Memory management by variable partitions

Variable partitions are similar to fixed partitions, but *number and sizes* of partitions change dynamically.

When a process is brought into main memory, it is allocated exactly as much memory as it requires. After this memory is deallocated, splitting and merging of partitions are possible.

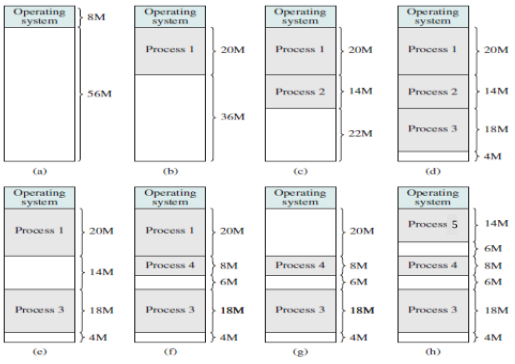
Slide 1

In this technique also, a memory partition of enough size is reserved for each process from its start to its termination. However, the partitions into which the memory is divided change dynamically.

Slide 2

Process is assigned the exact memory size it needs. If this is taken from a previously larger partition, the remainder of this partition can be used by some other process. Adjacent partitions that become free can be merged into one larger partition. These variable partitions enable avoiding the disadvantages of fixed partitions.

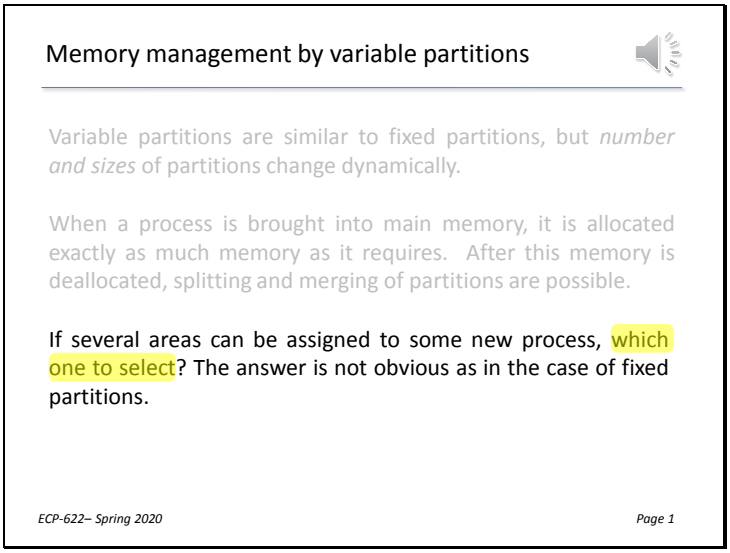
Memory management by variable partitions



Slide 3

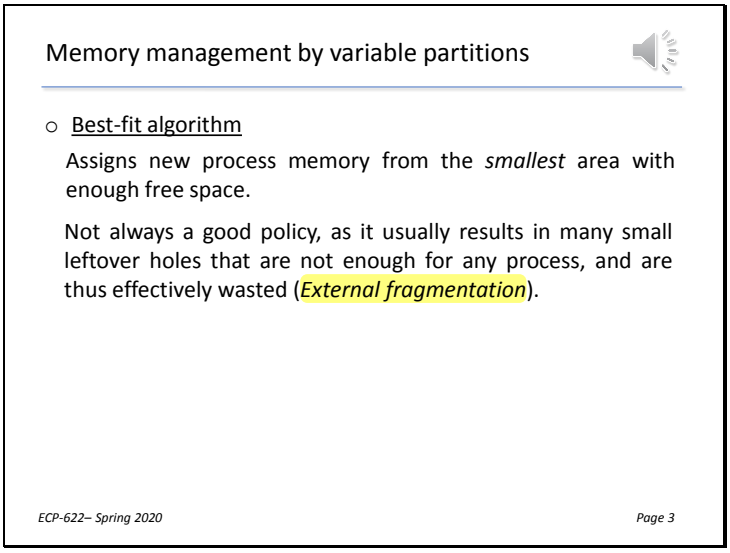
In figure (a), the operating system only is running, reserving a memory partition for itself. In figures (b),(c), and (d) processes 1,2, and 3 are assigned their memory requirements. In figure (e) process 2 terminates, thus releasing the partition it held. When process 4 starts in

figure (f), it needs only 8M, thus the partition of size 14 M is split into 8M assigned to new process, and 6M that remains free. Same occurs when process 5 starts.



