Memory Management

Reading:Section 4.1 & 4.2 from Textbook(Tanenbaum)

The entire program and data of a process must be in main memory for the process to execute.

How to keep the track of processes currently being executed?
Which processes to load when memory space is available?
How to load the processes that are larger than main memory?
How do processes share the main memory?

OS component that is responsible for handling these issues is a *memory manager*.

Memory Management Issues

- Uniprogrammnig or Multiprogramming system.
- Absolute or relocatable partitions.
- Multiple fixed or multiple variable partitions.
- Partition in rigid manner or dynamic.
- Program run from specific partition or anywhere.
- Job placing at contiguous block or any available slot.

Memory Management Requirements

Relocation

- Programmer does not know where the program will be placed in memory when it is executed
- While the program is executing, it may be swapped to disk and returned to main memory at a different location (relocated)
- Memory addresses must be translated in the code to actual physical memory address.

Hardwares used - Base (relocatable) register, limit register

Memory Management Requirements

Protection

Prevent processes from interfering with the OS or other processes.

Processes should not be able to reference memory locations in another process without permission.

Impossible to check absolute addresses in programs since the program could be relocated.

Often integrated with relocation.

Sharing

Allow several processes to access the same portion of memory (allow to share data).

Better to allow each process (person) access to the same copy of the program rather than have their own separate copy.

Logical vs. Physical Memory Address

Logical Address: The address generated by CPU.
Also known as virtual address.

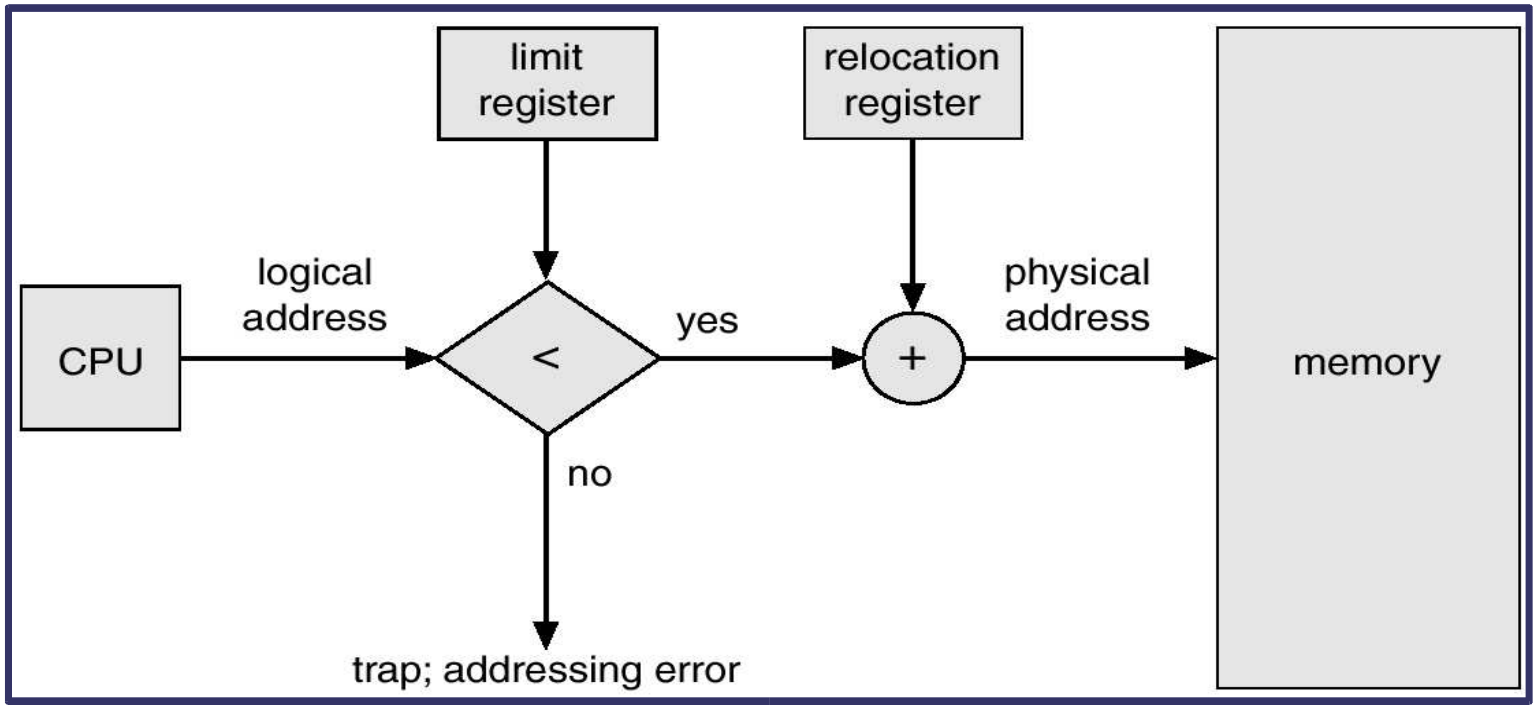
Physical Address: Actual address loaded into the memory address register.

