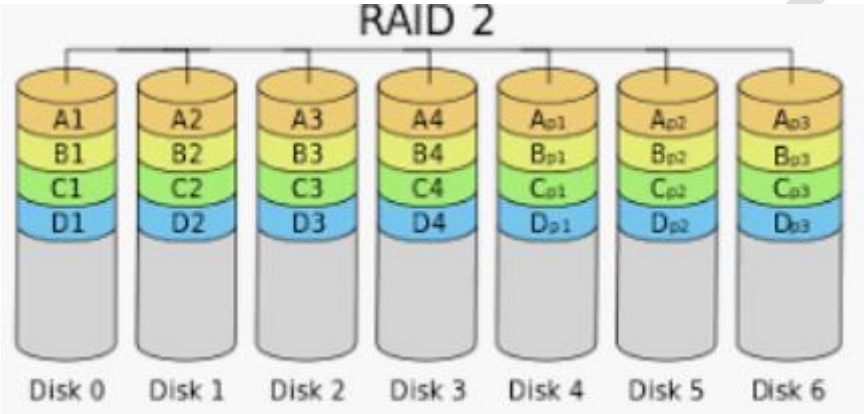



b. **RAID 1.**

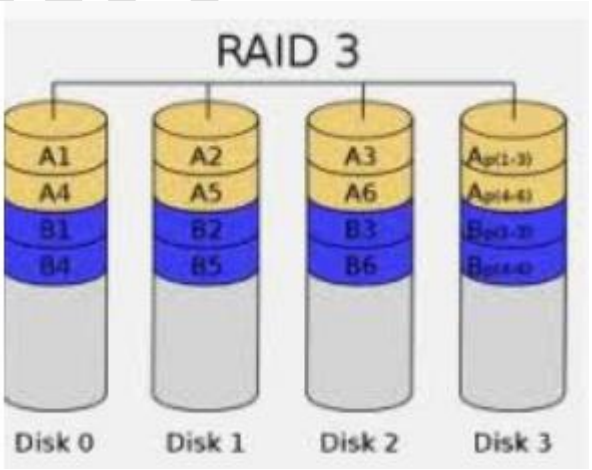
i. Mirrored
ii. Redundancy due to duplicating data on all disks
iii. No write penalty
iv. Drive fails; data accessible on 2nd drive; simple recovery
v. Major disadvantage is the cost
vi. Read request – serviced by either disks; involves disk with minimum seek time & rotational latency
vii. Write request – requires both disks to be updated; can be done in parallel



c. **RAID 2.**
   i. Redundancy through Hamming code
   ii. Parallel access technique
   iii. All disks involved in each operation
   iv. Data striping; using small strips
   v. Hamming code is used
   vi. Correct / detect single bit error
   vii. Detect double bit errors
   viii. Good choice when prone to many errors
   ix. Otherwise overkill; require too many disks



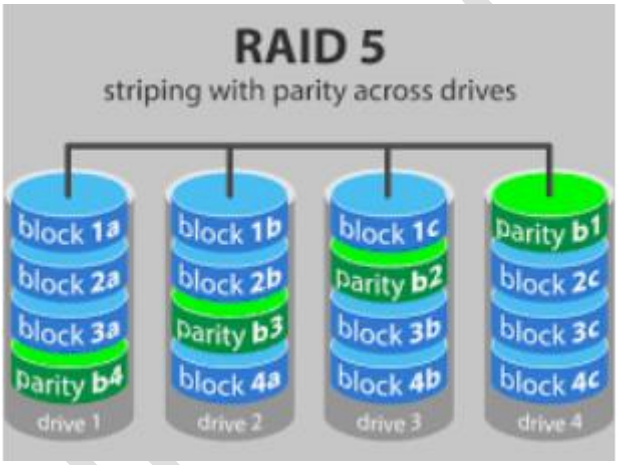

d. **RAID 3.**
   i. Bit interleaved parity
   ii. Single redundant disk
   iii. Parity disk helps recover any other disk failure
   iv. Employs parallel access
   v. Data distributed in small strips
   vi. High data transfer rates