Slide 4

In fixed partitions, the answer to this question was to select the smallest, to minimize the waste in memory. Here, no waste occurs as a result of internal fragmentation. So what is the best selection?

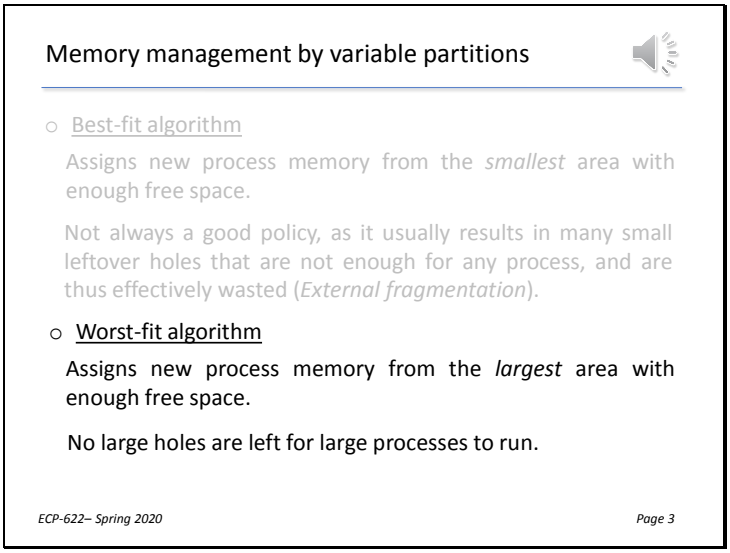


Slide 5

If we also take the memory requirements from the smallest partition, we call this the best-fit selection.

Slide 6

This tends to minimize the remaining free areas. These remaining areas may be not enough to run any process. Note that we still assume that a process needs to operate in one undivided partition. After using best fit for some time, the free area in memory will be scattered into small holes that are not enough to run processes. This phenomenon is called external fragmentation (i.e. fragments occur outside of partitions).

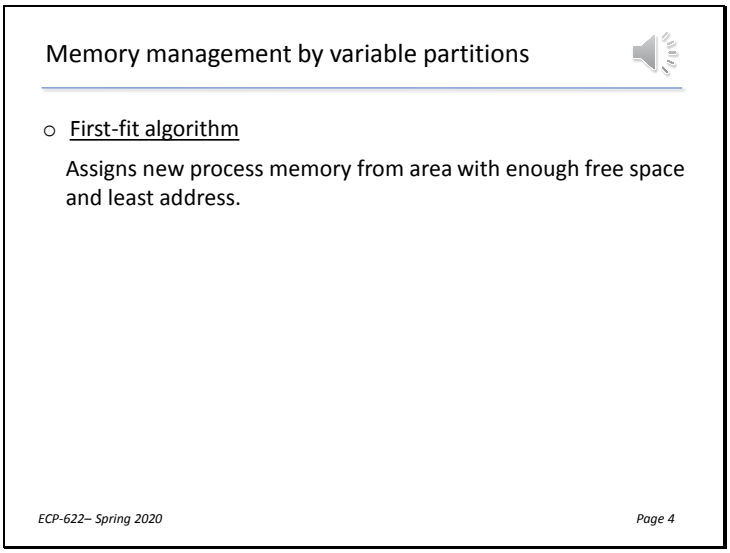


Slide 7

If we select instead the largest partition, we call this the worst-fit algorithm. It should avoid the disadvantage of best-fit.

