2020 JANUARY

Linux Internals → General Public License

MONDAY 20 WK 04 020-346

uname -r } - for linux version

uname -a } - for ubuntu version

Benefits of Linux

1) Free
2) Scalable
3) Portable
4) Secure
5) Run controlling
6) OS & have short dialog time

kernel.org for website visit

TUESDAY 21 WK 04 021-345

Kernel → Shell — (→ takes req. from kernel)

Core of OS
Manages H/w
First program that launched when we power on machine.
1st software loaded into RAM.

Commands & user shell

22 WEDNESDAY

022-344 WK 04

- date
- bash binary again shell.

Kernel — Types
Microlithic
Monolithic

File services — Why its req?
- Create — Read — Module
- Open — Write — Attributes
- Create directories

23 THURSDAY

023-343 WK 04

Device I/O services

Process — flow of code to run program

Dap — client
Ng

Services of Kernel

2020 JANUARY

VFS IPC
PM / TM etc.
MMU
N/W

FRIDAY 24
WK 04 024-342

Virtual File S/m - file service.

Process manag → Multi Processing services

Thread manag - Multi threading services.
1 MMP
2 MP

SATURDAY 25
WK 04 025-341

MMU - Calloc, malloc.
- Memory allocations
- Memory mappings.

Interprocess comm ~~Con~~ tech - pipe
- FIFO
- Msg q's
- shared memory
- sniffers

SUNDAY 26

Threads belongs to one particular application

Su	Mo	Tu	We	Th	Fr	Sa
			1			
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29

27 MONDAY
027-339 WK 05

— N/W Protocol stack.
— Server
— Timer
— Signals
— Synchronization
— ~~Remote Node~~
— Device Drivers

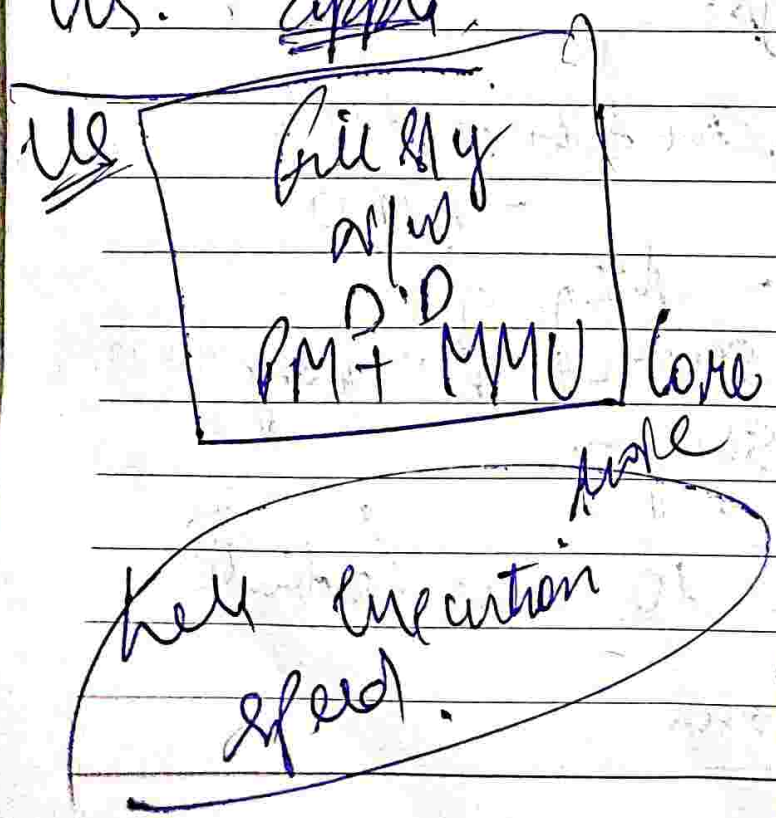
28 TUESDAY
028-338 WK 05

Types of Kernel
— Monolithic Kernel
— Microkernel Kernel
— Hybrid Kernel

Mo

Us. appli.