Mapping from logical address to physical address is relocation.

Two registers: *Base* and *Limit* are used in mapping.

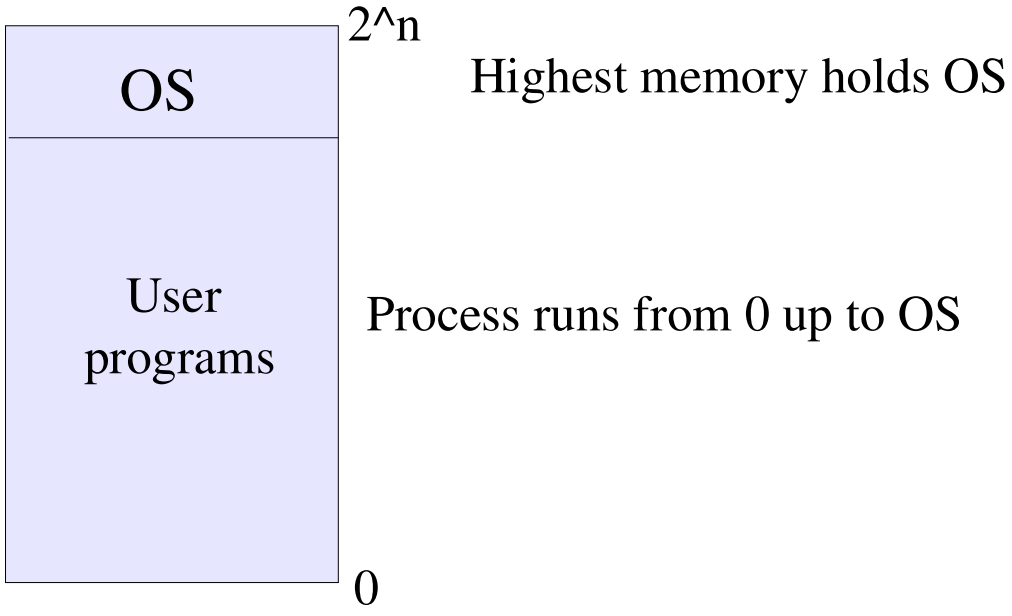
This mechanism also used in memory protection – memory outside the range are protected.

Logical to Physical Address



Base register (relocation): Holds smallest legal physical memory address.
Limit register: contain the size of range.
Example:
logical address = 300
if Base = 1500
physical address = 1500 + 300 = 1800.

Uniprogramming Model



Run one program at a time with single segment per process. Memory space is divided into two partitions by convention; one partition is allocated to OS and the other to executing process.

Uniprogramming Model

As soon as the user types the command, the OS copies the requested program from disk to memory and execute it.
Used in early Dos system.

Disadvantages:

- Only one process can run at a time.
- Processes can destroy OS (an erroneous process can crash the entire system).

Multiprogramming Model

Fixed Partition Multiprogramming

Multiple Partitions are created to allow multiple user processes to resident in memory simultaneously.

The partitions are fixed (possibly unequal), can be done using *base* and *limit* register.

Partition table stores the partition information for a process.