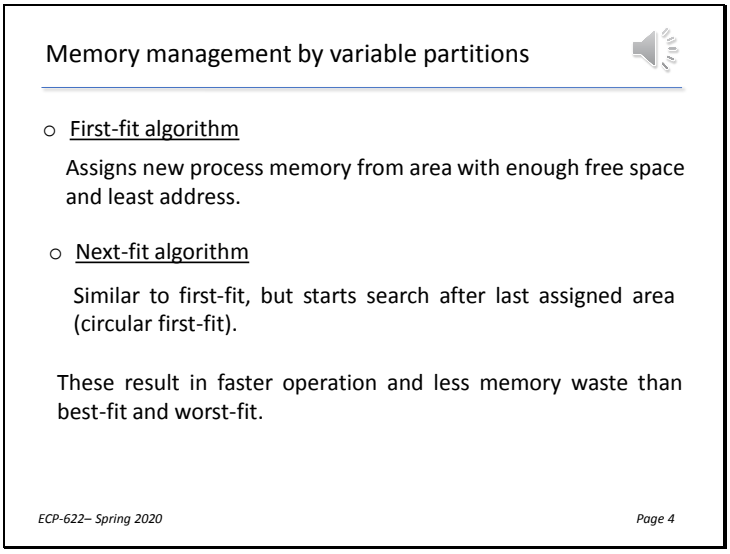
Slide 8

However, a new problem arises. Since in worst-fit we systematically split large partitions, we expect that no large partitions will be left after a while. Processes with large memory requirements will not be able to run.



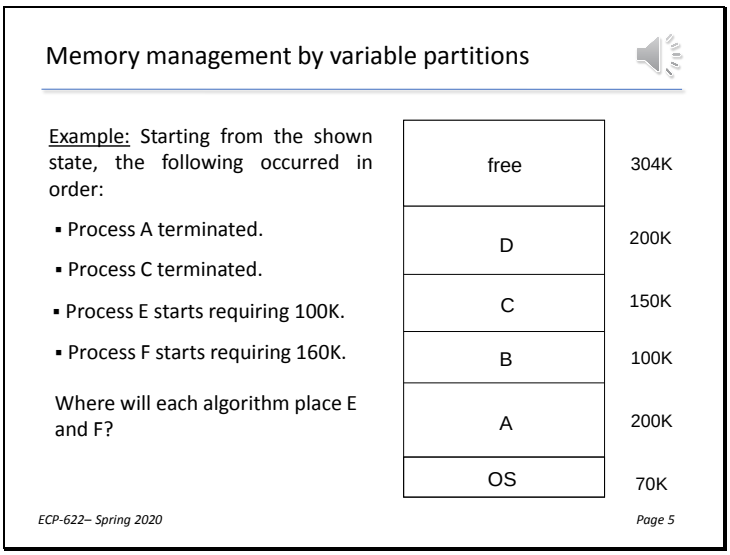
Slide 9

Since selection of smallest or largest partitions will both lead to problems, it is better not to base decision on size. The first-fit algorithm begins looking for a suitable partition from the start of memory, and assigns the first partition it finds. Sometimes this will be small, and sometimes it will be large, so disadvantages of best-fit and worst-fit will not persist.



