INDEX

- Dynamic Memory Management Functions
- Fixed Block Storage
- Variable Block Storage
- Data Placement Algorithms
  - First fit, Next fit, Best fit, Worst fit
- Compaction
- Garbage Collection

DYNAMIC MEMORY ALLOCATION

The process of allocating memory at run time is known as dynamic memory allocation. Although c does not inherently have this facility statically , but there are four library routines which allow this function.

Function	Task
malloc	Allocates memory requests size of bytes and returns a pointer to the lst byte of allocated space
calloc	Allocates space for an array of elements initializes them to zero and returns a pointer to the memory
free	Frees previously allocated space
realloc	Modifies the size of previously allocated space.

MEMORY ALLOCATIONS PROCESS

In Main memory program instructions , global and static variable are stored in a permanent storage area and local area variables are stored in stacks.

The memory space that is located between these two regions is available for dynamic allocation during the execution of the program. The free memory region is called the heap.

The size of heap keeps changing when program is executed due to creation and death of variables that are local for functions and blocks. Therefore it is possible to encounter memory overflow during dynamic allocation process

FIXED BLOCK STORAGE

- Fixed size blocks are allocated to the processes as and when required.
- Problem – of internal fragmentation.

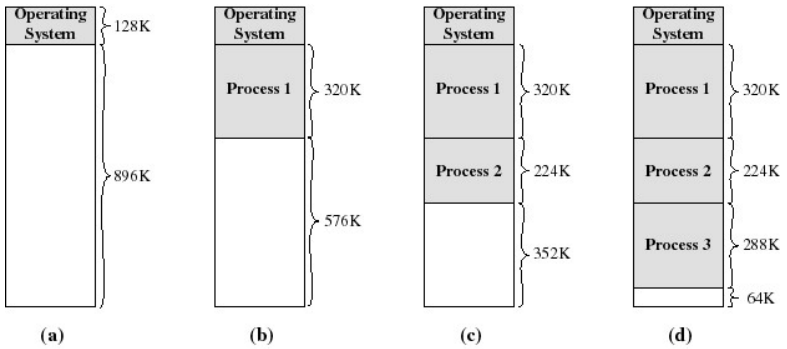
VARIABLE PARTITIONING

- With fixed partitions we have to deal with the problem of determining the number and sizes of partitions to minimize internal fragmentation.
- If we use variable partitioning instead, then partition sizes may vary dynamically.
- In the variable partitioning method, we keep a table (linked list) indicating used/free areas in memory.

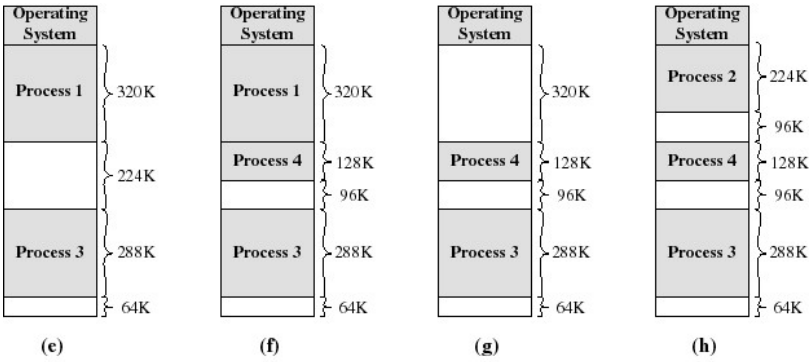
VARIABLE PARTITIONING

- Initially, the whole memory is free and it is considered as one large block.
- When a new process arrives, the OS searches for a block of free memory large enough for that process.
- We keep the rest available (free) for the future processes.
- If a block becomes free, then the OS tries to merge it with its neighbors if they are also free.

