Advanced C and System Programming

Anandkumar

D-Bus

D-Bus

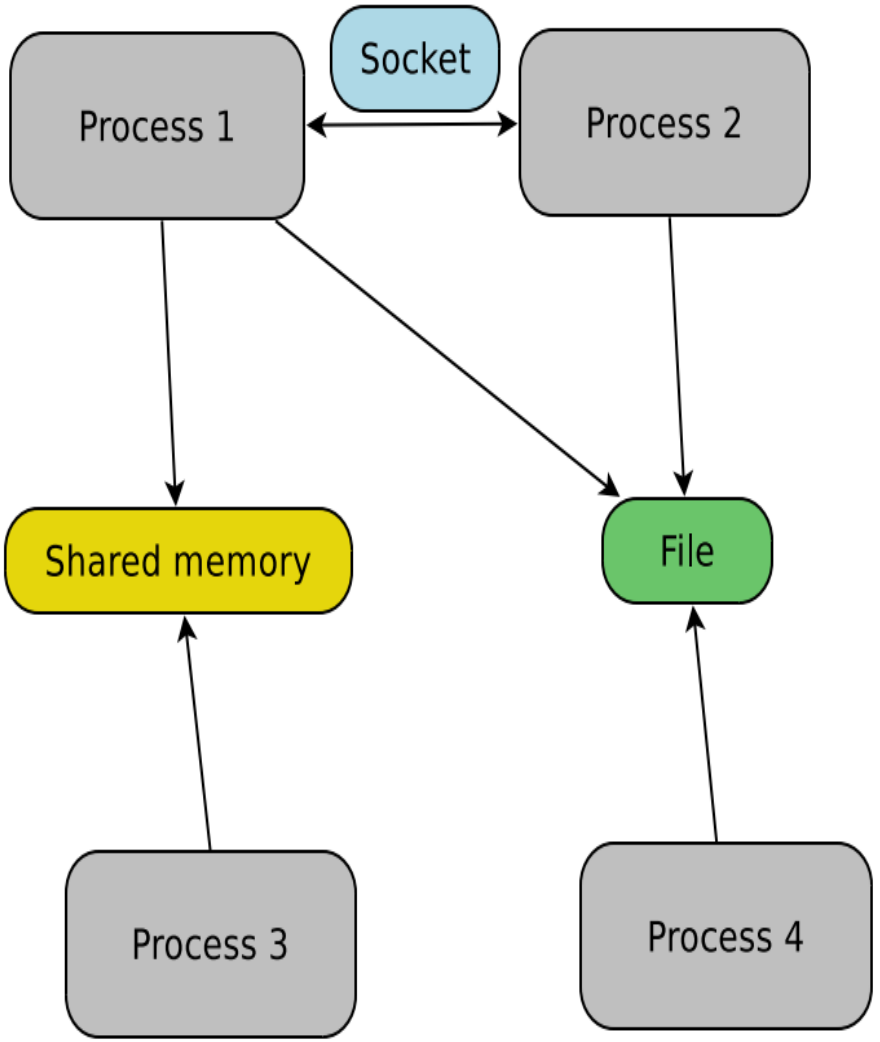
- Created in 2002
- Is part of the *freedesktop.org* project
- Maintained by RedHat and the community
- Is an Inter-process communication mechanism
- Initiated to standardize services of Linux desktop environments

freedesktop.org

redhat.

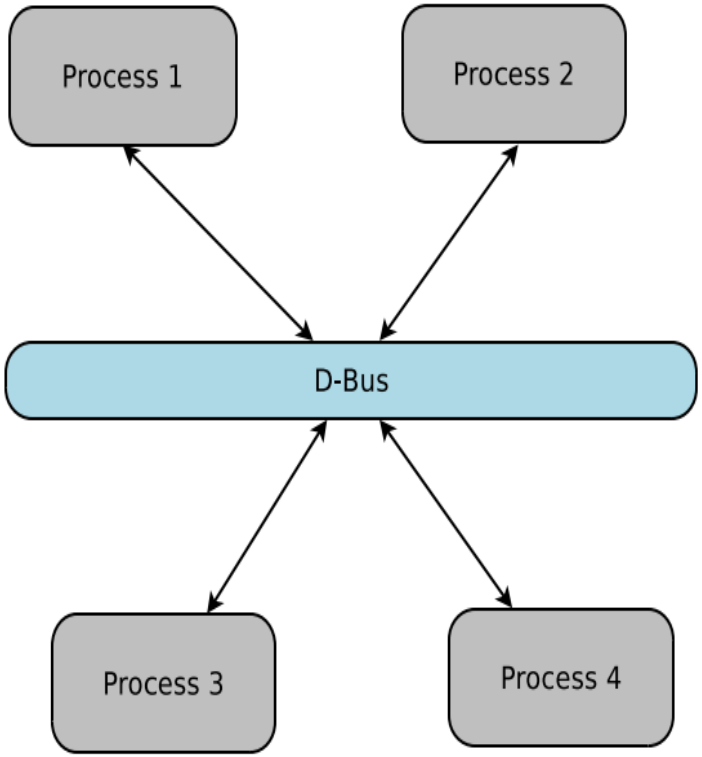
D-Bus

- Mechanisms allowing processes to communicate with each other
  - **Shared memory**: read/write into a defined memory location
  - **Memory-mapped file**: same as `shared memory` but uses a file
  - **Pipe**: two-way data stream (standard input / output)
  - **Named pipe**: same as `pipe` but uses a file (FIFO)
  - **Socket**: communication even on distant machines
  - and others



D-Bus

- Uses the socket mechanism
- Provides software bus abstraction
- Way simpler than most alternatives



D-Bus

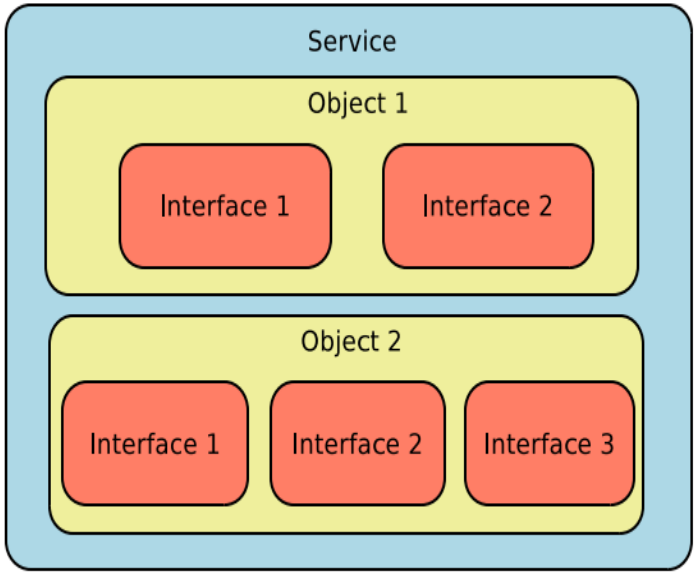
- D-Bus includes:
  - `libdbus`: a low-level library
  - `dbus-daemon`: a daemon based on libdbus. Handles and controls data transfers between DBus peers
  - two types of busses: a `system` and a `session` one. Each bus instance is managed by a `dbus-daemon`
  - a security mechanism using `policy` files

D-Bus

- System bus
  - On desktop, a single bus for all users
  - Dedicated to system services
  - Is about low-level events such as connection to a network, USB devices, etc
  - On embedded Linux systems, this bus is often the only D-Bus type
- Session bus
  - One instance per user session
  - Provides desktop services to user applications
  - Linked to the X session

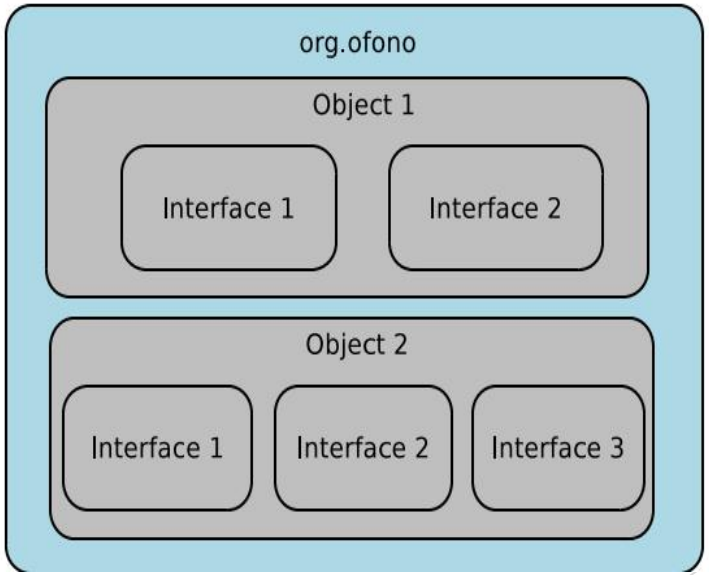
D-Bus

- D-Bus is working with different elements:
  - Services
  - Objects
  - Interfaces
  - Clients: applications using a D-Bus service
- One D-Bus *service* contains *object(s)* which implements *interface(s)*



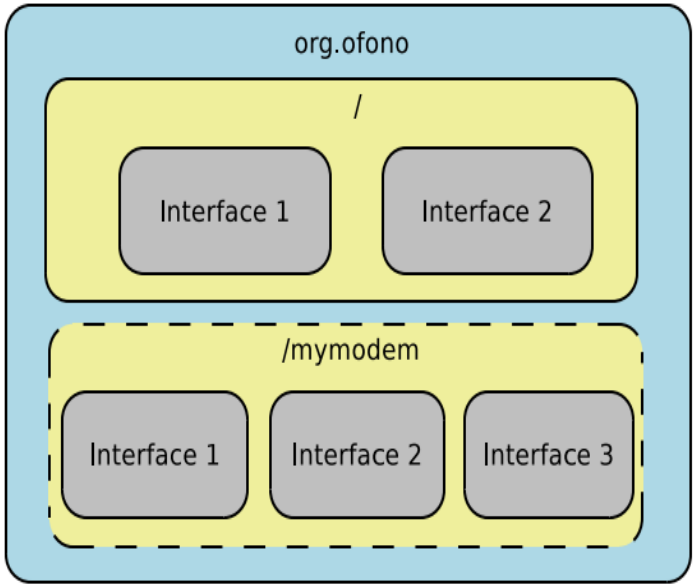
D-Bus Service

- An application can expose its services to all D-Bus users by registering to a bus instance
- A `service` is a collection of `objects` providing a specific set of features
- When an application opens a connection to a bus instance, it is assigned a unique name (ie `:1.40`)
- Can request a more human-readable service name: the **well-known name** (ie `org.ofono`) See the freedesktop.org specification



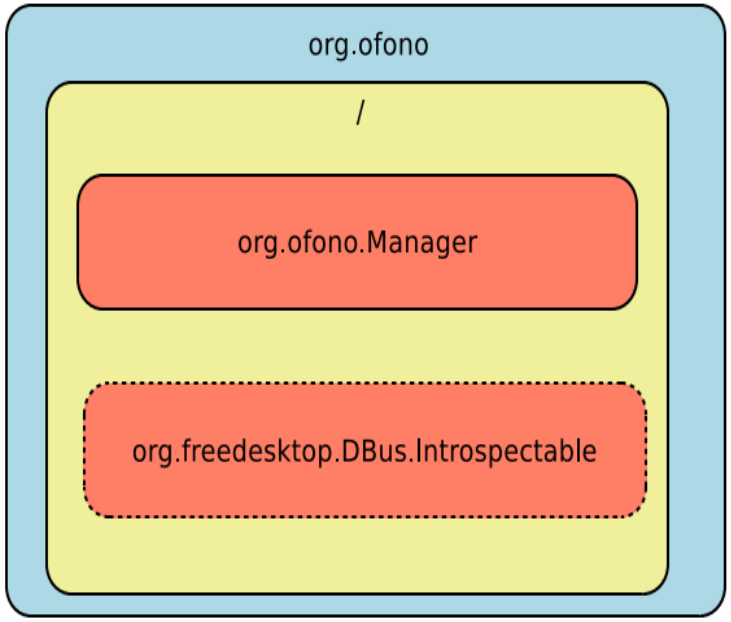
D-Bus Objects

- Are attached to one service
- Can be dynamically created or removed
- Are uniquely identified by an **object path** (ie / or `/net/connman/technology/cellular`)
- Implement one or several interfaces



D-Bus Interfaces

- Can be compared to a "namespace" in Java
- Has a unique name ressembling Java interface names, using dots (ie `org.ofono.Manager`)
- Contains *members*: properties, methods and signals



D-Bus Interfaces

- D-Bus defines a few standard interfaces
- They all belong to the namespace "org.freedesktop.DBus" :
  - **org.freedesktop.DBus.Introspectable** : Provides an introspection mechanism. Exposes information about the object (interfaces, methods and signals it implements)
  - **org.freedesktop.DBus.Peer** : Provides methods to know if a connection is alive (ping)
  - **org.freedesktop.DBus.Properties** : Provides methods and signals to handle properties
  - **org.freedesktop.DBus.ObjectManager** : Provides an helpful API to handle sub-tree objects
- Interfaces expose properties, methods and signals