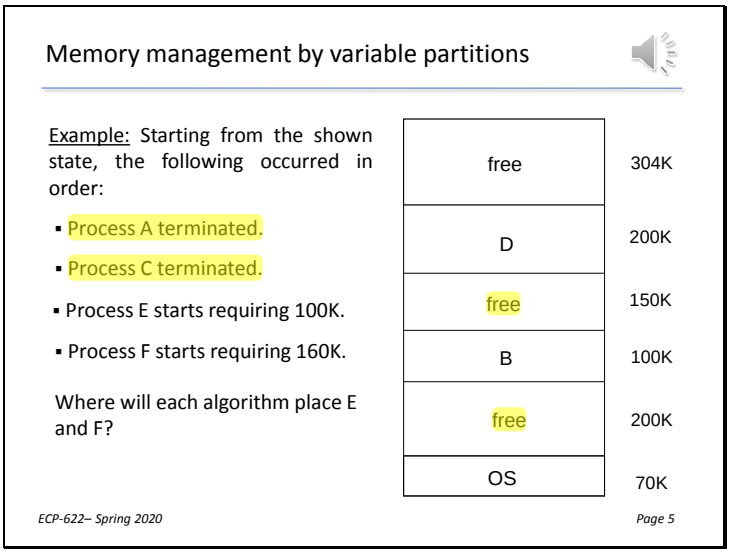
Slide 10

A slightly different method is the next-fit algorithm. It also assigns the first partition it finds irrespective of size. However, it does not begin the search from the start of memory each time. Instead, it begins search after the last assigned area. This usually will speed up the search, as in first-fit the start of memory will typically be fragmented, and suitable partitions will be found at higher addresses. The name circular first-fit may be used since when search reaches the end of memory, it returns back to the start.



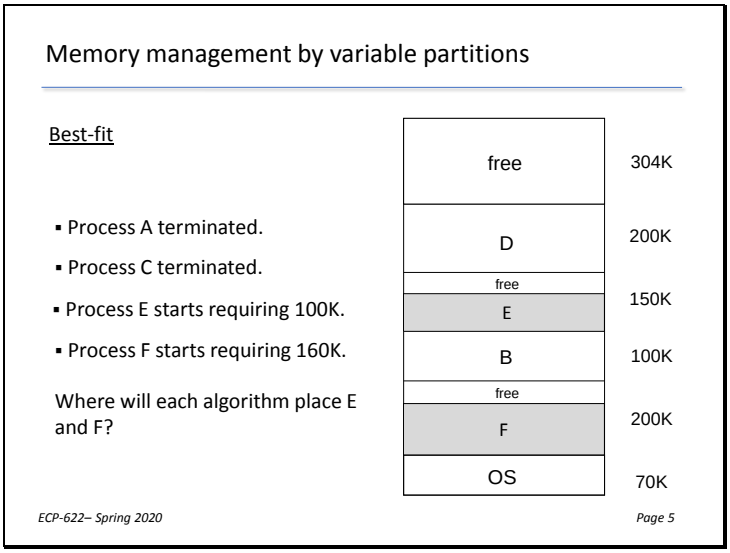
Slide 11

Note here that the order of events is important as it will change the result.



Slide 12

Now we have free areas of sizes 200 K, 150 K, and 304 K.

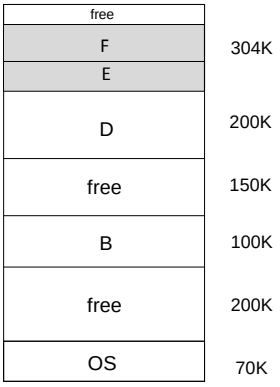


Memory management by variable partitions

Worst-fit

- Process A terminated.
- Process C terminated.
- Process E starts requiring 100K.
- Process F starts requiring 160K.

Where will each algorithm place E and F?

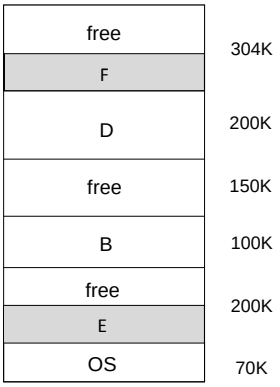


Memory management by variable partitions

First-fit assuming that addresses increase upwards

- Process A terminated.
- Process C terminated.
- Process E starts requiring 100K.
- Process F starts requiring 160K.

Where will each algorithm place E and F?