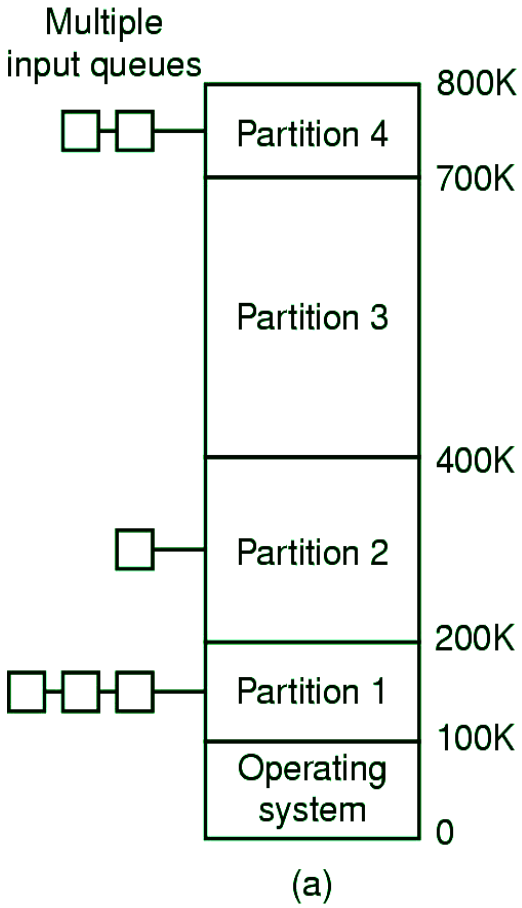
When job arrives, it can be put into the input queue for the smallest partition large enough to hold it. Science the partition are fixed, any space in partition not used by a process is lost.

Used by OS/360 on large mainframes.

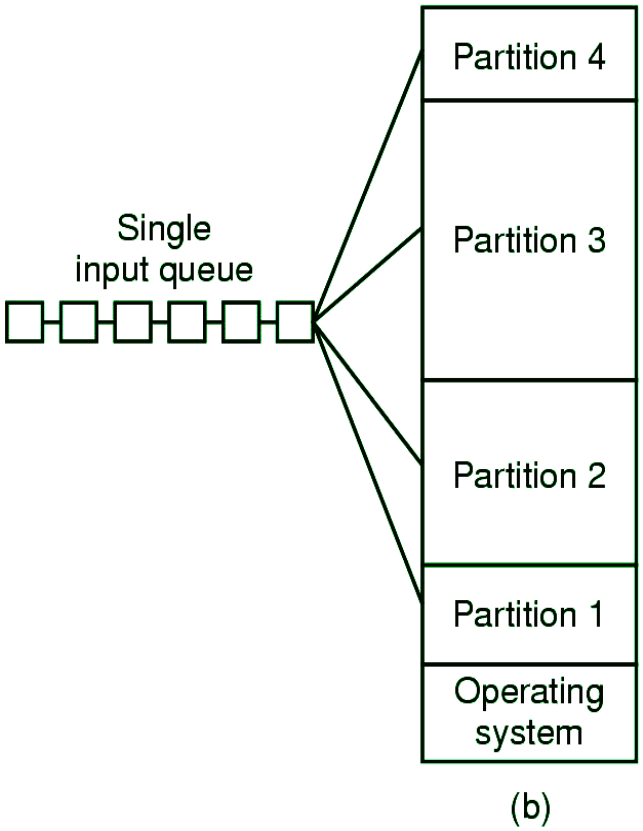
Implemented in two ways :

- Dedicate partitions for each process (Absolute translation).
- Maintaining a single queue (Relocatable translation).

Multiprogramming Model
Fixed Partition Multiprogramming



Absolute Translation



Relocatable Translation

Multiprogramming Model
Fixed Partition Multiprogramming

Dedicating Partitions

Separate input queue is maintained for each partition.

Processes are translated with absolute assemblers and compilers to run only in specific partition.

Problems:

if a process is ready to run and its partition is occupied then that process has to wait even if other partitions are available – *wastage of storage*.

Multiprogramming Model
Fixed Partition Multiprogramming

Maintaining Single Queue

Maintains a single queue for all processes.