D-Bus Properties

- Directly accessible fields
- Can be read / written
- Can be of different types defined by the D-Bus specification :
  - basic types: bytes, boolean, integer, double, ...
  - string-like types : string, object path (must be valid) and signature
  - container-types: structure, array, variant (complex types) and dictionnary entry (hash)
- Very convenient standard interface : `org.freedesktop.DBus.Properties`
- Types are represented by characters

byte	y	string	s	variant	v
boolean	b	object-path	o	array of int32	ai
int32	i	array	a	array of an array of int32	aai
uint32	u	struct	()	array of a struct with 2 int32 fields	a(ii)
double	d	dict	{}	dict of string and int32	{si}

D-Bus Methods

- allow remote procedure calls from one process to another
- Can be passed one or several parameters
- Can return values/objects
- Look like any method you could know from other languages

```
org.freedesktop.DBus.Properties :
  Get (String interface_name, String property_name) => Variant value
  GetAll (String interface_name) => Dict of {String, Variant} props
  Set (String interface_name, String property_name, Variant value)
```

D-Bus Signals

- Messages / notifications
- Unidirectionnal
- Sent to every clients that are listening to it
- Can contain parameters
- A client will subscribe to signals to get notifications

```
org.freedesktop.DBus.Properties :
   PropertiesChanged (String, Dict of {String, Variant}, Array of String)
```

D-Bus Policy

- Adds a security mechanism
- Represented by XML files
- Handled by each `dbus-daemon` (under `/etc/dbus-1/session.d` and `/etc/dbus-1/system.d`)
- Allows the administrator to control which user can talk to which interface, which user can send message to which interface, and so on
- If you are not able to talk with a D-Bus service or get an `org.freedesktop.DBus.Error.AccessDenied` error, check this file!
- `org.freedesktop.PolicyKit1` has been created to handle all security accesses

D-Bus

- In this example, "toto" can :
  - own the interface `org.ofono`
  - send messages to the owner of the given service
  - call `GetContexts` from interface `org.ofono.ConnectionManager`

```
<!DOCTYPE busconfig PUBLIC
 ``-//freedesktop//DTD D-BUS Bus Configuration 1.0//EN''
 ``http://www.freedesktop.org/standards/dbus/1.0/busconfig.dtd''>
 <busconfig>
    <policy user="toto">
        <allow own="org.ofono"/>
        <allow send_destination="org.ofono"/>
        <allow send_interface="org.ofono.ConnectionManager" send_member="GetContexts"/>
    </policy>
</busconfig>
```

- Can `allow` or `deny`

D-Bus

- Libdbus
  - This is the low-level library used by the dbus-daemon.
  - As the homepage of the project says: *"If you use this low-level API directly, you're signing up for some pain"*.
  - Recommended to use it only for small programs and you do not want to add many dependencies
- GDbus
  - Is part of GLib (GIO)
  - Provides a very comfortable API
- QtDbus
  - Is a Qt module
  - Is useful if you already have Qt on your system
  - Contains many classes to handle/interact such as `QDBusInterface`

D-Bus

- Bindings exist for other languages: dbus-python, dbus-java, ...
- All the bindings allow to:
  - Interact with existing D-Bus services
  - Create your own D-Bus services, objects, interfaces, and so on!
  - but... D-Bus is not a high performance IPC
  - Should be used only for **control** and not data
  - For example, you can use it to activate an audio pipeline but not to send the audio stream

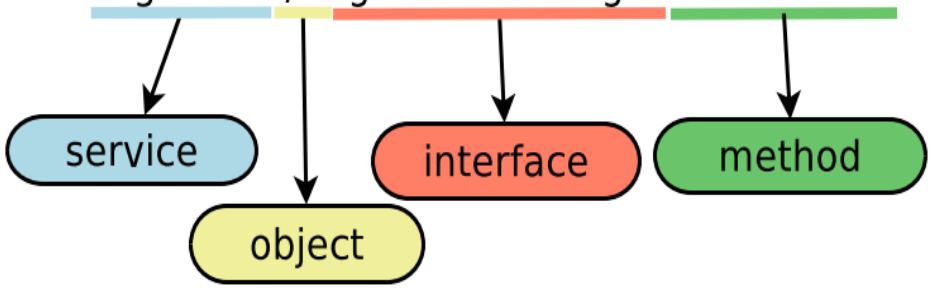
D-Bus

- Will present every tool with a demo
- dbus-send: Command-line interface (cli) to call method of interfaces (and get/set properties)
- dbus-monitor: Cli to subscribe and monitor signals
- gdbus: A GLib implementation of a more complete tool than dbus-send/monitor
- d-feet: A GUI application to handle all D-Bus services
- and others...

D-Bus

- Can chose the session or system bus (`--session` or `--system`)
- Here is an example:

```
dbus-send --system --print-reply --dest=org.ofono / org.ofono.Manager.GetModems
```



D-Bus

- Get properties:

```
dbus-send --system --print-reply --dest=net.connman / net.connman.Clock.GetProperties
```

- Set property:

```
dbus-send --system --print-reply --dest=net.connman \
          / net.connman.Clock.SetProperty \
          string:TimeUpdates variant:string:manual
```

- Using standard interfaces:

```
dbus-send --system --print-reply --dest=net.connman \
          / org.freedesktop.DBus.Introspectable.Introspect

dbus-send --system --print-reply --dest=fi.w1.wpa_supplicant1 \
          /fi/w1/wpa_supplicant1 org.freedesktop.DBus.Properties.Get \
          string:fi.w1.wpa_supplicant1 string:Interfaces
```

D-Bus

- Can monitor all traffic (including methods and signals if enabled in policy):

```
dbus-monitor
```

- Or filter messages based on the interface:

```
dbus-monitor --system type=signal interface=net.connman.Clock
```

D-Bus

- Also provides a command line interface
- Is more featureful than `dbus-send` because it handles "dict entry"
- Has a different interface: must add a "command" such as "call" or "monitor"

```
gdbus call --system --dest net.connman \
      --object-path / --method net.connman.Clock.GetProperties
gdbus call --system --dest net.connman --object-path / \
      --method net.connman.Clock.SetProperty 'TimeUpdates' "<'manual'>"
gdbus monitor --system --dest net.connman
```

- Can even emit signals

```
gdbus emit --session --object-path / --signal \\
       net.connman.Clock.PropertyChanged ``['TimeUpdates', ``\<'auto'\>'']''
```

D-Bus

- KDE: A desktop environment based on Qt
- Gnome: A desktop environment based on gtk
- Systemd: An init system
- Bluez: A project adding Bluetooth support under Linux
- Pidgin: An instant messaging client
- Network-manager: A daemon to manage network interfaces
- Modem-manager: A daemon to provide an API to dial with modems - works with Network-Manager
- Connman: Same as Network-Manager but works with Ofono for modem
- Ofono: A daemon that exposing features provided by telephony devices such as modem

Serial programming using Rapberry pi

Memory management

3.2 Variable Partitioning

With fixed partitions we have to deal with the problem of determining the number and sizes of partitions to minimize internal and external 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.

3.2 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.

3.2 Variable Partitioning

There are three algorithms for searching the list of free blocks for a specific amount of memory.

- First Fit
- Best Fit
- Worst Fit

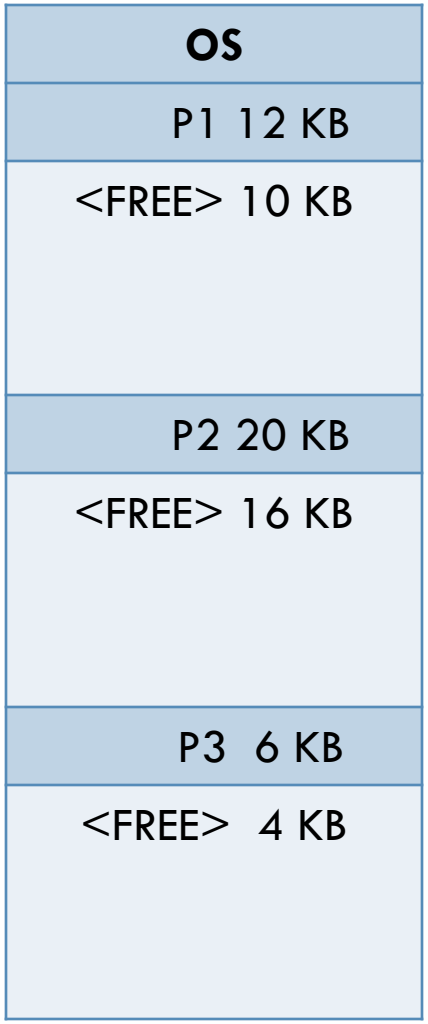
first fit

- First Fit : Allocate the first free block that is large enough for the new process.

- This is a fast algorithm.