DYNAMIC PARTITIONING: AN EXAMPLE



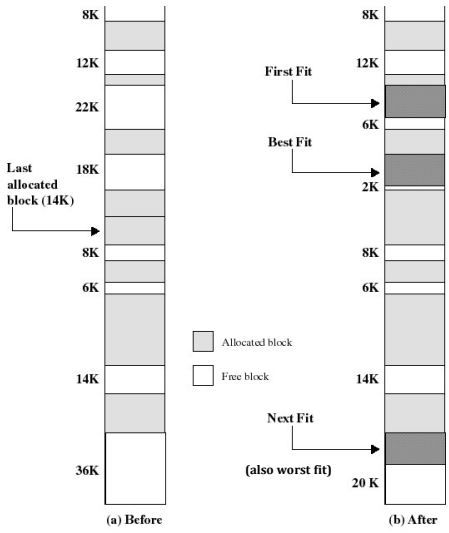
DYNAMIC PARTITIONING: AN EXAMPLE



Problem of External fragmentation

PLACEMENT ALGORITHM

- Decide which free block to allocate to a program
- Possible algorithms:
  - First-fit: choose first hole from beginning that fits
    - generally superior
    - Next-fit: variation: first hole from *last placement* that fits
  - Best-fit: choose smallest hole that fits
    - creates small holes!
  - Worst-fit: use largest hole
    - leaves largest leftover



Example Memory Configuration Before and After Allocation of 16 Kbyte Block

DYNAMIC PARTITIONING PLACEMENT ALGORITHM

- First-fit algorithm
  - Fastest
  - Chooses the block that is big enough size to the request
  - Begins to scan the memory from starting of memory

Dynamic Partitioning Placement Algorithm

- Best-fit algorithm
  - Chooses the block that is closest in size to the request
  - Worst performer overall
  - Since smallest block is found for process, the smallest fragment is left
  - Memory compaction must be done more often

Dynamic Partitioning Placement Algorithm

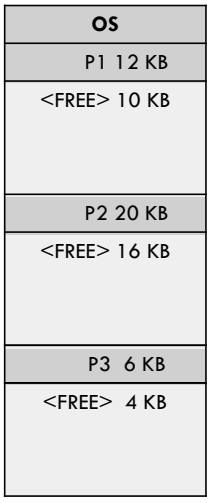
- Next-fit
  - Begins to scan the memory from the location of the last placement & chooses the next available block that is large enough
  - More often leads to allocation of a block at the end of the memory
  - Results in quick fragmentation of the largest block of free memory which usually appears at the end of the memory
  - Compaction is required to obtain a large block at the end of memory

Dynamic Partitioning Placement Algorithm

- Worst-fit
  - Scans the whole memory to find the biggest block
  - Holes are of substantial size
  - Less fragmentation

FIRST FIT

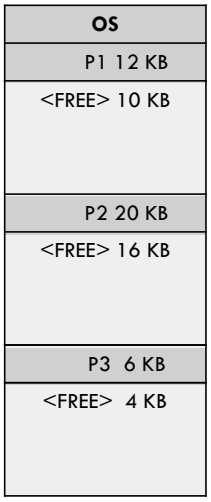
Initial memory mapping



15

FIRST FIT

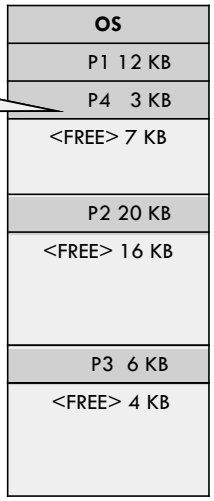
P4 of 3KB arrives



16

FIRST FIT

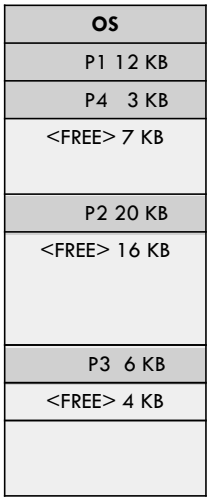
P4 of 3KB loaded here by FIRST FIT



17

FIRST FIT

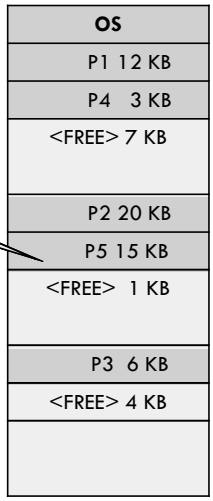
P5 of 15KB arrives



18

FIRST FIT

P5 of 15 KB loaded here by FIRST FIT



19

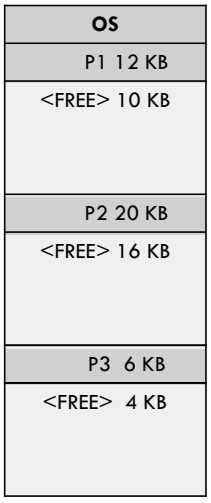
BEST FIT

- Best Fit : Allocate the smallest block among those that are large enough for the new process.
- In this method, the OS has to search the entire list, or it can keep it sorted and stop when it hits an entry which has a size larger than the size of new process.
- This algorithm produces the smallest left over block.
- However, it requires more time for searching all the list or sorting it