Us



Linux, Linux, Mac OS, Minix.
Windows OS

Hybrid kernel
Best of Both.
Less speed & Less size.

Vim --version : for Version.

2) /usr/share/vim files

1st Program

vim main1.c
gcc main1.c -o main1
./main1

Present Working directory

31. FRIDAY | 031-335 | WK 05

Compiler process { Pre processor, compiler, Assembler, Linker

Flags used
- E - S - c

gcc

Source code — Main1.c

Preprocessor directive

Preprocessor → comments remove, expand the macro, include header file <stdio.h>

01. SATURDAY | 032-334 | WK 05

.i format

Compiler → Target processor architect

.s

Assembler → ~~fast~~ binary code (Object code)

main1.o

Linker

Executable

02 SUNDAY

Header files
Prototypes,
declarations

Library files
Definitions

gcc -E main1.c -o main1.i

ls -l main1.i

cat main1.i (for viewing the header files taken by the preprocessor)

Assembler

gcc -E -v main1.c
2 times for debugging

cat

gcc -S main1.i -o main1.s

ls -l main1.s

cat

main1.s -o main1.o

file main1.o

Linker Types

05 WEDNESDAY
036-330 WK 06

Static = Dynamic ex. file
Static ex. file (gcc uses this by default)

gcc -static main.c
-o static_main
ls -l static_main
file static_main
ls -lh static_main

gcc main.c
-o main
file main
ls -l main
ls -lh main

06 THURSDAY
037-329 WK 06

RFS :- Service of Linux OS.
- uses memory efficiently
- stores files & dir's in structured manner.

- All files SAM are mounted under RFS at Kernel Boot up time.

020320 Serial device, keyboard, all char. devices

FEBRUARY

FRIDAY 07

WK 06 | 038-328

from device driver call with high data

→ deadlock, CP, BP,

→ block;

etc

↓

Cat/Stty/Pwd

for save & exit

SATURDAY 08

WK 06 | 039-327

Proc

Sys

logical file sys.

Process

DP to delete

To move L, R, N, D

Left

U - right

U - K

D - J

SUNDAY 09

MAR '20

Su	Mo	Tu	We	Th	Fr	Sa
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

performance particular char.