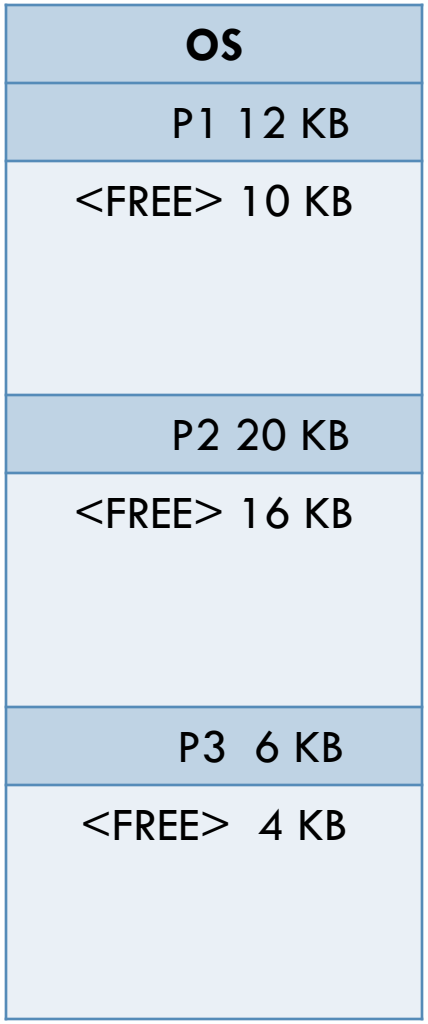
first fit

Initial memory mapping

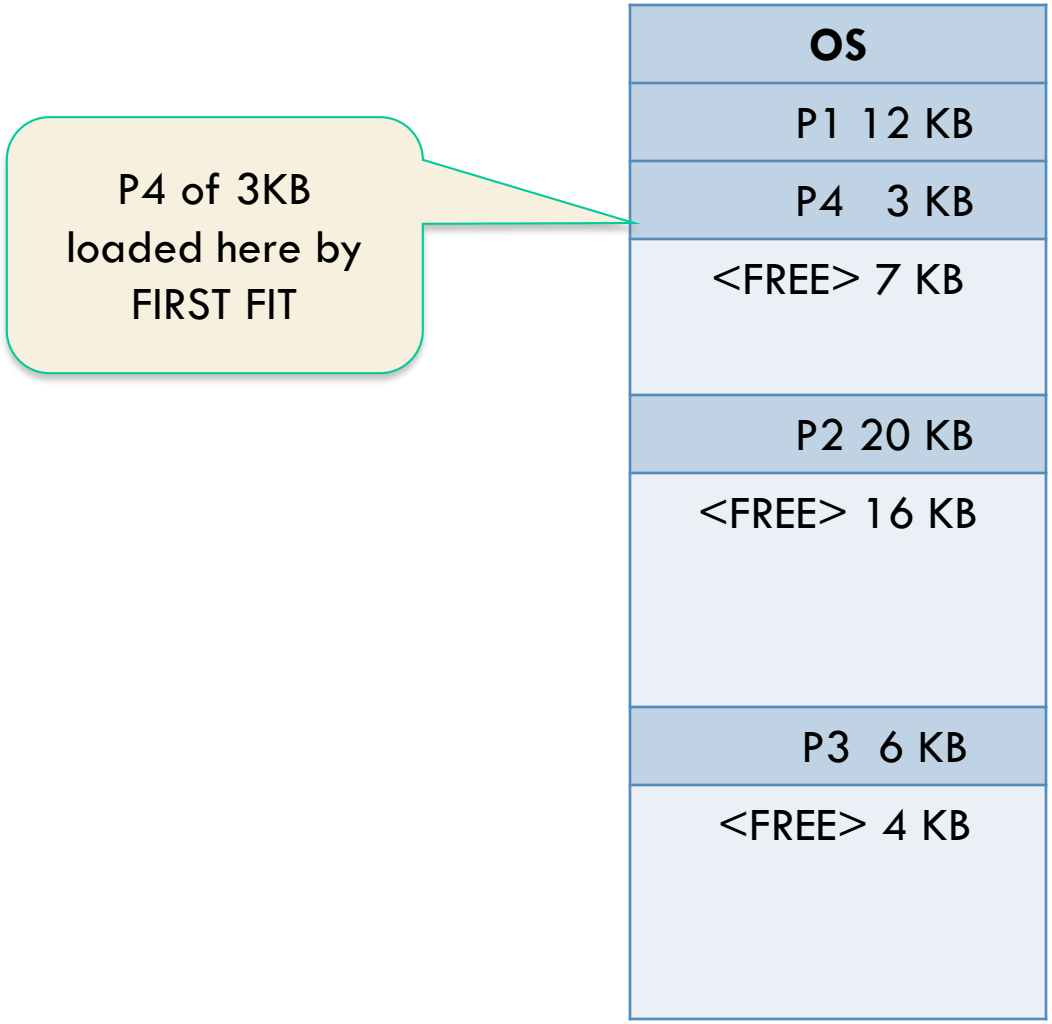


first fit

P4 of 3KB arrives

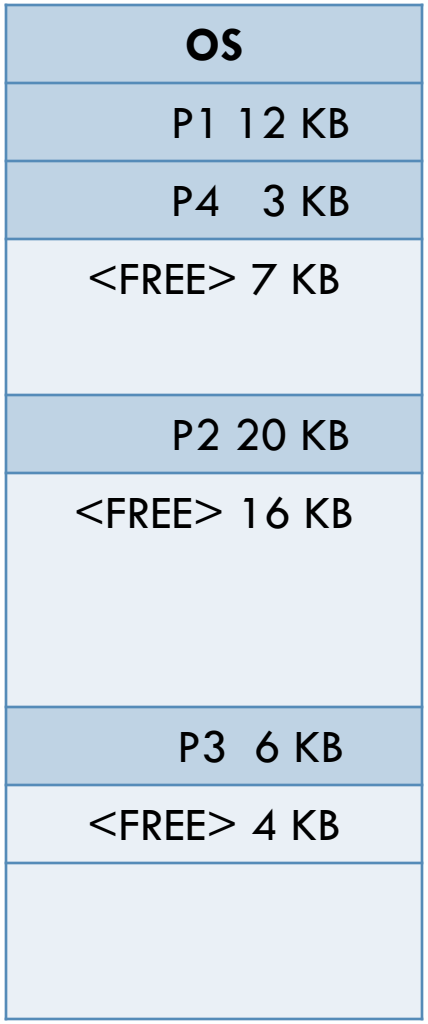


first fit


P4 of 3KB loaded here by FIRST FIT
OS
P1 12 KB
P4 3 KB
<FREE> 7 KB
P2 20 KB
<FREE> 16 KB
P3 6 KB
<FREE> 4 KB

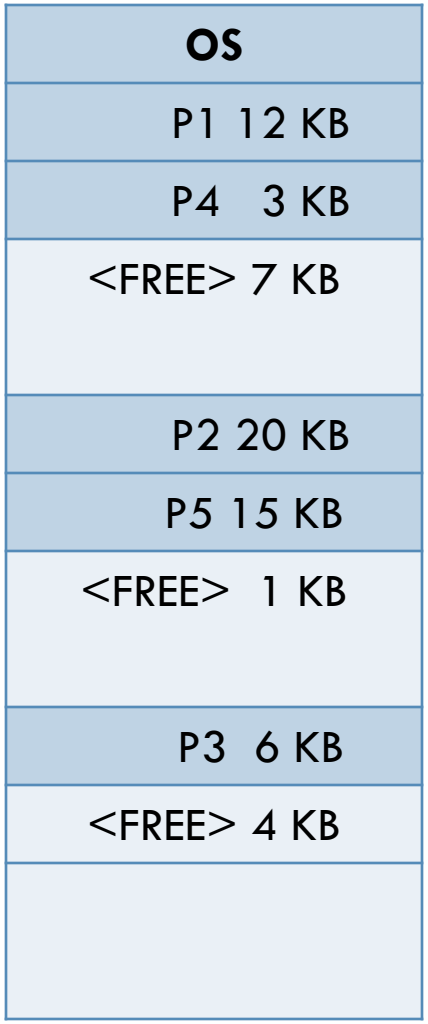
first fit

P5 of 15KB arrives



first fit

P5 of 15 KB loaded here by FIRST FIT

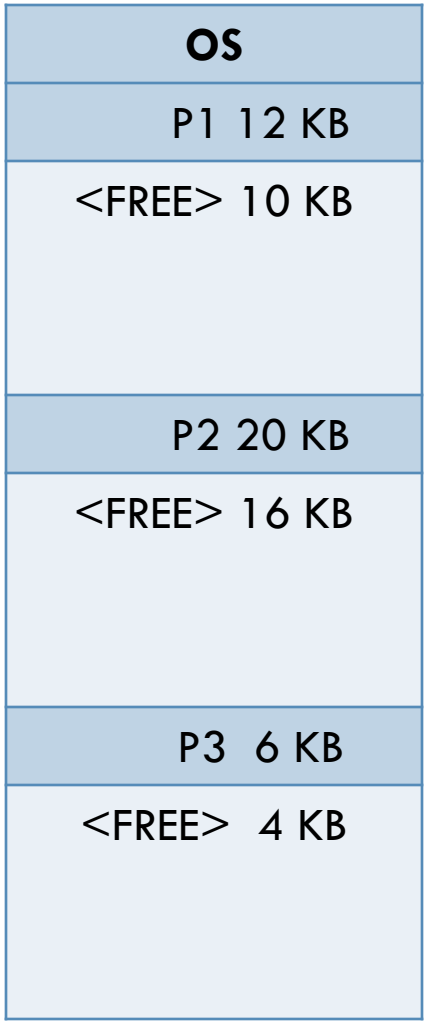


Best fit

- 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
- If sorting is used, merging the area released when a process terminates to neighboring free blocks, becomes complicated.

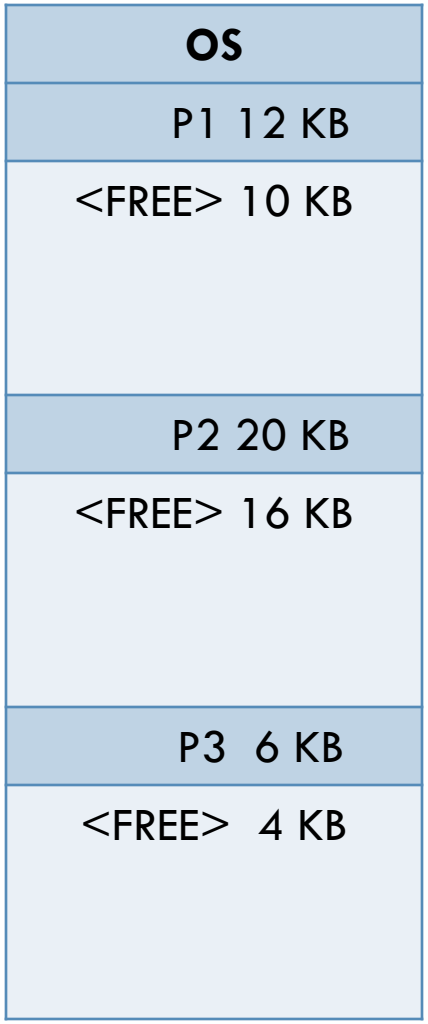
best fit

Initial memory mapping



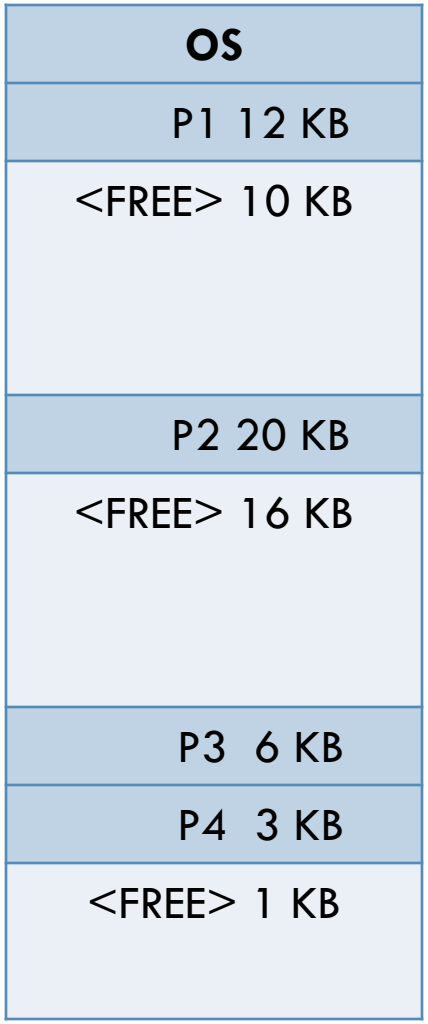
best fit

P4 of 3KB arrives



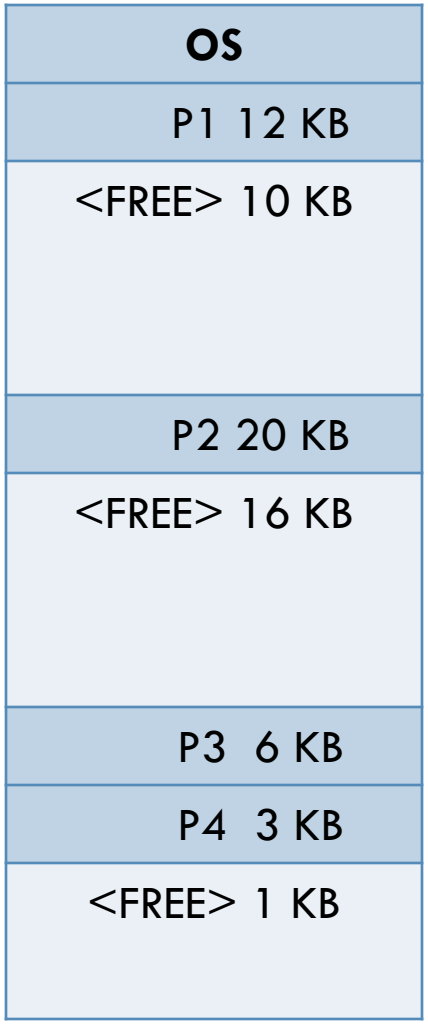
best fit

P4 of 3KB loaded here by BEST FIT



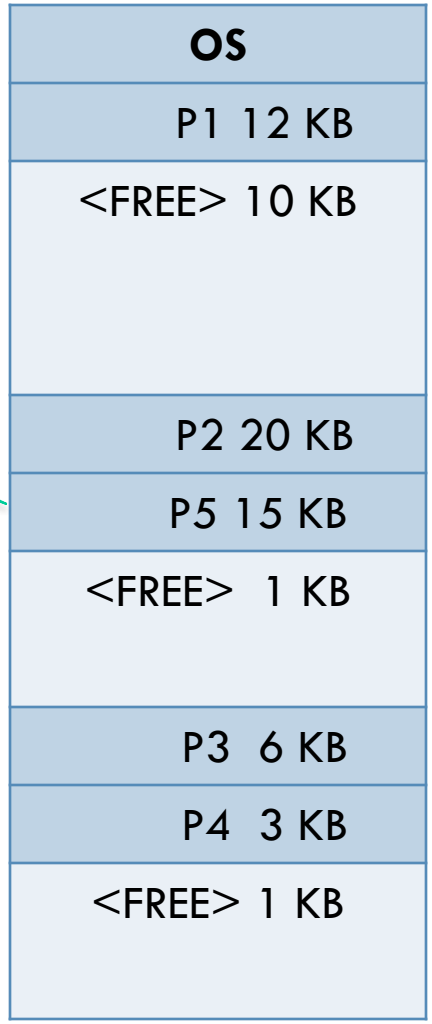
best fit

P5 of 15KB arrives



best fit

P5 of 15 KB loaded here by BEST FIT

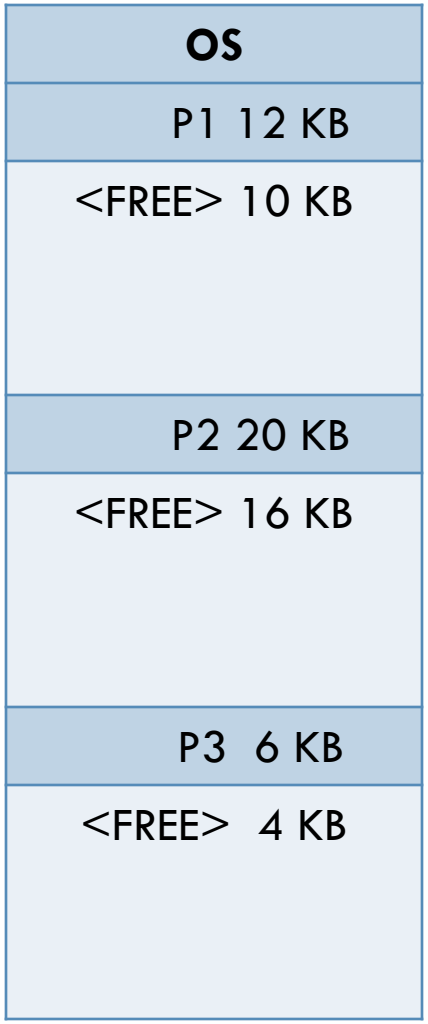


worst fit

- 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.