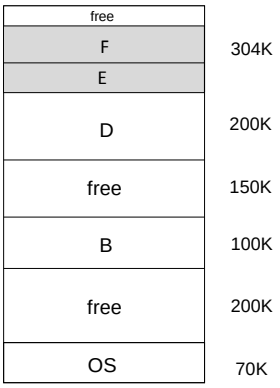


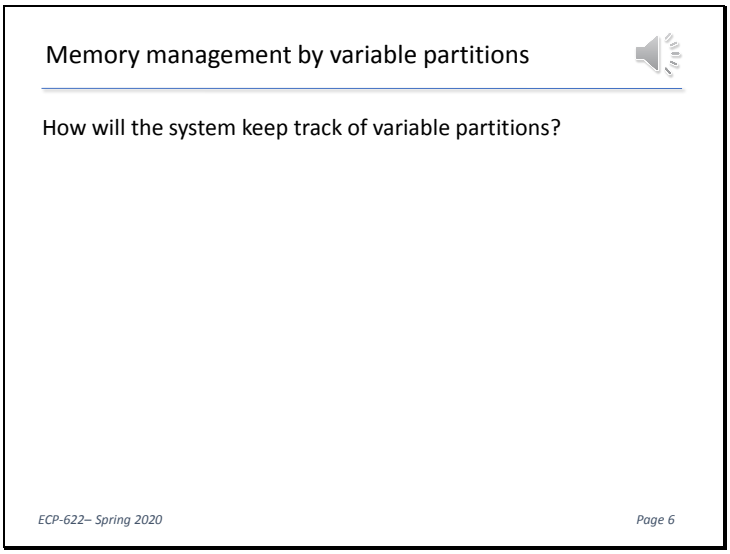
Memory management by variable partitions

Next-fit assuming that D was the process last assigned

- Process A terminated.
- Process C terminated.
- Process E starts requiring 100K.
- Process F starts requiring 160K.

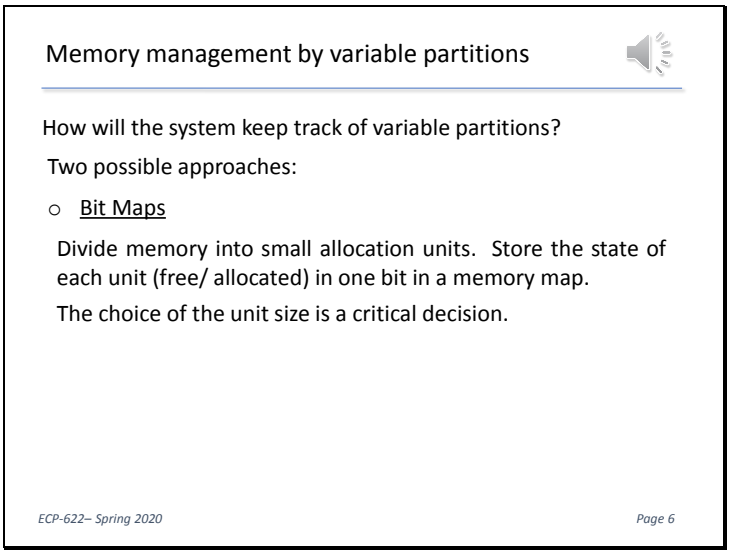
Where will each algorithm place E and F?





Slide 17

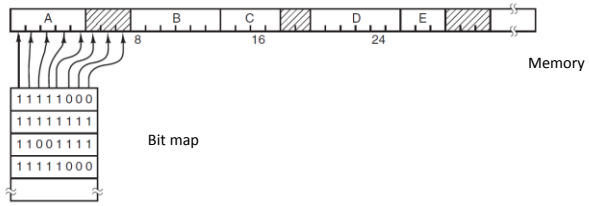
In fixed partitions, the system needs only to maintain a fixed table with start and end addresses of partitions. When partitions vary dynamically, a more complex data structure is needed.



Slide 18

Here, the system will maintain a map of memory indicating free and assigned areas. If this map shows the status of each address (byte), its size will be too large, taking itself a large portion of memory. Thus, map deals with larger units (blocks of addresses) to make its size more compact. Process is assigned an integer number of these allocation units, as map does not handle fractions of units. Unit should not be too large, as process may waste memory if assigned a large unit not used completely.

Memory management by variable partitions



Source: [Tanenbaum 15]

Slide 19

Note here how memory is divided into units, with a bit in map corresponding to each unit. Process A is assigned the first five units. This is reflected in map by five 1 bits. There is next three free units shown in map by three zero bits,… etc. A search of a free partition with a given size will be a search for a specific number of adjacent zero bits in map.

Memory management by variable partitions

How will the system keep track of variable partitions?