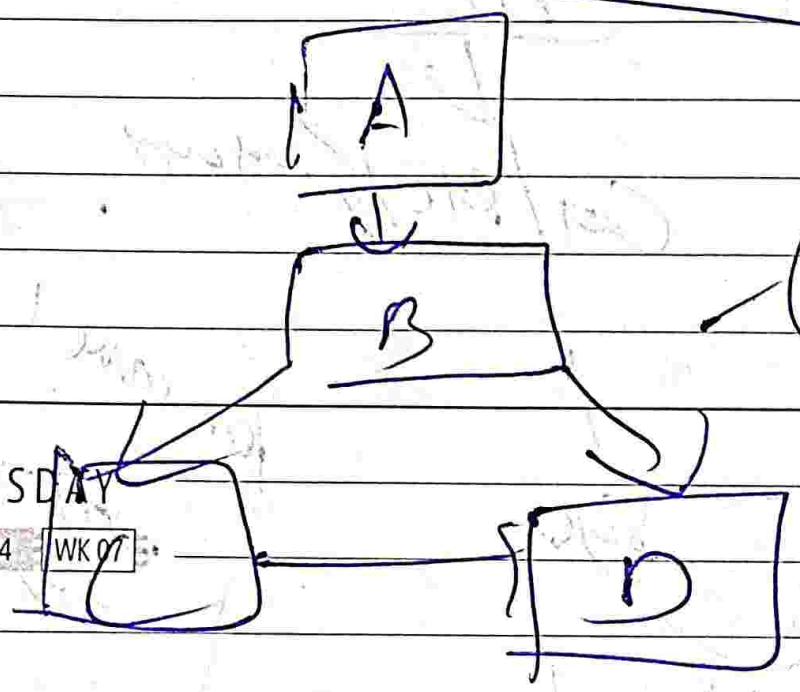
A — To add delete in end of line

10 MONDAY

041-325 WK 07

Self

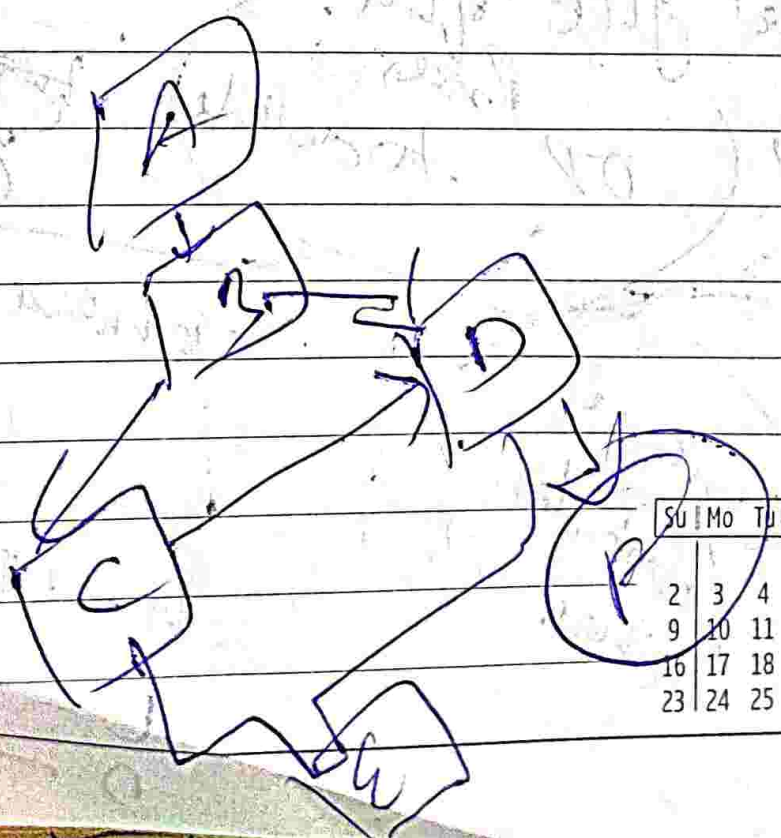
Keep creating hybrid classes in C++



5 Nb

11 TUESDAY

042-324 WK 07



Linux { GNU Binary utilities } 8-3-2L

FEBRUARY

2020

WEDNESDAY 12

WK 07 043-323

Tools :- Objdump & Readelf

bin → hex file

exe → assembly

disassemble

Objdump - D exfile

date

main

objdump - J . ~~text~~ - D ~~exfile~~

Readelf . - Read segment

THURSDAY 13

WK 07 044-322

In linux file OS all are located with non-ve integers.

0.0 Can't be -1.

Open (Const char *pathname, int flags, mode_t modes)

MAR '20	Su	Mo	Tu	We	Th	Fr	Sa
	1	2	3	4	5	6	7
	8	9	10	11	12	13	14
	15	16	17	18	19	20	21
	22	23	24	25	26	27	28
	29	30	31				

Kernel Data types

14 FRIDAY
046-320 WK 07

library call vs System call
↓ ↓
prov. by a standard lib prov. by kernel by linux s/m.

Append :- adds data at end of file.

mode_t
↓
wrapped around integer
System dependent one

In linux we can open 1024 files from single file.

15 SATURDAY
046-320 WK 07

File descriptor :-

```
#include <stdio.h>
    <unistd.h>
int main () {
    printf ("main pid : %d \n",
    getdtablesize();  // max no. of
    exit (0);
}
```

16 SUNDAY

Same - t, read

2020 Current fileoffset is a measure that measures the no of byte from the beg. of the file.

FEBRUARY MON 17 WK 08 049-318

System of write s/m call is same as of read s/m call.

write s/m call returns no. of bytes written to file.

open a file get the file des. & apply read s/m call.