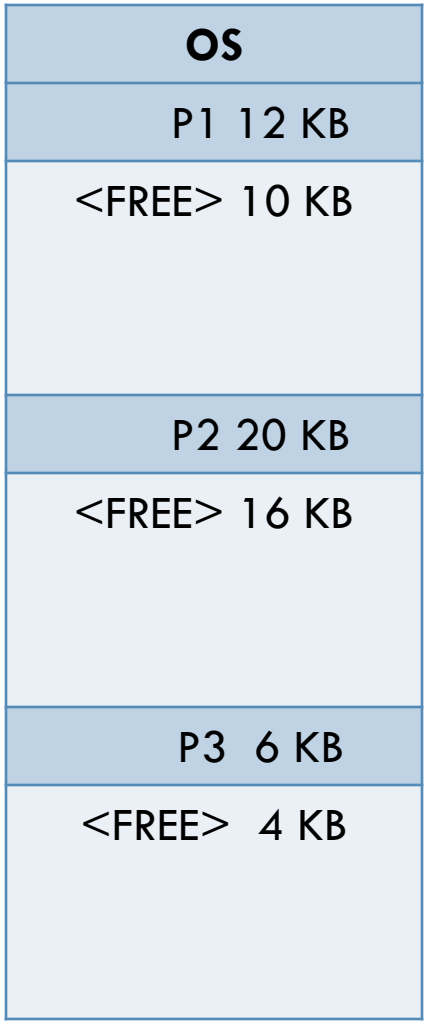
worst fit

Initial memory mapping



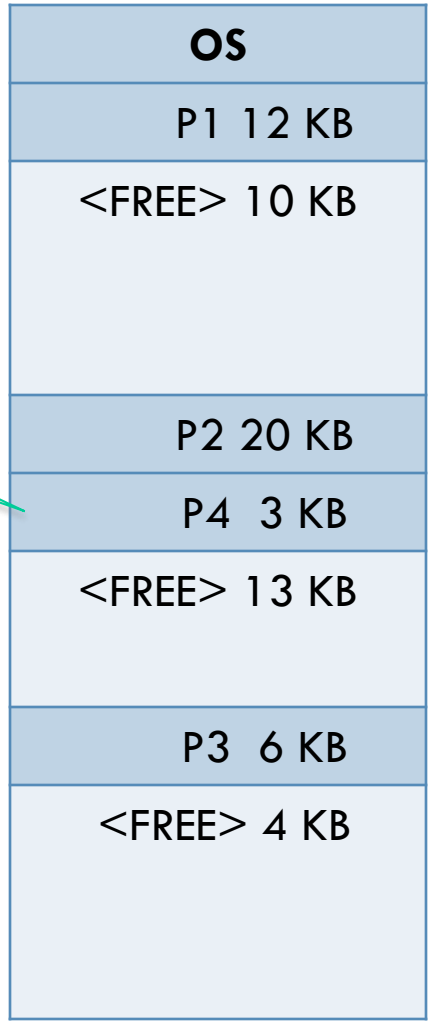
worst fit

P4 of 3KB arrives



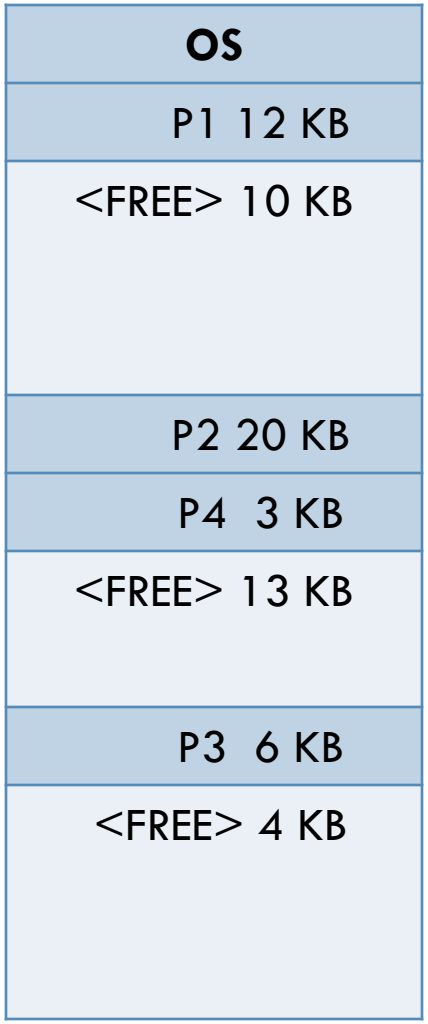
worst fit

P4 of 3KB
Loaded here by
WORST FIT



worst fit

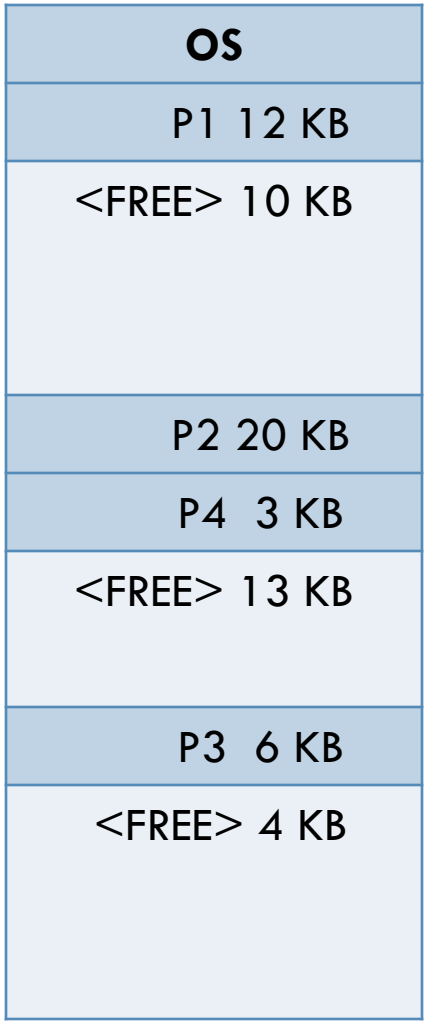
No place to load
P5 of 15K



worst fit

No place to load P5 of 15K

Compaction is needed !!



compaction

Compaction is a method to overcome the external fragmentation problem.

All free blocks are brought together as one large block of free space.

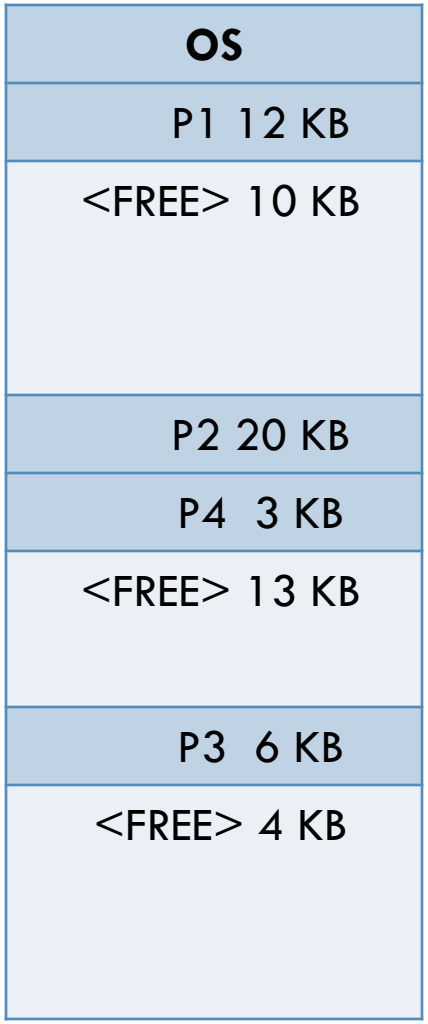
Compaction requires dynamic relocation.

Certainly, compaction has a cost and selection of an optimal compaction strategy is difficult.

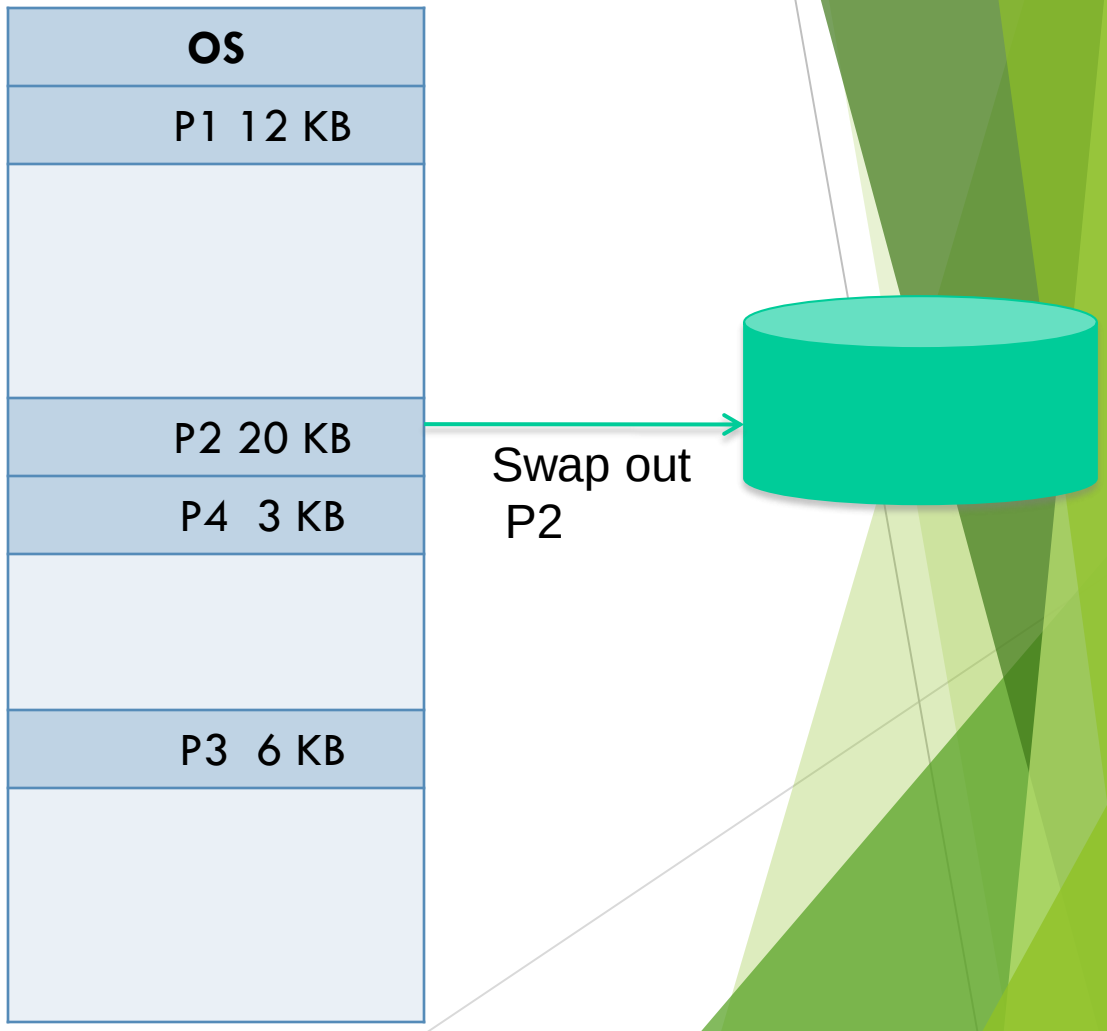
One method for compaction is swapping out those processes that are to be moved within the memory, and swapping them into different memory locations