When a partitions becomes free, the processes closest to the front of queue that fits in it could be loaded into the empty partition and run. Not to waste the large partition for small process, another strategy to search the whole input queue whenever a partition becomes free and pick the largest process that fits.

Problems:

Eliminate the absolute problems but implementation is complex.
Wastage of storage when many processes are small.

Read yourself: Section 4.1.3 and 4.1.4 (Tanenbaum)

Variable Partition Multiprogramming

How to allocate the space for a process as much as they need?

-variable partition multiprogramming.

When processes arrive, they are given as much storage as they need.

When processes finish, they leave holes in main memory; OS fills these holes with another processes from the input queue of processes.

Advantages:

Processes get the exact size partition of their request for memory – *no waste of memory.*

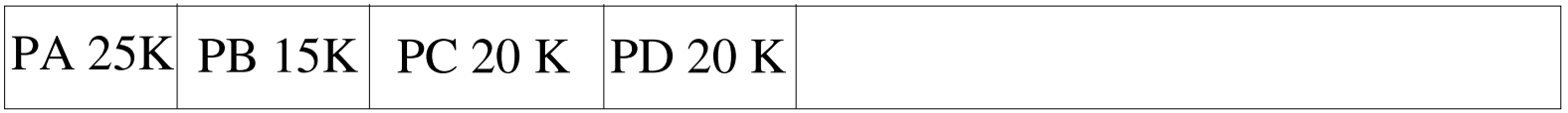
Problems:

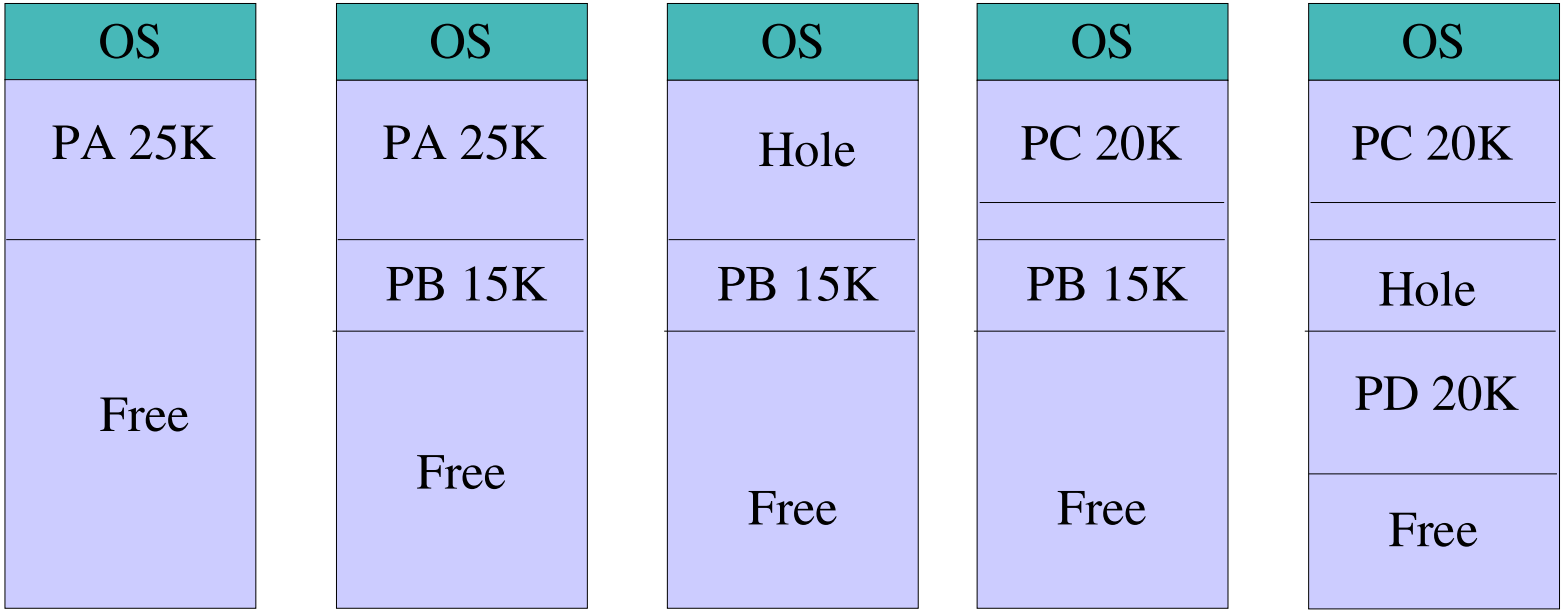
What will happen when we leave hear ???