20

BEST FIT

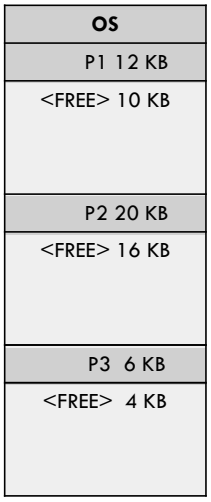
Initial memory mapping



21

BEST FIT

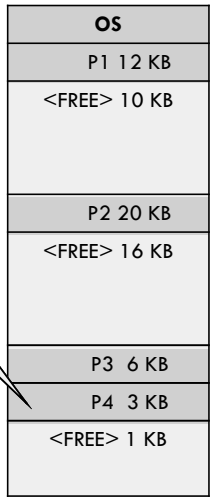
P4 of 3KB arrives



22

BEST FIT

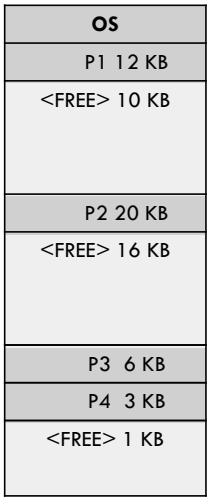
P4 of 3KB loaded here by BEST FIT



23

BEST FIT

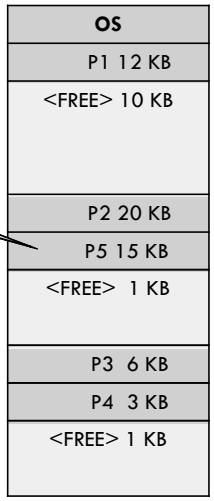
P5 of 15KB arrives



24

BEST FIT

P5 of 15 KB loaded here by BEST FIT



25

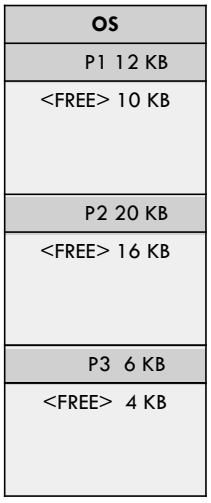
WORST FIT

- Worst Fit : Allocate the largest block among those that are large enough for the new process.
- Again a search of the entire list or sorting it is needed.
- This algorithm produces the largest over block.

26

WORST FIT

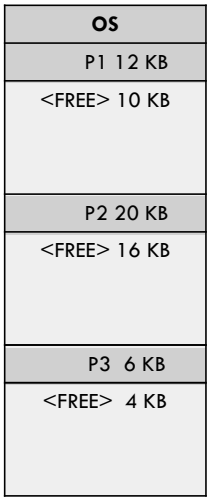
Initial memory mapping



27

WORST FIT

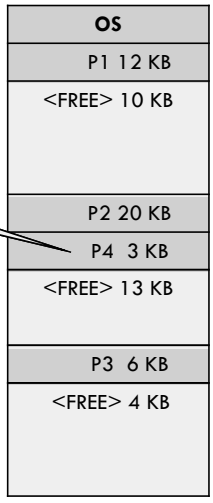
P4 of 3KB arrives



28

WORST FIT

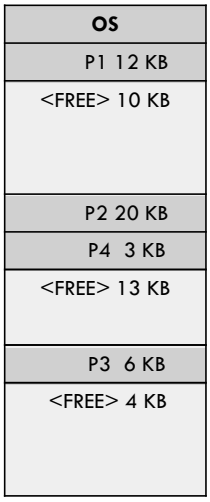
P4 of 3KB Loaded here by WORST FIT



29

WORST FIT

No place to load P5 of 15K

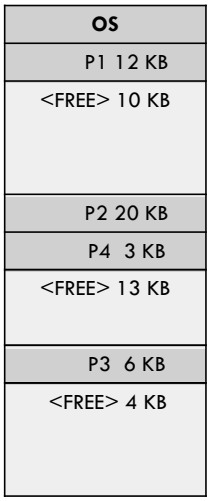


30

WORST FIT

No place to load P5 of 15K

Compaction is needed !!