compaction

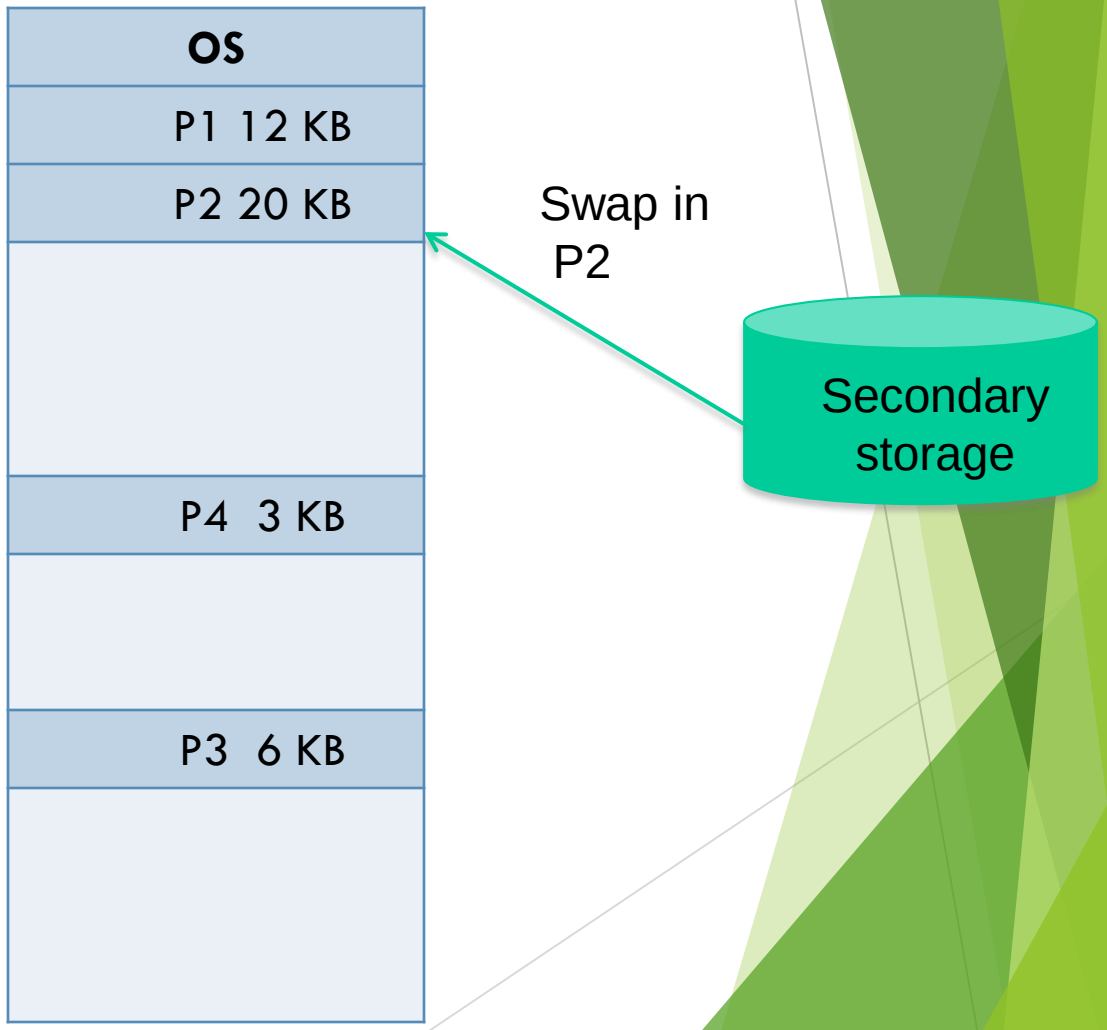
Memory mapping before compaction



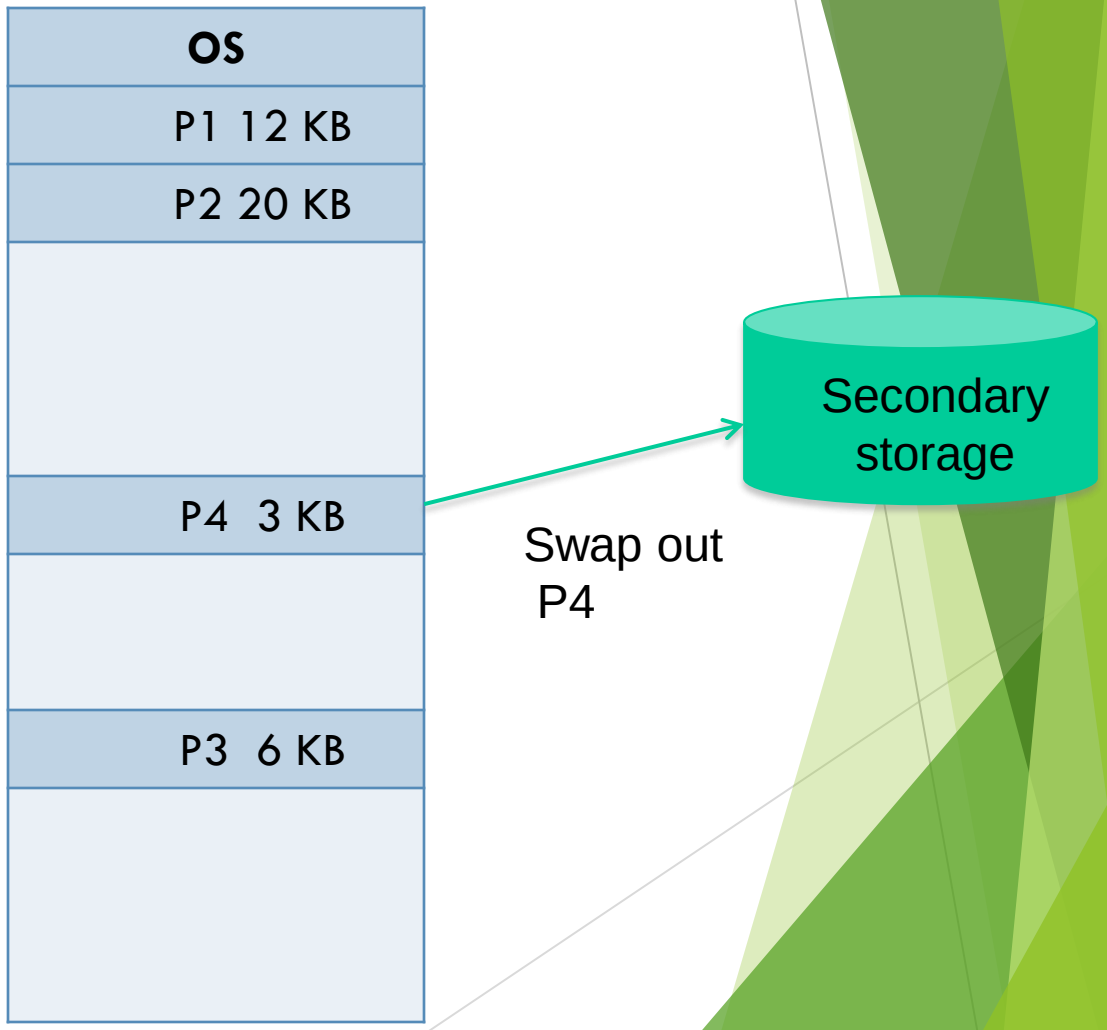
compaction



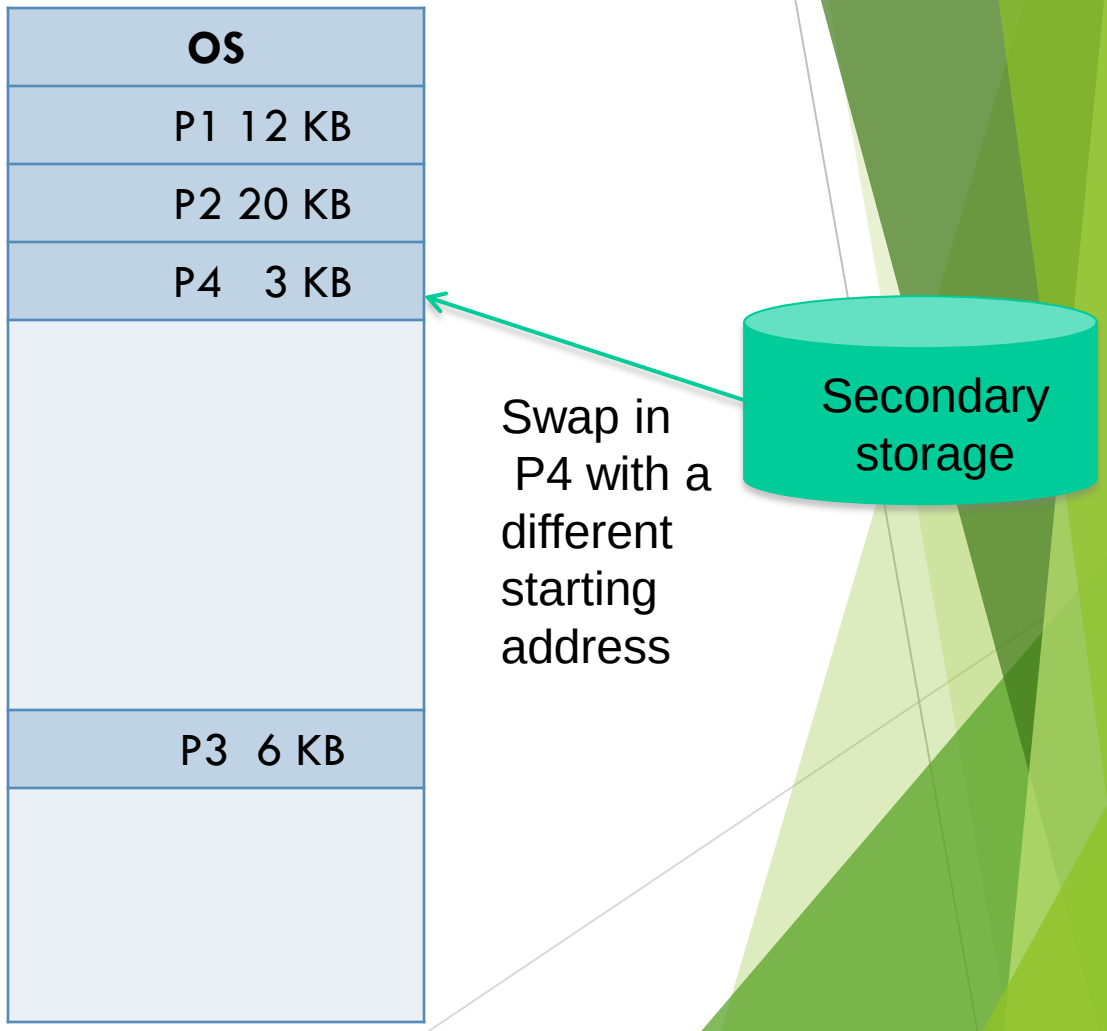
compaction



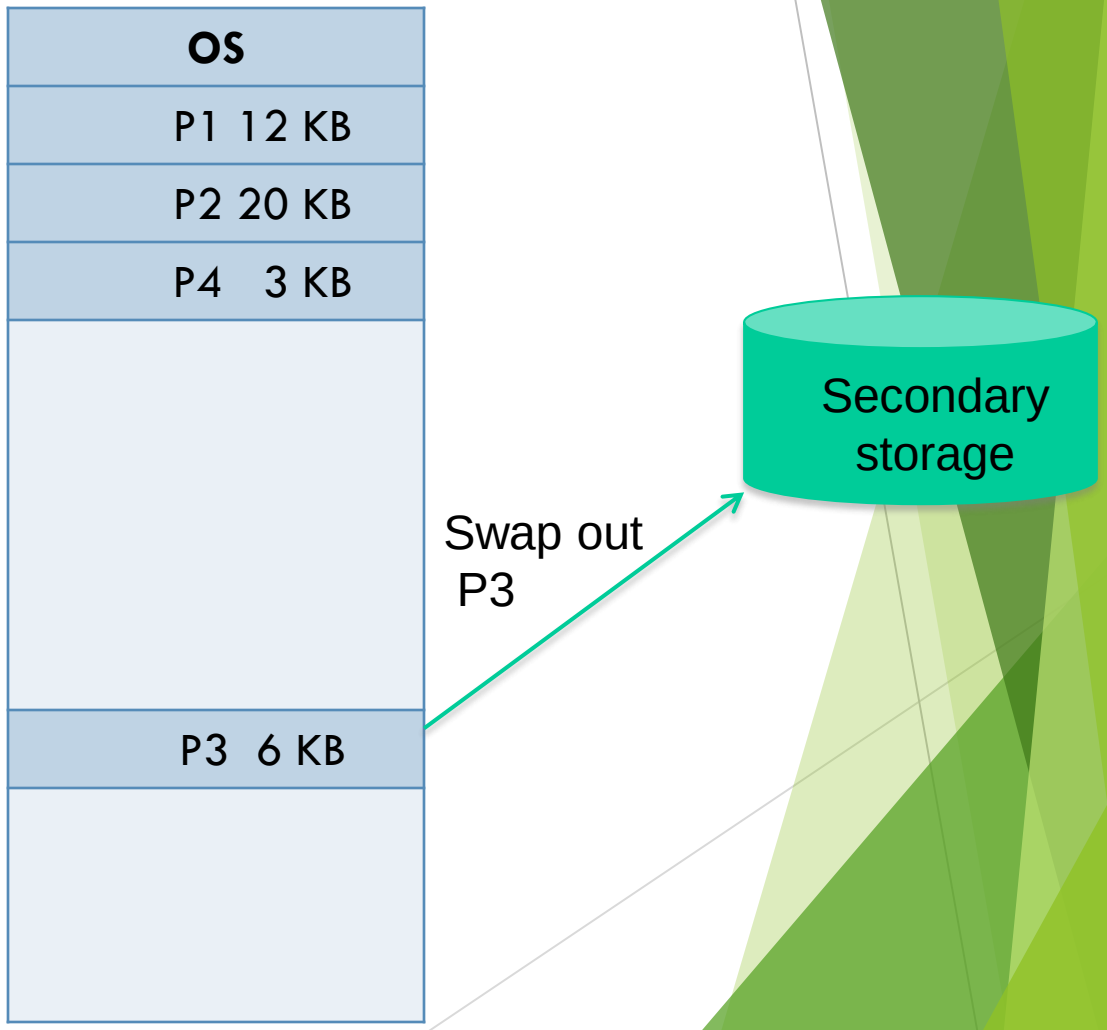
compaction



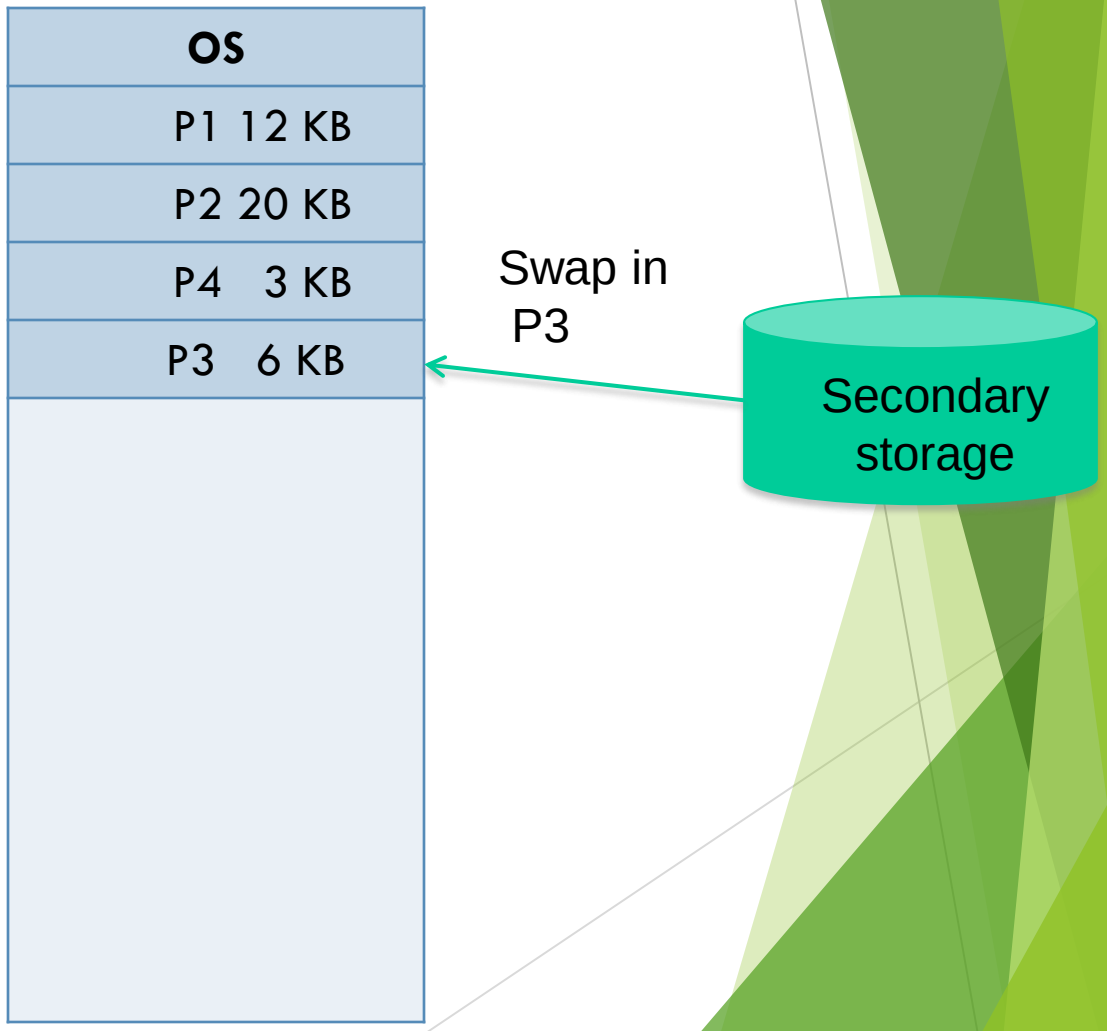
compaction


OS
P1 12 KB
P2 20 KB
P4 3 KB
P3 6 KB
Swap in P4 with a different starting address
Secondary storage

compaction

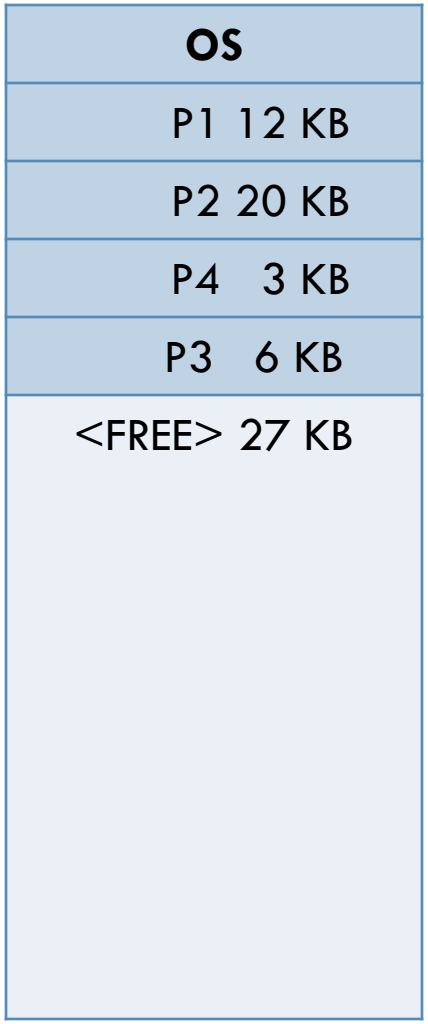


compaction



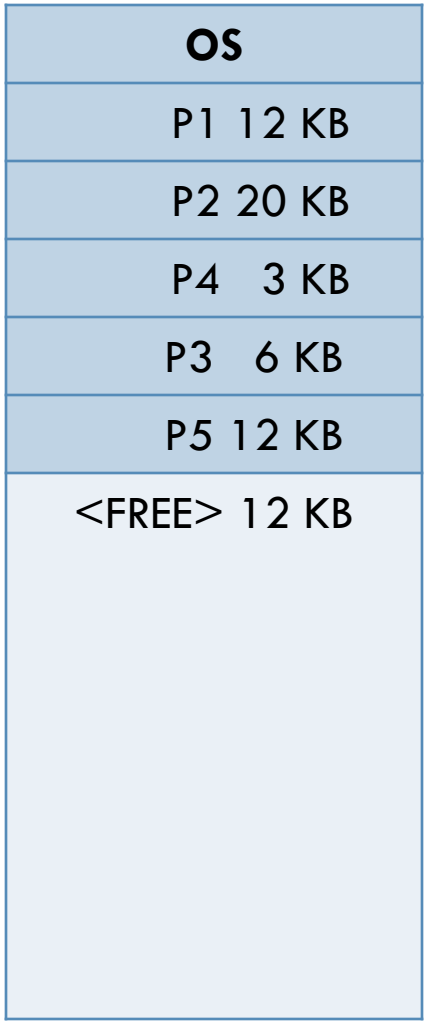
compaction

Memory mapping after compaction



Now P5 of 15KB can be loaded here

compaction



P5 of 15KB is loaded

Memory Allocation Functions

malloc

Allocates requested number of bytes and returns a pointer to the first byte of the allocated space

calloc

Allocates space for an array of elements, initializes them to zero and then returns a pointer to the memory.

free

Frees previously allocated space.

realloc

Modifies the size of previously allocated space.

We will only do malloc and free

Allocating a Block of Memory

A block of memory can be allocated using the function malloc

Reserves a block of memory of specified size and returns a pointer of type void

The return pointer can be type-casted to any pointer type

General format:

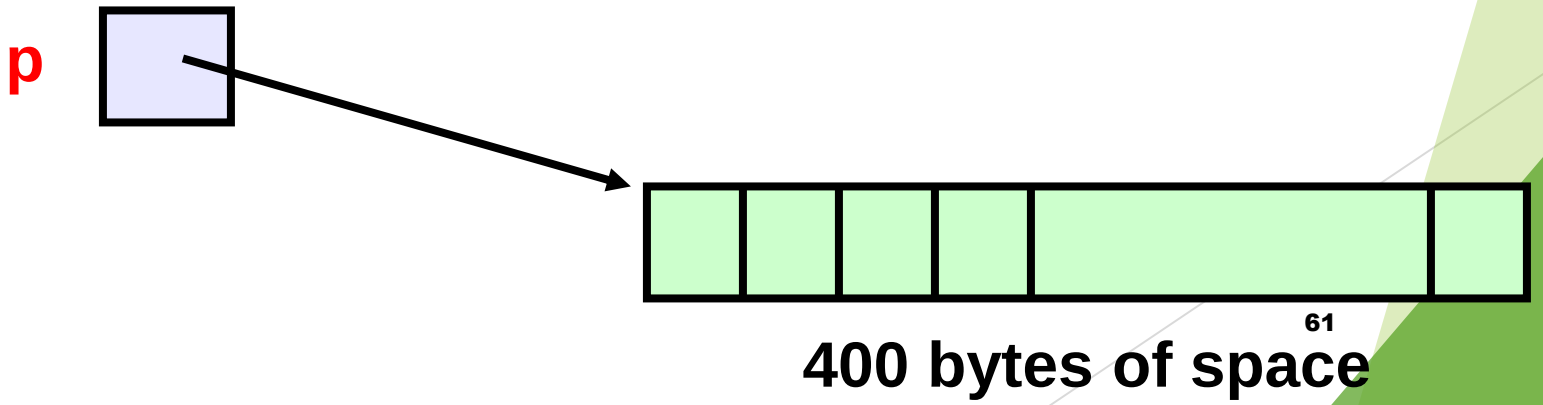
```
type *p;
p =  (type *) malloc (byte_size);
```

Example

```
p = (int *) malloc(100 * sizeof(int));