When holes given to other process they may again partitioned, the remaining holes gets smaller eventually becoming too small to hold new processes – waste of memory occur.

Variable Partition Multiprogramming

Input queue





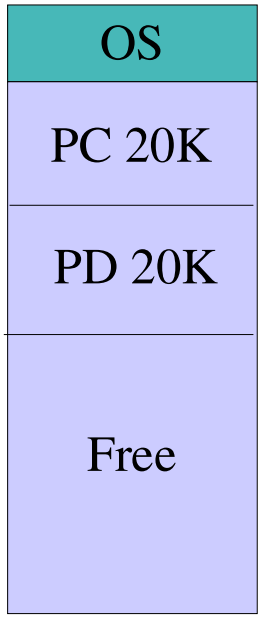
Memory allocation and holes

Solutions???

Variable Partition Multiprogramming

Memory Compaction

By moving processes in memory, the memory holes can be collected into a single section of unallocated space – memory compaction.



Memory compaction

Problems:

Not possible in absolute translation.

If it possible, it is highly expensive – it requires a lots of CPU time;

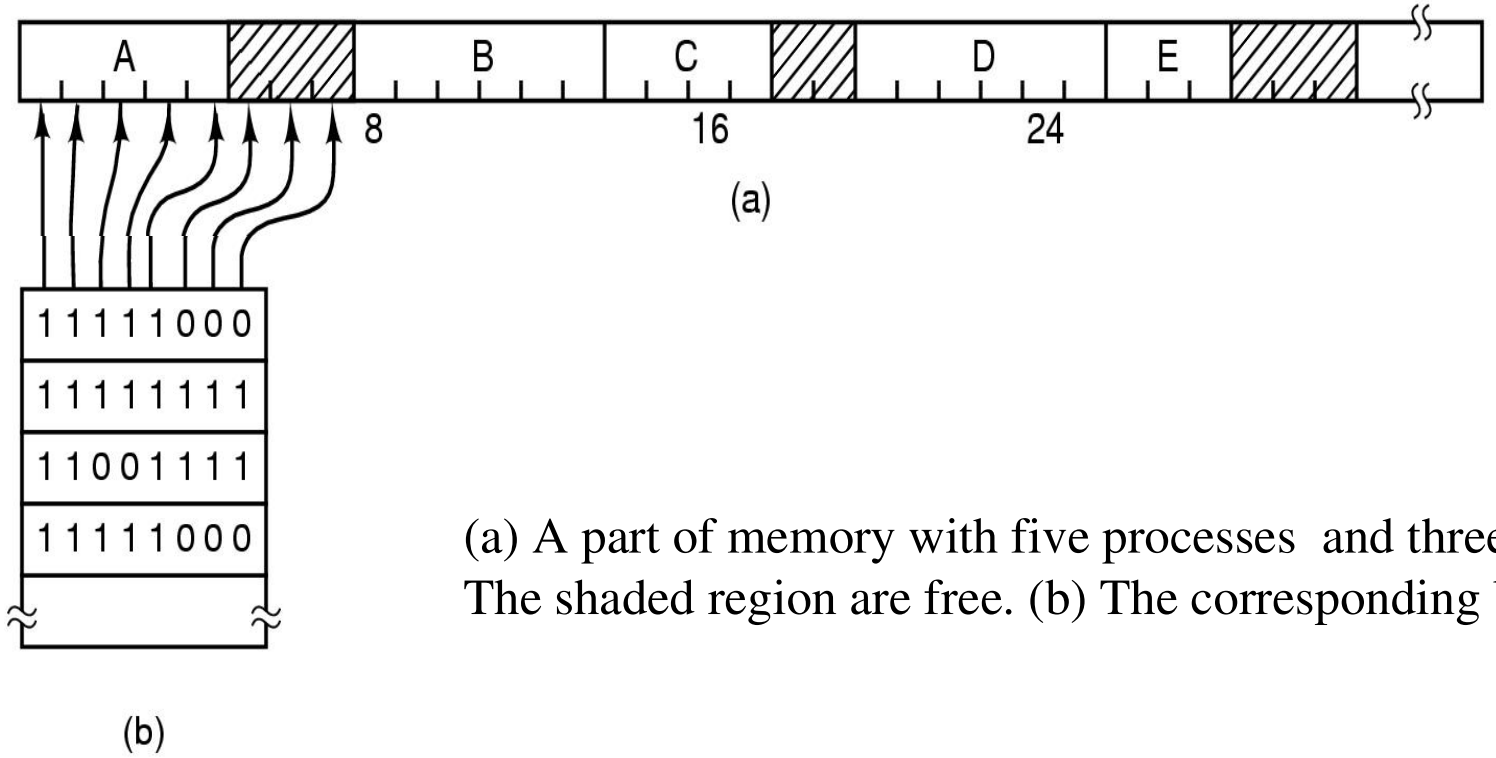
Eg: On a 256-MB machine that can copy 4 bytes in 40 nsec, it takes about 2.7 sec to compact all memory.