FEBRUARY TUESDAY 18 WK 08 049-317

Q Open a file O_create operation
Take 2 buffer — Read buffer (empty)
— Write buffer (data)
Now call write operation
Then call read buffer. Then print

Q Lseek s/m call performs repositioning of your current file offset location.

lseek (int fd, off_t, value, int whence)

int

MAR 20
Su Mo Tu We Th Fr Sa
1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31

When C's J

19 WEDNESDAY 050-316 WK 08

SEEK_SET - at the beg of file + no. of value

SEET_CUR

SEEK_END

Validate read & write request

read (fd, buf, 100);

so

REQ < MAX_AVLB_SIZE

20 THURSDAY 051-315 WK 08

In most cases read func is going to return less no. of bytes than actually made request for.

Common S/m calls in linux

Common API's.

+9 — Success

-1 — failure

Types of Blocks

FEBRUARY

1. Boot (Only 1)
2. Super (Few)
3. Inode - (Type, name, size, permission, struct, file of, Pointer, Pt, data blocks) (More)
4. Data

FRIDAY 21

WK 08 052-314

Will have booting info when you on the s/m. , Booting cycle

In order to maintain a single file kernel is mount. 2 ways.

Linux allows file sharing among multiple processes.

SATURDAY 22

WK 08 053-313

Min. data block size - 512 Bytes

Q&A

SUNDAY 23

Su	Mo	Tu	We	Th	Fr	Sa
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

FEBRUARY 2020

24 MONDAY 055-311 WK 09

Manipulate API

Const Dest. Why (new, delete) you can use

(Constructors) Syntax

① class A {
public:
int x; // constructor
A() { } // object initialization
};

25 TUESDAY 056-310 WK 09

② Inside class ~~function~~ definition or outside using class name & scope resolution :)

class A {
public:
int i;
A();
};
A::A() { i=1; }

declare

Su Mo Tu We Th Fr Sa
2 3 4 5 6 7 8
9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
FEB '20

copy parameterised

WEDNESDAY 26 WK 09 057-309

Dup s/m call creates a duplicate file descriptor copy.

int dup (int fd) (3) (4)

fd

fcntl (int fd, int cmd)

F_DUPFD

THURSDAY 27 WK 09 058-308

3 ways to dup: file descp fd, dup, fcntl.

Status flags :- Read, Write only

A pg which is in execution is a process.

Linux can run multiple programe, mul. Process concurrently.

At some point of time in order to perform multitasking operation. At any given pt of time tone of application

FEB/MAR

28 FRIDAY

059-307 WK 09

running in your machine

To view that command is {ps -aux}

{ps aux}

Each & every process have a unique PID. {ps -elf} is given more inf.

Processes are isolated from each other. They run in independent environment

29 SATURDAY

060-306 WK 09

Process — I/O, Memory, CPU

No process will interfere into other processes.

PID (Process identification no)

Each & every process will have a unique identifier & thats allocated by the OS.

01 SUNDAY

Su Mo Tu We Th Fr Sa
1
2 3 4 5 6 7 8
9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
FEB 20

Pid==0 — core kernel process
Swapper process

MONDAY 02
WK 10 062-304

Pid = 1 — Init process } Act like a parent

getpid() :- takes nothing returns the current process.

getppid :- will return parent process id.

TUESDAY 03
WK 10 063-303

Each & every process will have a resp. parent".

'1.out' :- Bash is furnishing it (is acting like a parent.