```

A memory space equivalent to 100 times the size of an int bytes is reserved

The address of the first byte of the allocated memory is assigned to the pointer p of type int



Contd.

```
cptr = (char *) malloc (20);
```

Allocates 20 bytes of space for the pointer cptr of type char

```
sptr = (struct stud *) malloc(10*sizeof(struct stud));
```

Allocates space for a structure array of 10 elements. sptr points to a structure element of type struct stud

Always use sizeof operator to find number of bytes for a data type, as it can vary from machine to machine

Points to Note

malloc always allocates a block of contiguous bytes

The allocation can fail if sufficient contiguous memory space is not available

If it fails, malloc returns NULL

```
if  ((p = (int *) malloc(100 * sizeof(int))) == NULL)
  {
      printf ("\n Memory cannot be allocated");
      exit();
   }
```

Releasing the Allocated Space: free

An allocated block can be returned to the system for future use by using the free function

General syntax:

```
free (ptr);
```

where ptr is a pointer to a memory block which has been previously created using malloc

Note that no size needs to be mentioned for the allocated block, the system remembers it for each pointer returned

Implementation of malloc and free

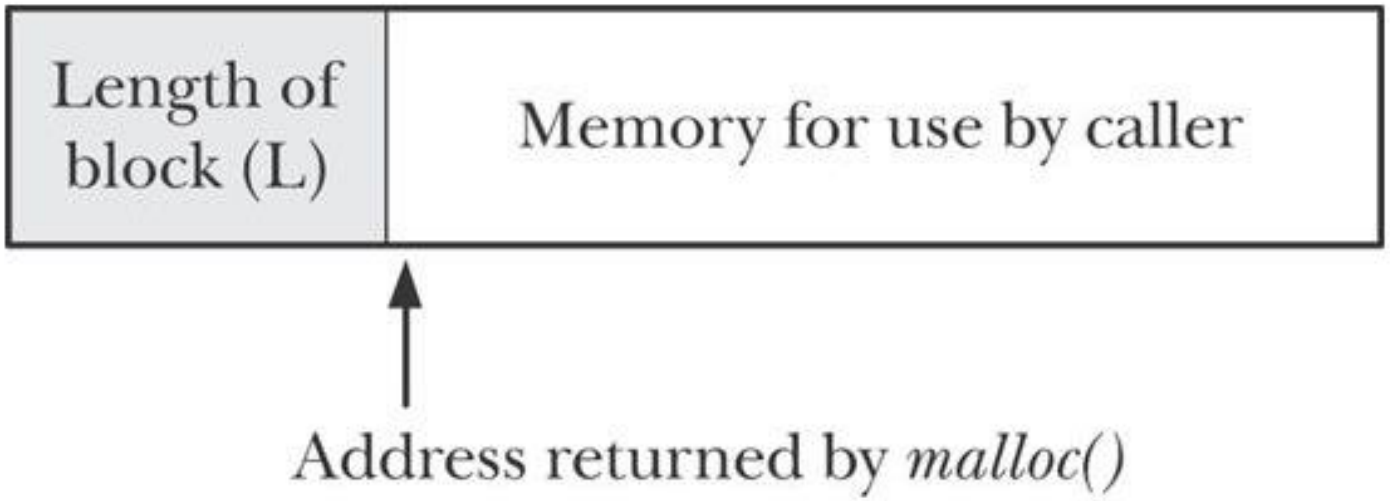


Figure 7-1. Memory block returned by malloc()