It stops every thing when compaction.

This technique is not used.

Variable Partition Multiprogramming
Bitmap Management

Memory is divided up into allocations units and each bit in the map indicates whether or not the corresponding unit is allocated.



(a) A part of memory with five processes and three holes. The shaded region are free. (b) The corresponding bitmap.

Variable Partition Multiprogramming

Bitmap Management

- Each allocation unit is a bit in the bit map, 0 if the unit is free, and 1 if it is occupied (or vice versa).
- When a process come into the memory, it search required numbers of consecutive 0 bits in the map.
- The size of the allocation unit is an important design issue; Increasing the size of allocation decrease the size of bitmap, but increase the amount of memory wasted when the process size is not a multiple of size of the allocation unit.

Advantages:

Provides a simple way to keep track of memory words.

Problems:

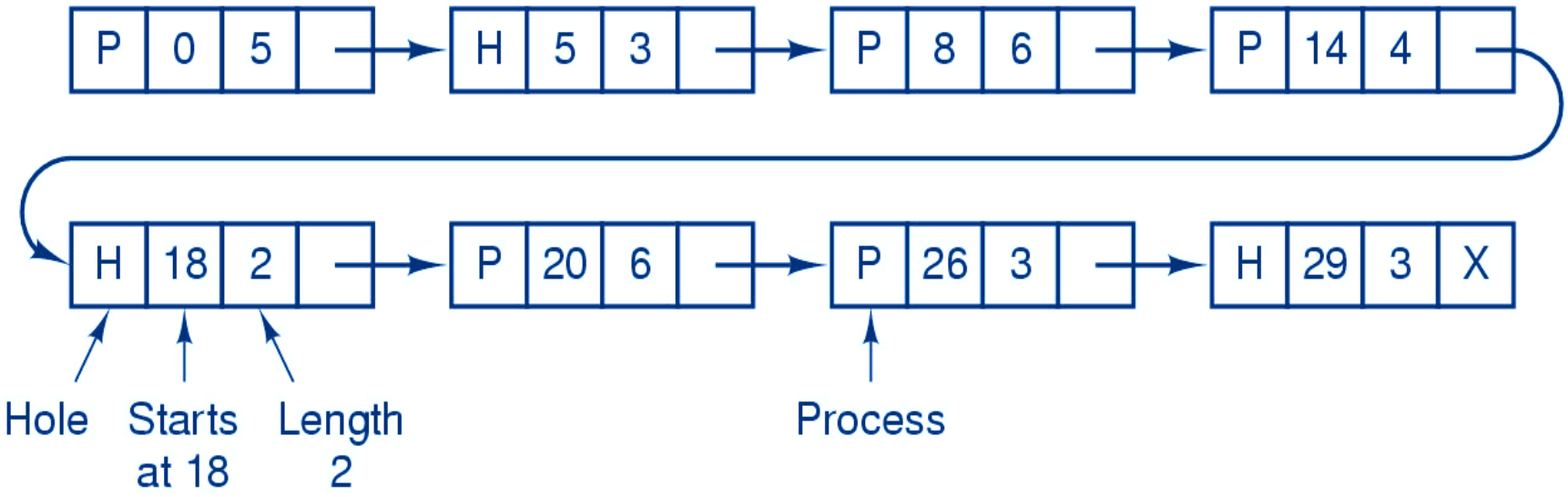
Searching a bitmap for a run of given length is a slow operation.