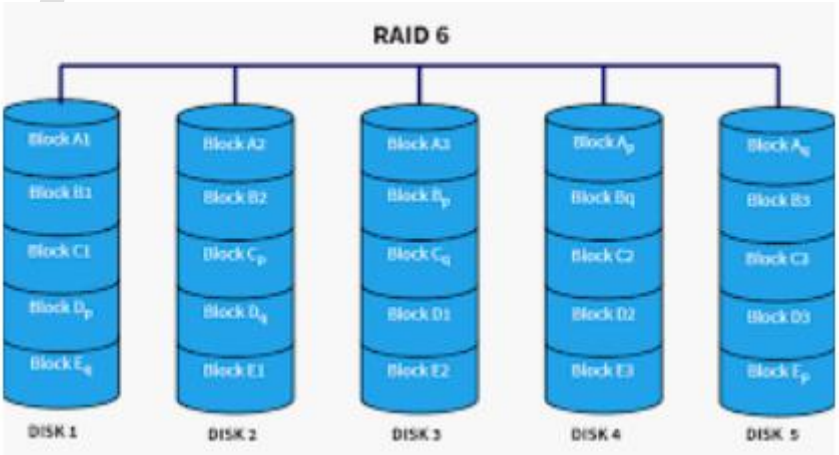
e. **RAID 4.**
   i. Block level parity
   ii. Independent access technique
   iii. Bit by bit parity strip
   iv. Across corresponding strips on each data disk
   v. Parity bits stored in corresponding strips on parity disk
   vi. Has write penalty when I / O write request of small size is performed

**(for more information, or diagram, please refer internet)**

f. **RAID 5.**
   i. Block level distributed parity
   ii. Parity bits across all disks
   iii. Round robin scheme
   iv. Loss of any one disk does not result in data loss




g. **RAID 6.**
   i. Dual redundancy
   ii. 2 different parity calculations
   iii. Stored on separate blocks in different disks
   iv. Extremely high availability
   v. Considerable write penalty; affects two parity blocks



10. **Master Slave Architecture.**
    a. Key kernel functions of OS run on particular processor
    b. Other processors run user programs
    c. Master responsible for scheduling jobs
    d. When slave needs service; sends requests to master; wait for service to be performed
    e. Simple approach; requires little enhancement; uniprocessor multiprogramming OS
    f. Simplified conflict resolution; one processor having control of all memory and I / O resources
    g. *Disadvantages* – failure of master brings down whole system, master can become performance bottleneck

**(for more information, or diagram, please refer internet)**

11. **Load sharing – pros and cons; different versions.**
    a. Form of thread scheduling
    b. Process is not assigned to particular processor
    c. Global queue of ready threads maintained
    d. Idle processor selects one from queue
    e. **Advantages.**
        i. Load distributed evenly among processors
        ii. No centralised scheduler required
        iii. Scheduling routine is run on processor to select next thread
        iv. Global queue can be organised in different schemas
    f. **Disadvantages.**
        i. Central queue occupies region of memory
        ii. Bottleneck if many processors look for work at the same time
        iii. Pre-empted threads are unlikely to resume on same processor
        iv. Caching is less efficient if each processor has a local cache
        v. Unlikely that all threads of same program gets access to processors at same time
        vi. Process switches compromises performance
    g. **Versions of Load sharing.**
        i. **First come first serve.**
            1. Threads of the new job is placed at the end of shared queue
            2. Processor keeps on taking next ready threads
            3. Which it executes until completion / blocking
        ii. **Smallest number of threads first.**
            1. Shared queue is priority queue
            2. Highest priority to job with least number of unscheduled threads
            3. If equal priority case, then *first come first serve*
        iii. **Pre-emptive smallest number of threads first.**
            1. Base logic same as *smallest number of threads first*
            2. If new job has less number of unscheduled threads, then executing job will pre-empt threads belonging to scheduled job

12. **Threads – types, its differences and benefits.**
    a. Path of execution within process
    b. Called lightweight process
    c. Process can have multiple threads
    d. **Types.**
        i. **User Level Thread.**
            1. All work of thread management is done by application
            2. Kernel not aware of existence of threads
            3. **Threads library.**
                a. Code for creating / destroying threads
                b. Passing data and message between threads

c. Scheduling thread execution
d. Saving and restoring thread context

4. Kernel manages it as it is single threaded process
5. **Advantages.**
    a. Thread switching; not require kernel mode privileges; thread management; data structure; within user address space of single process
    b. No need to switch to kernel mode
    c. Saving overhead of switching modes
    d. Scheduling can be application specific
    e. Scheduling types – round robin, priority based; depends on application
    f. Can run on any OS
    g. No changes required to make in underlying kernel to support threading
    h. Thread library – set of application level functions shared by all applications
6. **Disadvantages.**
    a. Many system calls are blocking; user level thread executing system call often blocked; all threads within process is blocked
    b. Multithread application cannot take benefit of multiprocessing
    c. Kernel assigns one process to one processor