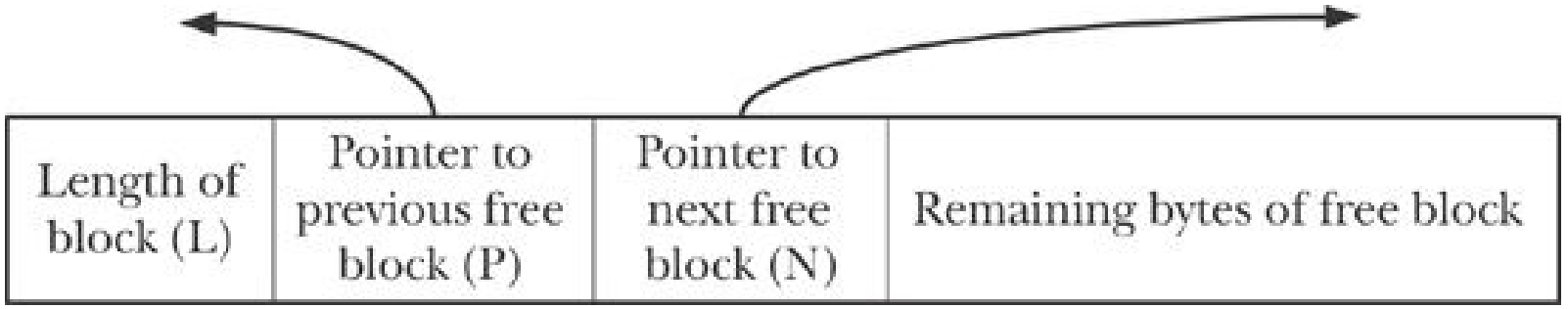


Figure 7-2. A block on the free list

Fig 7.3

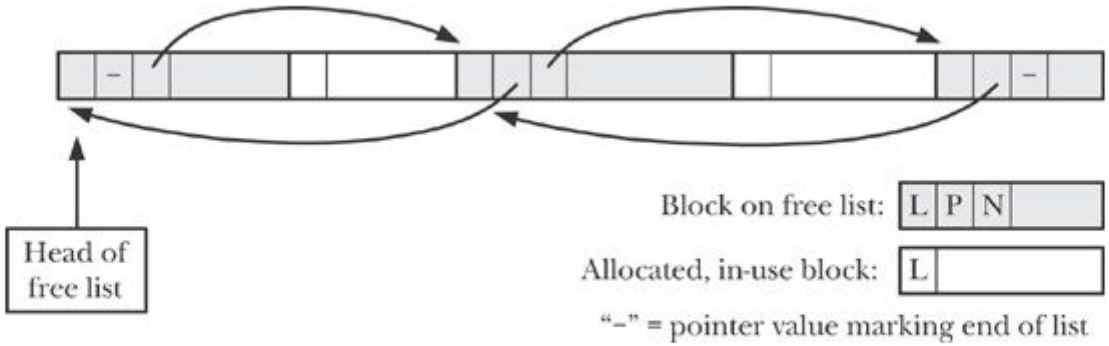


Figure 7-3. Heap containing allocated blocks and a free list

Checking for memory leaks?

NAME -- brk, sbrk - change data segment size

SYNOPSIS

```
#include <unistd.h>

int brk(void *addr);

void *sbrk(intptr_t increment);

brk(), sbrk(): _BSD_SOURCE || _SVID_SOURCE || _XOPEN_SOURCE >= 500
```

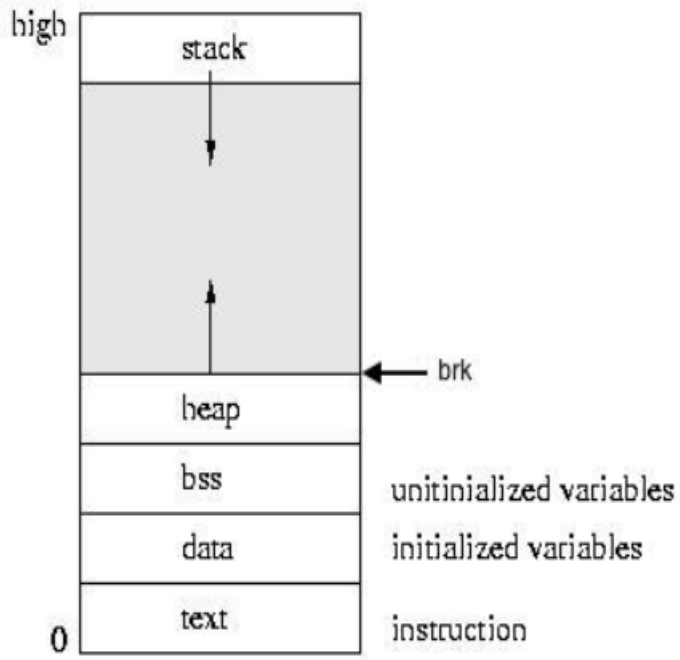
DESCRIPTION - brk() and sbrk() change the location of the program break, which defines the end of the process's data segment. Increasing the program break has the effect of allocating memory to the process; decreasing the break deallocates memory.

brk() sets the end of the data segment to the value specified by addr,

when that value is reasonable, the system has enough memory, and the process does not exceed its maximum data size (see setrlimit(2)).

sbrk() increments the program's data space by increment bytes.

Text section: The part that contains the binary instructions to be executed by the processor.
Data section: Contains non-zero initialized static data.
BSS (Block Started by Symbol) : Contains zero-initialized static data. Static data uninitialized in program is initialized 0 and goes here.
Heap: Contains the dynamically allocated data.
Stack: Contains your automatic variables, function arguments, copy of base pointer etc.



As you can see in the image, the stack and the heap grow in the opposite directions.
Sometimes the data, bss and heap sections are collectively referred to as the "*data segment*",
the end of which is demarcated by a pointer named program break or **brk**.
That is, *brk* points to the end of the heap.

Now if we want to allocate more memory in the heap, we need to request the system to increment brk. Similarly, to release memory we need to request the system to decrement *brk*.

malloc_usable_size - obtain size of block of memory allocated from heap

Synopsis

#include <malloc.h>

size_t malloc_usable_size (void *_ptr_);

Description

The **malloc_usable_size**() function returns the number of usable bytes in the block pointed to by *ptr*, a pointer to a block of memory allocated by ***malloc***(*3*) or a related function.

Return Value

malloc_usable_size() returns the number of usable bytes in the block of allocated memory pointed to by *ptr*. If *ptr* is NULL, 0 is returned.

NAME top

malloc_stats - print memory allocation statistics

SYNOPSIS top

```
#include <malloc.h>

void malloc_stats(void);
```

DESCRIPTION top

The **malloc_stats**() function prints (on standard error) statistics about memory allocated by malloc(3) and related functions. For each arena (allocation area), this function prints the total amount of memory allocated and the total number of bytes consumed by in-use allocations. (These two values correspond to the *arena* and *uordblks* fields retrieved by mallinfo(3).) In addition, the function prints the sum of these two statistics for all arenas, and the maximum number of blocks and bytes that were ever simultaneously allocated using mmap(2).

__malloc_hook, __malloc_initialize_hook, __memalign_hook, __free_hook, __realloc_hook, __after_morecore_hook - malloc debugging variables

Synopsis

```
#include <malloc.h>
void *(*__malloc_hook)(size_t size, const void *caller);
void *(*__realloc_hook)(void *ptr, size_t size", const void *" caller );

void *(*__memalign_hook)(size_t alignment, size_t size,
         const void *caller);
void (*__free_hook)(void *ptr, const void *caller);
void (*__malloc_initialize_hook)(void);
void (*__after_morecore_hook)(void);
```

Description

The GNU C library lets you modify the behavior of ***malloc***(3), ***realloc***(3), and ***free***(3) by specifying appropriate hook functions. You can use these hooks to help you debug programs that use dynamic memory allocation, for example.

The variable **__malloc_initialize_hook** points at a function that is called once when the malloc implementation is initialized. This is a weak variable, so it can be overridden in the application with a definition like the following:

```
void (*__malloc_initialize_hook)(void) = my_init_hook;
```

Now the function *my_init_hook*() can do the initialization of all hooks.

alloca - allocate memory that is automatically freed

Synopsis

#include <alloca.h>

void *alloca(size_t *size***);**

Description

The **alloca**() function allocates *size* bytes of space in the stack frame of the caller. This temporary space is automatically freed when the function that called **alloca**() returns to its caller.

Introduction to Bash Shell

What is Shell?

The shell is a command interpreter.

It is the layer between the operating system kernel and the user.

Some Special characters used in shell scripts

#:Comments

~:home directory

Invoking the script

The first line must be “#!/bin/bash”.

- setup the shell path

chmod u+x scriptname (gives only the script owner execute permission)

./scripname

Some Internal Commands and Builtins

getopts:

parses command line arguments passed to the script.

exit:

- Unconditionally terminates a script

set:

- changes the value of internal script variables.

read:

- **Reads" the value of a variable from stdin**
- **also "read" its variable value from a file redirected to stdin**

Some Internal Commands and Builtins (cont.)

grep:

grep pattern file

- search the files file, etc. for occurrences of *pattern*

expr:

- evaluates the arguments according to the operation given

y=`expr $y + 1` (same as **y=$(($y+1))**

I/O Redirection

>: Redirect stdout to a file, Creates the file if not present, otherwise overwrites it

< : Accept input from a file.

>>: Creates the file if not present, otherwise appends to it.

<<:

Forces the input to a command to be the shell's input, which until there is a line that contains only *label*.

cat >> mshfile << .

if

```
 if [ condition ] then
    command1
 elif   # Same as else if
then
    command1
else
    default-command
 fi
```

case

```
x=5
 case $x in
   0) echo "Value of x is 0."
      ;
   5) echo "Value of x is 5."
      ;
   9) echo "Value of x is 9."
      ;
   *) echo "Unrecognized value."
esac
 done
```

Loops

for [arg] in [list];

do

command

done

while [condition];

do

command...

done

Loops (cont.)

break, continue

- **break** command terminates the loop
- **continue** causes a jump to the next iteration of the loop

Introduction to Variables

$: variable substitution

- If **variable1** is the name of a variable, then **$variable1** is a reference to its *value*.

Pattern Matching

${variable#pattern}

${variable##pattern}

${variable%pattern}

${variable%%pattern}

Examples of Pattern Matching

```
x=/home/cam/book/long.file.name
echo ${x#/*/}
echo ${x##/*/}
echo ${x%.*}
echo ${x%%.*}
```

cam/book/long.file.name

long.file.name

/home/cam/book/long.file

/home/cam/book/long

Aliases

avoiding typing a long command sequence

Ex: alias lm="ls -l | more"

Array

Declare:

declare -a array_name

To dereference (find the contents of) an array variable, use *curly bracket* notation, that is, **${ array[xx]}**

refers to *all* the elements of the array

${array_name[@]}* or *${array_name[*]}

get a count of the number of elements in an array

${#array_name[@]}* or *${#array_name[*]}

Functions

Type

function function-name {
command...
}

function-name () {
command...
}

Local variables in function:

Declare: local var_name

functions may have arguments

function-name $arg1 $arg2

Positional Parameters

$1, $2, $3 …..

$0 is the name of the script.

The variable $# holds the number of positional parameter.

Positional Parameters in Functions

$1, $2, $3….

Not from $0

The *read* Statement

Use to get input (data from user) from keyboard and store (data) to variable.

Syntax:

```
read variable1, variable2,...variableN
```

Example: Write a shell script to

- first ask user, name
- then waits to enter name from the user via keyboard.
- Then user enters name from keyboard (after giving name you have to press ENTER key)
- entered name through keyboard is stored (assigned) to variable fname.
- Solution is in the next slide

Example (*read* Statement)

$ vi sayH

```
#
#Script to read your name from key-board
#
echo "Your first name please:"
read fname
echo "Hello $fname, Lets be friend!"
```

Run it as follows:

```
$ chmod 755 sayH
$ ./sayH
```

Your first name please: ***vivek***

Hello vivek, Lets be friend!