Pstree — command

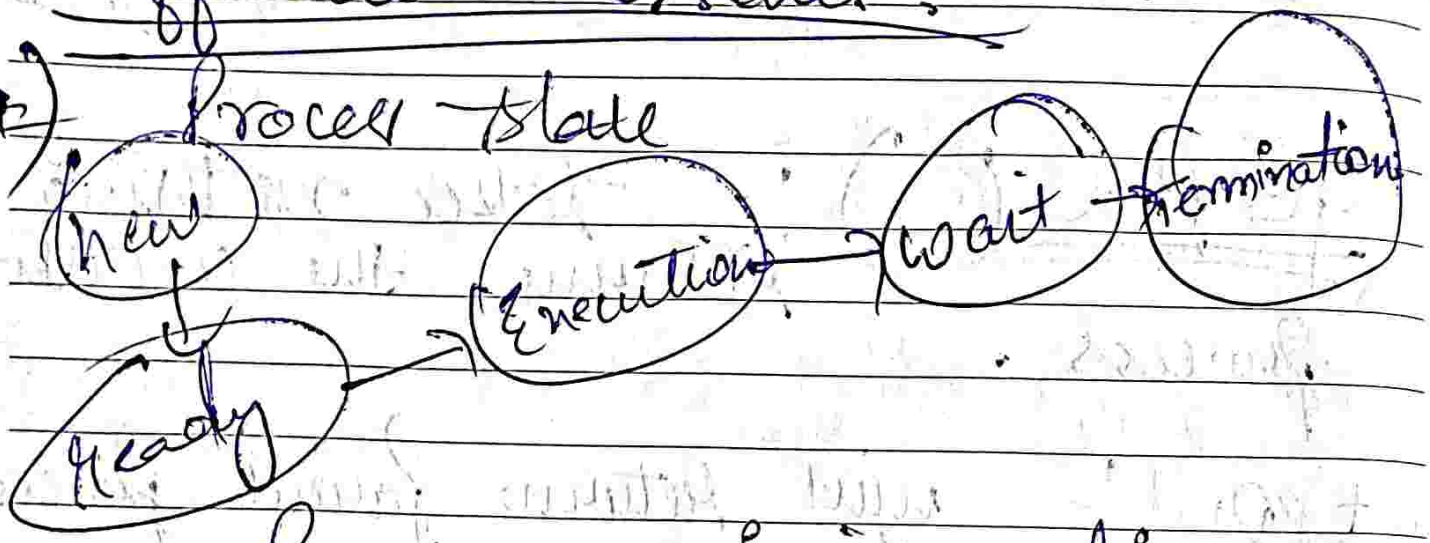
Ctrl C :- H/w interrupts generated by a key & notified by kernel & process to the calling process.

	Su	Mo	Tu	We	Th	Fr	Sa
M							
A	1	2	3	4	5	6	7
R	8	9	10	11	12	13	14
	15	16	17	18	19	20	21
	22	23	24	25	26	27	28
	29	30	31				

On reception of a S/g the calling process here to terminated.

Process is going through 5 different states :-

→ Process state



Readyqueue :- is a linked list of processes that are ready to get load & execute.

Scheduler takes but process in time of priority.

Priority :- Ranking

Waitqueue :- is a linked list of processes that are suspended & waiting for execution.

Ready queue / for CPU
Wait queue / for resource

Creation :-

```
# include <unistd.h>
fork (void);
```

* Linux s/m provides an API called as fork to launch a new process from the current task.

* fork is widely used in concurrent applications.

* Shell is also using fork to launch a new process.

* So after fork s/m call scheduler decides which process to get execute first either a parent or child depending on head of the s/m.

fork num 0

MARCH 2021

09 MONDAY 069-297 WK 11

n - no of times fork called.

Copy on write :- Parent process address space segments are copied toward child process of the same segment.

fork :- Executes only Returns value twice

* fork *

10 TUESDAY 070-296 WK 11

Parent child

By using return values we can manipulate the code.

call fork manipulate Parent & child code, from child call fork.

Su	Mo	Tu	We	Th	Fr	Sa
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

(7+) Multiple - more than 2 8-3-22
Multilevel - (3) (4)

MARCH

WEDNESDAY 11
WK 11 071-295

Polymorphism :- One entity diff. roles.