ii. **Kernel Level Thread.**

1. Thread management done by kernel
2. Not by application
3. Maintains context information about process as whole
4. Even for every individual thread within the process
5. **Advantages.**
    a. Schedule different threads of the same process on different processors
    b. If one thread is blocked, kernel can schedule another of the same process
6. **Disadvantages.**
    a. Transfer of control; one thread to another; within same process; mode switch to kernel
    b. Increases thread operation latencies

e. **Difference between User level and Kernel level Thread.**

| Characteristic | User Level Thread | Kernel Level Thread |
|---|---|---|
| Implementation | Users and application | Operating system |
| Recognition | OS / kernel doesn't recognise | Recognised by OS / kernel |
| Difficulty | Easy | Complicated |
| Context switch time | Less | More |
| Hardware support | Not required | Needed |
| Blockage | Entire process will be blocked | Another thread of the same process can continue execution |
| Example | Java thread, POSIX thread | Windows Solaris |

f. **Benefits of threads.**

i. **Responsiveness.**

1. One thread execution completion
2. Output is immediately returned

ii. **Faster context switch.**

1. Lesser time than process context switch
2. Faster
3. Process context switch requires more overhead from CPU

iii. **Effective utilisation of multiprocessor system.**

1. Multiple threads of same process on different processors
2. Faster execution

iv. **Resource sharing.**

1. All resources shared among all threads within process

v. **Communication.**

1. Easier communication between threads
2. Threads share same address space

**13. Difference between Processes and Threads.**

| Characteristic | Processes | Threads |
|---|---|---|
| Program | Program in execution | Lightweight process / part of process |
| Memory | Isolated; does not share memory | Shares memory with each other |
| Resource consumption | High resource consumption | Low resource consumption |
| Create and terminate | More time to create and terminate | Less time to create and terminate |
| Context switch | More time to context switch | Less time to context switch |
| Communication | Can't communicate with other processes | Can communicate with other threads |

**14. Modes of operation execution.**

a. **User mode.**
   i. User program executes in user mode
   ii. Certain area of memory is protected from user's use
   iii. Certain instructions may not be executed
   iv. Executing code cannot access hardware or reference memory
   v. Code must delegate to system API's to access above 2 mentioned
   vi. Crashes recoverable
   vii. Most of the codes run in user mode

b. **Kernel mode.**
   i. Executing code; complete and unrestricted access to underlying hardware
   ii. Execute any memory address
   iii. Reserved for lowest level
   iv. Most trusted functions of OS
   v. Crashes will halt the computer
   vi. Monitor executes in kernel mode

**15. Buddy system and buddy tree representation.**

a. Memory allocation and memory management algorithm
b. Manages memory in power of 2 increments
c. Suppose memory size is 2^U and S size is required
d. If 2^(U-1) < S <= 2^U, allocate whole block
e. Else, recursively divide block until above condition is satisfied
f. System keeps record of unallocated blocks
g. Can merge these different sized blocks to make one big chunk
h. **Advantages.**
   i. Easy to implement
   ii. Allocates block of correct size
   iii. Easy to merge adjacent holes
   iv. Fast to allocate and de-allocate memory
i. **Disadvantages.**
   i. Leads to internal fragmentation
   ii. Allocation units must be power of 2

j. **Diagram.**

i.