Wild Cards

Wild card /Shorthand	Meaning	Examples	
*	Matches any string or group of characters.	**$ ls ***	will show all files
		$ ls a*	will show all files whose first name is starting with letter 'a'
		$ ls *.c	will show all files having extension .c
		$ ls ut*.c	will show all files having extension .c but file name must begin with 'ut'.
?	Matches any single character.	**$ ls ?**	will show all files whose names are 1 character long
		$ ls fo?	will show all files whose names are 3 character long and file name begin with fo
[...]	Matches any one of the enclosed characters	**$ ls [abc]***	will show all files beginning with letters a,b,c

More commands on one command line

Syntax:

`command1;command2`

To run two command in one command line.

Examples:

`$ date;who`

Will print today's date followed by users who are currently login.

Note that You can't use

`$ date who`

for same purpose, you must put semicolon in between the **`date`** and **`who`** command.

Command Line Arguments

1. Telling the command/utility
   - which option to use.
2. Informing the utility/command
   - which file or group of files to process

Let's take ***rm*** command,

is used to remove file,

which of the file?

how to tail this to ***rm*** command

rm command does not ask the name of the file

So what we do is to write command as follows:

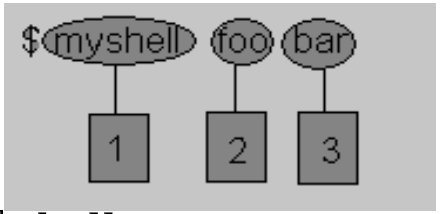
$ *rm* {file-name}

rm : is the command

file-name :file to remove

Arguments - Specification

$ myshell foo bar



1. Shell Script name i.e. **myshell**
2. First command line argument passed to **myshell** i.e. ***foo***
3. Second command line argument passed to **myshell** i.e. **bar**

In shell if we wish to refer this command line argument we refer above as follows

```
1    myshell it is $0
2    foo it is $1
3    bar it is $2
```

•Here **$#** (built in shell variable) will be 2 (Since ***foo*** and **bar** only two Arguments),

•Please note at a time such 9 arguments can be used from $1..$9,

•You can also refer all of them by using $* (which expand to `$1,$2...$9`).

•Note that $1..$9 i.e command line arguments to shell script is know as "*positional parameters*".

Arguments - Example

```
$ vi demo
   #!/bin/sh
   #
   # Script that demos, command line args
   #
   echo "Total number of command line argument are $#"
   echo "$0 is script name"
   echo "$1 is first argument"
   echo "$2 is second argument"
   echo "All of them are :- $* or $@"
```

- Run it as follows
- Set execute permission as follows:

```
$ chmod 755 demo
```

- Run it & test it as follows:

```
$ ./demo Hello World
```

- If test successful, copy script to your own bin directory (Install script for private use)

```
$ cp demo ~/bin
```

- Check whether it is working or not (?)

```
$ demo
$ demo Hello World
```

Redirection of Input/Output

In Linux (and in other OSs also)

it's possible to send output to file

or to read input from a file

For e.g.

$ ***ls*** command gives output to screen;

to send output to file of ls command give command

ls > filename

It means put output of ***ls*** command to filename.

redirection symbols: '>'

There are three main redirection symbols: **>,>>,<**

(1) **>** Redirector Symbol

Syntax:

Linux-command > filename

To output Linux-commands result to file.

Note that if the file already exist,

it will be overwritten

else a new file will be created.

For e.g.

To send output of ***ls*** command give

$ *ls* > myfiles

if '**myfiles**' exist in your current directory

it will be overwritten without any warning.

redirection symbols: '>>'

(2) **>>** Redirector Symbol

Syntax:

```
Linux-command >> filename
```

To output Linux-commands result

to the END of the file.

if file exist:

it will be opened

new information/data will be written to the **END** of the file,

without losing previous information/data,

if file does not exist, a new file is created.

For e.g.

To send output of date command

to already exist file give command

```
$ date >> myfiles
```

redirection symbols: ‘<’

(3) < Redirector Symbol

Syntax:

```
Linux-command < filename
```

To provide input to Linux-command

from the file instead of standart input (key-board).

For e.g. To take input for cat command give

```
$ cat < myfiles
```

Pipes

A pipe is a way

to connect the output of one program

to the input of another program

without any temporary file.

Definition

"*A pipe is nothing but a temporary storage place*

where the output of one command

is stored and then passed

as the input for second command.

Pipes are used

to run more than two commands

Multiple commands

from same command line."

Pipe - Examples

Command using Pipes	Meaning or Use of Pipes
`$ ls | more`	Output of **`ls`** command is given as input to the command **`more`** So output is printed one screen full page at a time.
`$ who | sort`	Output of **`who`** command is given as input to sort command So it will print sorted list of users
`$ who | sort > user_list`	Same as above except output of sort is send to (redirected) the file named user_list
`$ who | wc -l`	**`who`** command provides the input of **`wc`** command So it will count the users logged in.
`$ ls -l | wc -l`	**`ls`** command provides the input of **`wc`** command So it will count files in current directory.
`$ who | grep raju`	Output of **`who`** command is given as input to **`grep`** command So it will print if particular user is logged in. Otherwise nothing is printed

Filter

Accepting the input

from the standard input

and producing the result

on the standard output

is know as a filter.

A filter

performs some kind of process on the input

and provides output.

Filter: Examples

Suppose you have a file

called 'hotel.txt'

with 100 lines data,

you would like to print the content

only between the line numbers 20 and 30

and then store this result to the file 'hlist'

The appropriate command:

```
$ tail +20 < hotel.txt | head -n30 >hlist
```

Here ***head*** command is filter:

takes its input from ***tail*** command

tail command starts selecting

from line number 20 of given file

i.e. hotel.txt

and passes this lines as input to the ***head***,

whose output is redirected

to the 'hlist' file.

Filter: Examples

Consider one more following example

```
$ sort < sname | uniq > u_sname
```

Here ***uniq*** is filter

takes its input from ***sort*** command

and redirects to "u_sname" file.

Processing in Background

Use ampersand (&)

at the end of command

To start the execution in background

and enable the user to continue his/her processing

during the execution of the command

without interrupting

```
$ ls / -R | wc -l
```

This command will take lot of time

to search all files on your system.

So you can run such commands

in Background or simultaneously

by adding the ampersand (&):

```
$ ls / -R | wc -l&
```

Commands Related With Processes

For this purpose	Use this Command	Examples
To see currently running process	**ps**	$ **ps**
To stop any process by PID i.e. to kill process	**kill** {PID}	$ **kill** 1012
To stop processes by name i.e. to kill process	**killall** {Proc-name}	$ **killall** httpd
To get information about all running process	**ps** -ag	$ **ps** -ag
To stop all process except your shell	**kill** 0	$ **kill** 0
For background processing (With &, use to put particular command and program in background)	linux-command &	$ **ls** / -R | **wc** -l &
To display the owner of the processes along with the processes	**ps** aux	$ **ps** aux
To see if a particular process is running or not. For this purpose you have to use ps command in combination with the grep command	**ps** ax | **grep** {Proc-name}	For e.g. you want to see whether Apache web server process is running or not then give command **$ *ps* ax | *grep* httpd**
To see currently running processes and other information like memory and CPU usage with real time updates.	**top**	$ **top** Note that to exit from top command press q.
To display a tree of processes	**pstree**	$ **pstree**

if condition

if condition

used for making decisions in shell script,

If the condition is true

then command1 is executed.

Syntax:

```
if condition then command1 if condition is
   true or if exit status of condition is 0
   (zero) ... ... fi
```

condition

is defined as:

"*Condition is nothing but comparison between two values.*"

For compression

you can use test

or [expr] statements

or even exist status

if condition - Examples

```
$ cat > showfile
   #!/bin/sh
   #
   #Script to print file
   #
   if cat $1
   then
   echo -e "\n\nFile $1, found and successfully echoed"
   fi
```

Run above script as:

```
   $ chmod 755 showfile
   $./showfile foo
```

Shell script name is: **showfile** (**$0**)

The argument is **foo** (**$1**).

Then shell compare it as follows:

if cat $1 :is expanded to ***if cat foo***.

Example: Detailed explanation

if ***cat*** command finds foo file

and if its successfully shown on screen,

it means our ***cat*** command

is successful and

its exist status is 0 (indicates success),

So our if condition is also true

the statement **`echo -e "\n\nFile $1, found and successfully echoed"`**

is proceed by shell.

if cat command is not successful

then it returns non-zero value

indicates some sort of failure

the statement ***echo*** **`-e`** `"\n\nFile $1, found and successfully echoed"`

is skipped by our shell.

test command or [expr]

test command or **[expr]**

is used to see if an expression is true,

and if it is true it return zero(0),

otherwise returns nonzero for false.

Syntax:

test `expression` or **[** `expression` **]**

test command - Example

determine whether given argument number is positive.

```
$ cat > ispostive
   #!/bin/sh
   #
   # Script to see whether argument is positive
   #
   if test $1 -gt 0
   then
   echo "$1 number is positive"
   fi
```

Run it as follows

```
   $ chmod 755 ispostive
```

```
$ ispostive 5
```

5 number is positive

```
$ispostive -45
```

Nothing is printed

Mathematical Operators

Mathematical Operator in Shell Script	Meaning	Normal Arithmetical/ Mathematical Statements	But in Shell	
			For test statement with if command	For [expr] statement with if command
-eq	is equal to	5 == 6	if test 5 -eq 6	if [5 -eq 6]
-ne	is not equal to	5 != 6	if test 5 -ne 6	if [5 -ne 6]
-lt	is less than	5 < 6	if test 5 -lt 6	if [5 -lt 6]
-le	is less than or equal to	5 <= 6	if test 5 -le 6	if [5 -le 6]
-gt	is greater than	5 > 6	if test 5 -gt 6	if [5 -gt 6]
-ge	is greater than or equal to	5 >= 6	if test 5 -ge 6	if [5 -ge 6]

String Operators

Operator	Meaning
string1 = string2	string1 is equal to string2
string1 != string2	string1 is NOT equal to string2
string1	string1 is NOT NULL or not defined
-n string1	string1 is NOT NULL and does exist
-z string1	string1 is NULL and does exist

File and Directory Operators

Test	Meaning
-s file	Non empty file
-f file	File exists or is a normal file and not a directory
-d dir	Directory exists and not a file
-w file	file is a writeable file
-r file	file is a read-only file
-x file	file is executable

Logical Operators

Operator	Meaning
! expression	Logical NOT
expression1 -a expression2	Logical AND
expression1 -o expression2	Logical OR

if...else...fi

If given condition is true

then command1 is executed

otherwise command2 is executed.

Syntax:

```
if condition
then
 condition is zero (true - 0)
 execute all commands up to else statement
else
 if condition is not true then
 execute all commands up to fi
fi
```

if...else…fi -Example

```
$ vi isnump_n
#!/bin/sh
#
# Script to see whether argument is
positive or negative
#
if [ $# -eq 0 ]
then
   echo "$0 : You must give/supply one
   integers"
   exit 1
fi
if test $1 -gt 0
then
   echo "$1 number is positive"
   else echo "$1 number is negative"
fi
```

Try it as follows:

$ chmod 755 isnump_n

$ isnump_n 5

5 number is positive

$ isnump_n -45

-45 number is negative

$ isnump_n

./ispos_n : You must give/supply one integers

$ isnump_n 0

0 number is negative

Loops in Shell Scripts

Bash supports:

for loop

while loop

Note that in each and every loop,

(a) First, the variable used in loop condition

- must be initialized,
- then execution of the loop begins.

(b) A test (condition) is made

at the beginning of each iteration.

(c) The body of loop ends

with a statement modifies

the value of the test (condition) variable.

for Loop

Syntax:

```
for { variable name } in { list }
do
   execute one for each item in the list
   until the list is not finished
   (And repeat all statements between do and done)
done
```

for Loop: Example

Example:

```
$ cat > testfor
for i in 1 2 3 4 5
do

        echo "Welcome $i times"

done
```

- The for loop first creates i variable
  - and assigned a number to i from the list of numbers 1 to 5,
- The shell executes echo statement for each assignment of i.
- This process will continue until all the items in the list were not finished,
- because of this it will repeat 5 echo statements.

Run it above script as follows:

```
$ chmod +x testfor
$ ./testfor
```

for loop - Example

- ```
  $ vi chessboard
  for (( i = 1; i <= 9; i++ )) ### Outer for loop ###
  do
     for (( j = 1 ; j <= 9; j++ )) ### Inner for loop ###
     do
          tot=`expr $i + $j`
          tmp=`expr $tot % 2`
          if [ $tmp -eq 0 ]; then
              echo -e -n "\033[47m "
          else
              echo -e -n "\033[40m "
          fi
    done
   echo -e -n "\033[40m" #### set back background colour to
  black
   echo "" #### print the new line ###
  done
  ```
  ```

while Loop

Syntax:

```
while [ condition ]
do
   command1
   command2
   ..
   ....
done
```

```
while [ $i -le 10 ]
do
  echo "$n * $i = `expr $i \* $n`"
  i=`expr $i + 1`
done
```