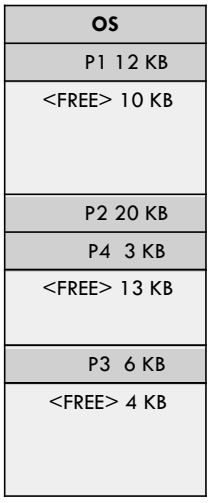
31

COMPACTION

- External fragmentation can be resolved through compaction
- Shuffles the memory contents to put all free memory together in one block
- Compaction algorithm is expensive, but so is not making efficient use of memory, especially with a lot of concurrent processes
- Only possible if relocation is dynamic
  - Relocation requires moving program and data, changing the base register

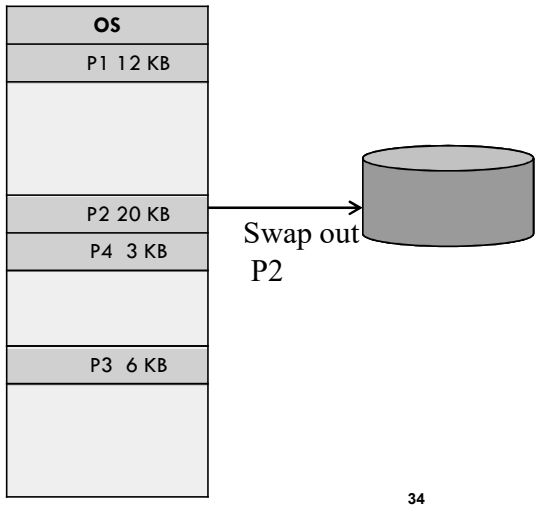
COMPACTION

Memory mapping before compaction



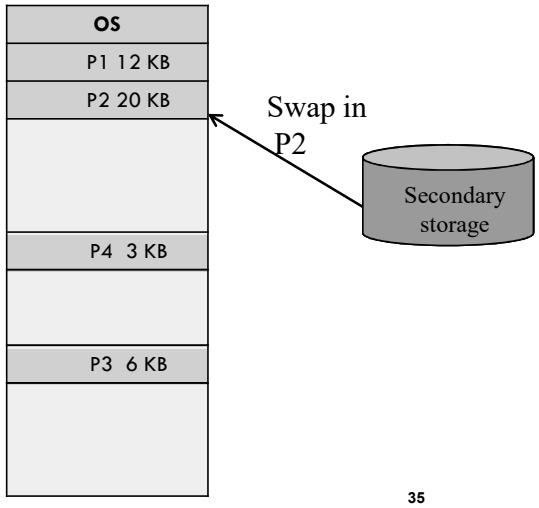
33

COMPACTION



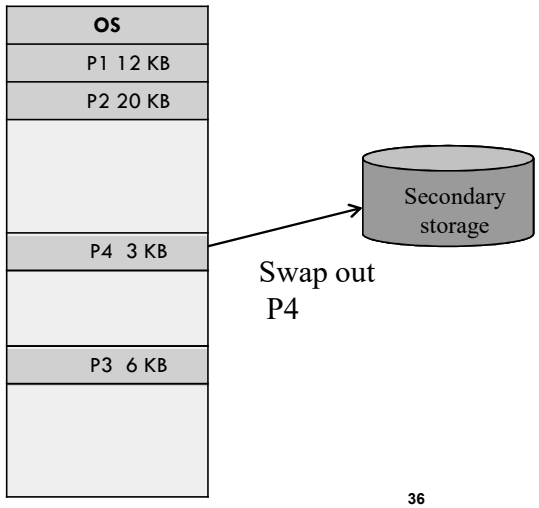
34

COMPACTION



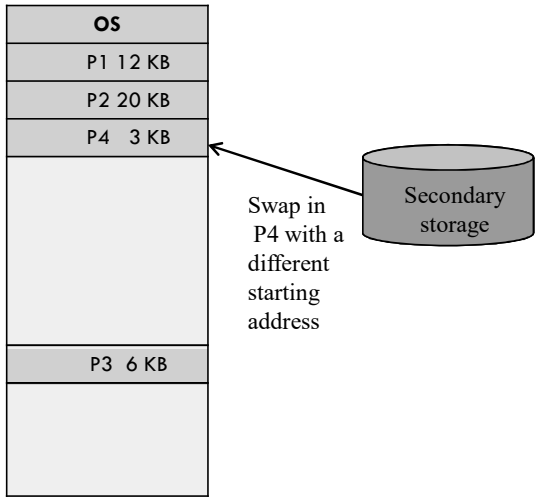
35

COMPACTION

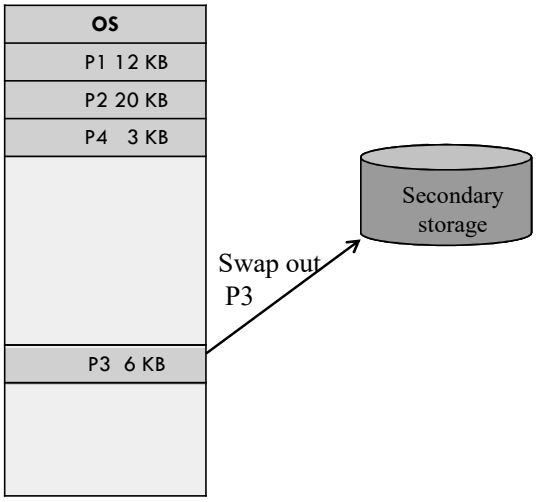


36

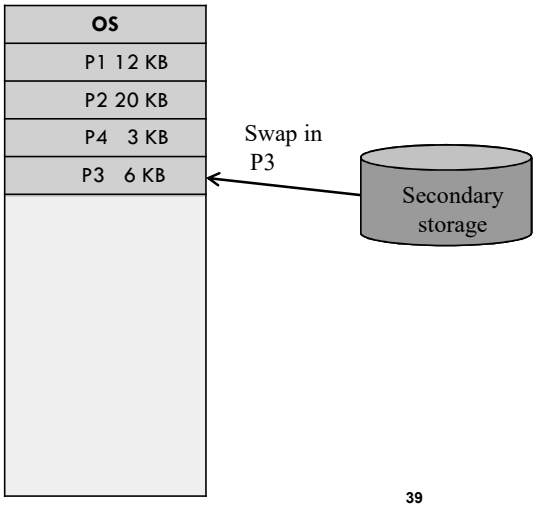
COMPACTION



COMPACTION

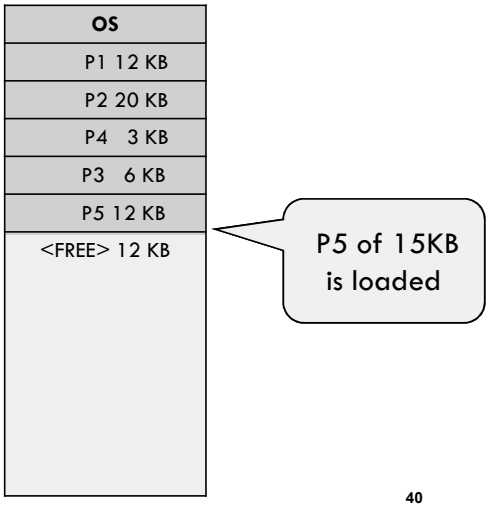


COMPACTION



39

COMPACTION



40

GARBAGE COLLECTION

Can be done by OS by two different methods

- Reference counts
- Mark and sweep

REFERENCE COUNT METHOD

In this method additional field count (reference count) is kept with each node, when pointer (internal / external) pointing to a particular node then count will be increase by 1. and when this value will be changed then count will decrease by 1. When reference count of any node becomes 0 that node can be return to the available list of free nodes.

The drawback of reference count method is amount of work must be performed each time list manipulation work is done.

Another disadvantage is to kept one additional field with each node.

MARK AND SWEEP

In this method garbage collection is done in two phases marking phase, in this all nodes accessible from an external node are marked for that either additional field is kept that have been marked or separate area in memory can be reserved to hold a long array of mark bits, one bit for each node has been allocated.

In collection / swap phase free all the nodes that have not been marked and also all marked fields are set to false. So at the end of garbage collector program all fields again set to unmarked.

CONTINUE…..

- Whenever the garbage collector is called all user processing is halt while algorithm examine all allocated node memory.
- For this purpose it is desirable to call G.C. infrequently as possible.
- The phenomenon, is which system G.C. (storage management routine) is executing almost all time is called **thrashing.** Thrashing is situation to be avoid.