Variable Partition Multiprogramming

Linked List Management

Maintains a linked list of allocated and free memory segments, where a segment is either a process or a hole between the two process.

Each segment contains an entry indicating the segment size and a pointer to the next segment in the list.



Variable Partition Multiprogramming

Linked List Management

H represents hole and P represents process and the segment list is kept sorted by address. Sorting has an advantage that when a process terminates, updating the list is straightforward.
It may be implemented as doubly-linked list to make a more convenient search.
When a process terminates, if any neighbors is already hole, merge the holes – called coalescing.

Problem : Still some form of fragmentation

Variable Partition Multiprogramming

Partition Selection Algorithms

Situation: Multiple memory holes are large enough to contain a process, OS must select which hole the process will be loaded into.

First Fit

Allocate the first hole that is big enough. It stop the searching as soon as it find a free hole that is large enough. The hole is then broken up into two pieces, one for the process and one for unused memory.

Advantage: It is a fast algorithm because it search as little as possible.

Problem: not good in terms of storage utilization.

Variable Partition Multiprogramming

Partition Selection Algorithms

Best Fit

Allocate the smallest hole that is big enough. It search the entire list, and takes the smallest hole that is big enough. Rather than breaking up a big hole that might be needed later, it finds a hole that is close to the actual size.

Advantage: More storage utilization than first fit but not always.

Problem: Slower than first fit because it requires to search whole list at every time.

Creates tiny hole that may not be used.

Variable Partition Multiprogramming
Partition Selection Algorithms

Worst Fit

Allocate the largest hole. It search the entire list, and takes the largest hole. Rather than creating tiny hole it produces the largest leftover hole, which may be more useful.

Advantage: Some time it has more storage utilization than first fit and best fit.

Problem: not good for both performance and utilization.

Swapping

Till now: *User process remains in main memory until completion and contiguous allocation.*

In timesharing system, the situation may be different, some times there is no enough memory to hold all processes.

How to handle this problem? – *swapping*.

Access process must be kept on the disk and brought it into memory and run dynamically.

A process, however, can be swapped temporarily out of the memory to disk, and then brought back into memory for continued execution.

In timesharing system processes will be swapped in and out many times before its completion.

In some swapping systems, one processes occupies the main storage at once; when process can no longer to continue it relinquishes both the storage and CPU.

Overlays

In the past when the programs were too big to fit in available main memory, the solution adopted was to split the program into pieces called overlays.
Overlays 1 would start running first; when it was done, it would call next overlay 2 and so on.
The overlays are kept on disk and swapped in and out by OS.
Problem:
The job of splitting program into pieces (making overlays) had to be done by programmer; time consuming and boring for programmer.

Solution: Virtual memory

Home Works

HW #7

1. Problems 1, 3, 4 and 5 (Textbook, Tanenbaum, Ch. 4).
2. How fragmentation occur? Discuss the techniques that manage the fragmentation.
3. A program is created by assuming it will be loaded at the address 100, so the program refers address as: 135, 160, 164, 220, 224. But it loaded at the location 500, the will be physical address of the program.
4. Given a memory partitions of 100KB, 500KB, 200KB, 300KB, & 600KB (in order), how would each of the first fit, best fit and worst fit algorithms places processes of 212KB, 417KB, 112KB, & 426KB (in order)? Which algorithm makes the most efficient use of memory?

Be ready for virtual memory.