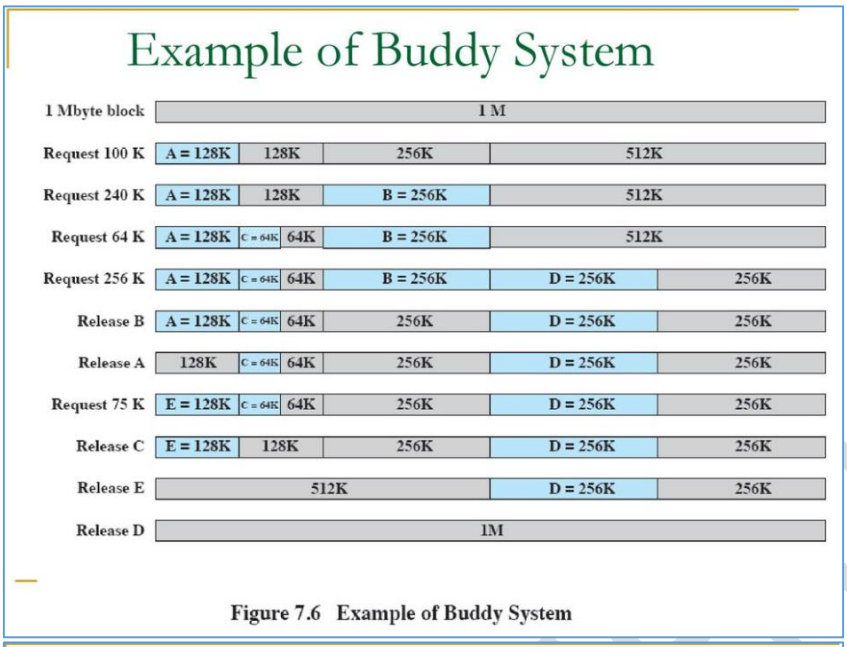



Figure 7.6 Example of Buddy System

ii.

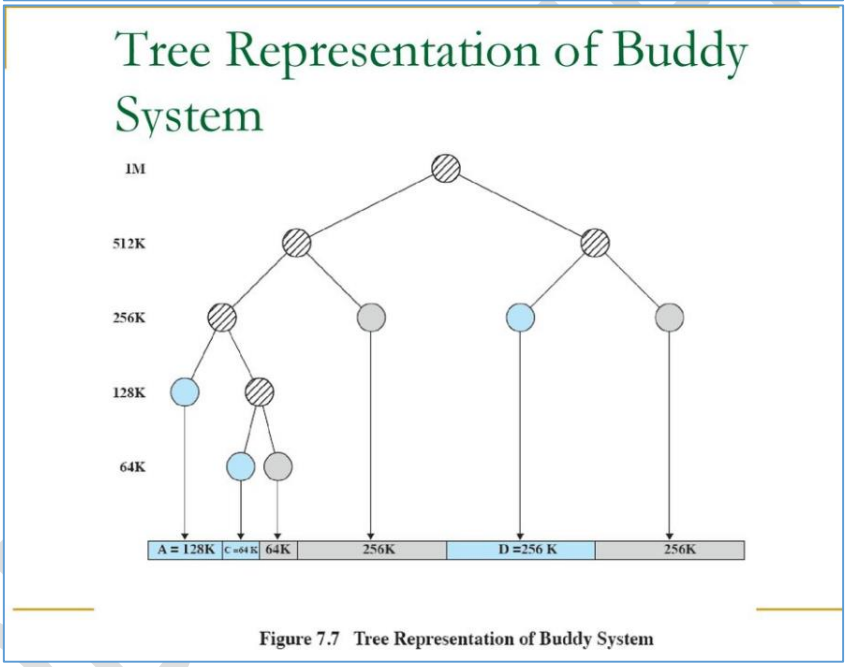



Figure 7.7 Tree Representation of Buddy System

**16. File Management.**

a. Users and application interact with file system
b. Issue commands like create, delete, read / write on the files
c. System must identify and locate required file
d. Done by directory having location of file and attributes
e. Some systems enforce access control
f. Authorised users can access particular files in particular ways
g. Basic operations by user done on record level of file
h. User / application views file as having some structure for organising records
i. Ex. Sequential structure (ascending order of names based on last names)
j. Translating user commands (file manipulation) requires access method suitable to file structure
k. Users / application concerned with records / fields
l. I / O is block based; records / fields organised as sequence of blocks; blocked for output; unblocked after input
m. Supporting I / O files requires several functions; secondary storage management

n. Allocate files to free blocks in SS, and manage free storage for new files
o. Block I / O requests scheduled; disk scheduling and file allocation required for optimising performance
p. Optimisation depends on structure of file and access pattern
q. Designing optimum file management system is difficult task
r. One method – file management concerns different; OS concerns different; intersection of *record processing*
s. Other various approaches are there for various systems

**17. Gang scheduling.**

a. Simultaneous scheduling of threads of same process
b. Useful for medium-grained to fine-grained parallel applications
c. Performance degrades; part of application is not running; others parts ready
d. Process switches are minimised
e. Ex. One thread needs to synchronise with other thread; other thread not running; in the ready queue; process switch brings other thread on some other processor; first thread is hung up
f. Tight coordination is required
g. Process switches reduce performance
h. Also saves time in resource allocation
i. Multiple scheduled threads access same file without overhead of locking seek / read / write operation
j. **Advantages.**
    i. Synchronisation blocking is reduced
    ii. Increased performance
    iii. Reduced scheduling overhead
    iv. Less process switching necessary