Two possible approaches:

- Bit Maps

  Divide memory into small allocation units. Store the state of each unit (free/ allocated) in one bit in a memory map.

  The choice of the unit size is a critical decision.

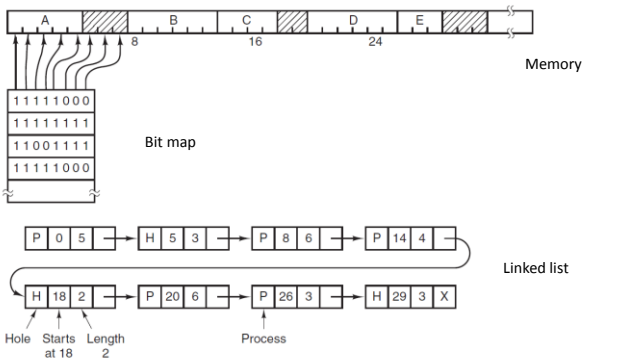
- Linked Lists

  Maintain a linked list with a node for each memory area, whether free or assigned. Update list whenever processes start or terminate.

Slide 20

In this alternative method system uses a linked list with a node for every memory partition, whether free or assigned. As partitions are created, split, and merged, the linked list is updated to reflect these changes.

Memory management by variable partitions



Source: [Tanenbaum 15]

Slide 21

Linked list has a node for each area assigned to process (P) or free or hole (H). Node indicates length of partition and where it starts. Note that units are not necessary for the linked list method and used here for illustration only. If for example process B terminates, the area it occupies will be merged with the preceding hole, forming a single hole of length 9.

Memory management by variable partitions

The previous algorithms do not scale well as search time increases with memory size. Some algorithms provide better search time

- Segregated free lists

Use a separate list for free partitions within a given size range.

- Indexed-Fit

Use a data structure (e.g. ordered binary tree) to store available partition sizes. For each size, a pointer is used to access a list of the available partitions.

Buddy Allocator System

Allocates memory partitions of size 2^K with $L \leq K \leq U$, where 2^L is the smallest partition size and 2^U is the size of available memory. A list is maintained for free partitions of each size 2^i.

For a request for memory block of size s, system assigns a partition of size 2^n, where $2^{n-1} < s \leq 2^n$. If necessary, system splits larger partitions in halves.

When a partition becomes free it can be merged with an adjacent partition of the same size.

Slide 23

Buddy allocator method combines some advantages of fixed and variable partitions. Possible partition sizes are limited to powers of 2. A separate list is maintained for free partitions of each size of the finite possible ones. This will speed up the search for an area of particular size.

Slide 24

Process is assigned partition with a size the nearest power of 2 to its requirements (some internal fragmentation thus occurs). Partitions can be split in two, and two adjacent free partitions of the same size can be merged (thus maintaining the power of 2 restrictions).

Buddy Allocator System

1-Mbyte block	1M				
Request 100K	A = 128K	128K	256K	512K	
Request 240K	A = 128K	128K	B = 256K	512K	
Request 64K	A = 128K	C = 64K	64K	B = 256K	512K
Request 256K	A = 128K	C = 64K	64K	B = 256K	D = 256K	256K
Release B	A = 128K	C = 64K	64K	256K	D = 256K	256K
Release A	128K	C = 64K	64K	256K	D = 256K	256K
Request 75K	E = 128K	C = 64K	64K	256K	D = 256K	256K
Release C	E = 128K	128K	256K	D = 256K	256K
Release E	512K			D = 256K	256K
Release D	1M				

Source: [Stallings 14]

Slide 25

In this example, memory is first consisting of a single free partition of size 2 power 10.

Slide 26

Process A starts requiring 100 K. It will be assigned the nearest larger power of 2, which is 128 K. This is obtained by successive splitting of partitions. Now memory consists of 128 K assigned to A, and three free partitions of sizes 128, 256, and 512 K.

Slide 30

B releases its memory, but no merging occurs yet as sizes are different.

Slide 33

When C ends, merging occurs since two adjacent free partitions have the same size.