**18. Buffer Overflows.**

a. Condition where more input in placed than its data – holding capacity
b. It overwrites other information
c. Can occur as result of programming error
d. Where more data gets stored than capacity
e. Leads to overwriting adjacent memory locations
f. These locations can hold other program variables, program control data, return addresses, pointers etc.
g. Buffer could be in stack, on heap or data section of process
h. **Results of buffer overflow.**
    i. Corruption of data
    ii. Loss of data
    iii. Unexpected transfer control in program
    iv. Memory access violations
    v. Termination of program
i. Most prevalent and dangerous types of security attacks
j. **Attackers on buffer overflow.**
    i. Use it to crash a system
    ii. Gains control of system
    iii. Inserts specially crafted code to destroy system
    iv. This code can execute any other code

**19. Process Control Block – elements and role.**

a. Data structure in OS kernel
b. Contains all information needed to manage scheduling of particular process
c. *Central* role in process management
d. Access and modified by OS utilities
e. Defines current state of the OS
f. **Elements of PCB.**
    i. **Scheduling and State information.**

1. Information needed to perform scheduling function
2. **Typical Items of Information.**
    a. Process state – readiness of the process which is supposed to be scheduled for execution (running, halted, ready, waiting)
    b. Priority – used to describe scheduling priority of process; some systems require several values (default, current)
    c. Scheduling related information – depends on scheduling algorithm (time of wait of process, time taken for execution the last time)
    d. Event – identity of the event for which the process has been waiting to be resumed

ii. **Data Structure.**
1. Processes are linked to each other
2. May have structures like queue, ring etc.
3. PCB may have pointers to other processes to support the structure

iii. **Inter-process Communication.**
1. Flags, signals / messages may be associated between 2 independent processes
2. Some / all of the information maintained in PCB

iv. **Process privileges.**
1. Granted in terms of memory
2. Privileges apply to use of system utilities and services

v. **Memory Management.**
1. Contains pointers to segments or pages
2. Describes virtual memory
3. Assigned to the process

vi. **Resource Ownership and utilisation.**
1. Resources controlled are indicated (ex. Opened files)
2. History of utilisation of resources may also be included
3. This information may be needed by scheduler

g. Most important data structure in OS
h. Blocks read / modified by virtually every block in OS
i. Protection is difficult
j. Access is easy
k. Bug in single routine damages PCB; can destroy the system's ability to manage affected processes
l. Change in design / semantics / structure could affect the modules of OS

**20. Amdahl's Law.**

a. Improvement in technology
b. Change in design
c. Factors in improving system performance
d. Done by – parallel processors, use of memory cache, speed up in memory access time
e. Speeding up is not completely related to improvement in performance
f. This limitation is explained by Amdahl's law
g. **Consider.**
    i. Single processor
    ii. (1-f) fraction of time – inherently serial
    iii. (f) fraction of time is infinitely parallelisable with no scheduling overhead
    iv. T is total time
    v. Speedup using parallel processor have N processors (fully exploits parallel portion of program) is given by: (**please refer internet)**
h. **Conclusions.**
    i. If "f" is small then use of parallel processors has very little effect
    ii. As "N" tends to infinity, speedup is bound by 1/(1-f); diminishing returns for using more processors
    iii. " Best case scenario is to use *High "f" and High "N"* "

**21. Inverted Page Table and its structure.**

a. Global page table
b. Maintained by OS
c. For all processes
d. Number of entries = number of frames in main memory
e. Overcomes drawbacks of page table
f. Saves details of pages present in the main memory
g. Frames are indices
h. Inside the block - <Pid><Pno>
i. Each entry corresponds to page frame instead of page number
j. Fixed portion of main memory is required
k. Irrespective of the number of processes
l. Chaining technique used to manage overflow
m. Page number portion of the virtual address is mapped into a hash value using a simple hashing function
n. Hash value is pointer to inverted page table containing the page table entries
o. Hashing technique results in chains which are typically short – one and two entries
p. **Page number.**
    i. Page number portion of the virtual address
q. **Process number.**
    i. Process that owns this page
r. *Combination of page number and process number identifies a page within virtual address space of particular process*
s. **Control bits.**
    i. Includes flags
    ii. Valid, referenced, modified
    iii. Protection and locking information
t. **Chain pointer.**
    i. Null if no chained entries for this entry
    ii. Otherwise contains index value (number between 0 and 2m-1) of next entry in chain