Test your Skills.

Pointer to derived class

Hum 10-3-22

THURSDAY 12
WK 11 072-294

Diff b/w fork & vfork

The moment process calls excel process loads a brand new pg from the hard disk into the process at the space & starts executes the new pg.

- No pid generated throughout process.

Pg where parent & child joins and use excel to execute any pgs.

Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

MARCH 2020

Implement a P/g ~~[illegible]~~ exec P/g to load and run P/g.

13 FRIDAY

073-293 WK 11

(Ls-l)

Differed routines :- The func^n that are executed after some time out or a delayed execution

At exit takes func. as argument & registers with the kernel to execute later.

14 SATURDAY

074-292 WK 11

It executes on :- normal process ~~exec~~ execution.

— Returns (0)

— Exit

15 ~~SUNDAY~~ Implements default routines.

At exit register the process with kernel. On normal termination executes functions in reverse order of their registration.

- Max 32 func. can be registered using at exit function.

- Takes nothing & is expected to return nothing.

Syntax:

The main normal func. is taking place at exit is to call,

At exit func. calling the reg. func which are previously reg. for a process by atexit func. & executes in the order of reverse order of their registration.

- exit(0) func. will not call the func. which are registered at atexit func.

Kernel processing priority:

Higher no higher priority

Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

18 WEDNESDAY 078-288 WK 12

NI :- Nice value which can be on der. priority of a process

SZ :- Size of process

WCHAN :- Address in kernel space where process is going to sleep.

TTY :- terminal from where the process started.

19 THURSDAY 079-287 WK 12

CMD :- that gen. process.

Wait will return the exit status of terminated child.

In third case it returns error.

Wait has 3 behaviors

Block hang

Disadvantages of Fork

FRIDAY 20 WK 12 080-286

To maintain & create process allocates lot of resources & very expensive in terms of CPU, memory & I/O resources.

THREADS

Creating, maintaining & destroying threads are cheaper when compared to process.

SATURDAY 21 WK 12 081-285

Threads are called as lightweight processes, needs only few resources.

Thread is a "Context of execution"

"Set of inst. that are executing to perform a particular task"

SUNDAY 22

MARCH

Multithreading: Parallel threads of execution.

23 MONDAY
083-283 WK 13

Sets of inst. that are executing simultaneously.

Each thread performs a specific task.

Threads are very well employed on schedulers.

Thread Resources

24 TUESDAY
084-282 WK 13

- Stack region
- Registers
- Specific data
- Own scheduling policy & priority
- Own Signals (seek, block, seek mask sig).

Linux Thread model :- 1-1 thread model

Threads don't carry their address spaces

Su Mo Tu We Th Fr Sa
1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31

Thread Classification

2020 MARCH

User level — Kernel level

WEDNESDAY 25

Advantages Thread :- 1) Do not carry their own address space.

2) They share other space of the process.

3) Comm. b/w process & threads takes place by default.

Disadv. :- If atleast 1 thread is carrying a bug then entire threads & applications is crashed.

THURSDAY 26

WK 13 | 086-280

Linux → Clone() → Pthreads → Gaming app

Windows

IEEE has launched a std. Posix.

(Portable OS interface)

implemented by set of functionalities

APR '20

Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

MARCH 2020

that we need to implement portable appln. on other OS.

27 FRIDAY 087-279 WK 13

POSIX THREAD CREATION

Data types

Data type :-

pthread_t xxx

Jnum's

28 SATURDAY 088-278 WK 13

pid_t - talk about pid of process

pthread_t → thread id the end object

29 SUNDAY Thread gets its own attributes.

Su	Mo	Tu	We	Th	Fr	Sa
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

MAR 2020

Thread creation function

MARCH

MONDAY 30

WK 14

include <pthread.h>

pthread_create (pthread_t *pt, pthread_attr_t *attr,

Immediately on success executes thread task function.

On failure returns non-zero value.

Error code

TUESDAY 31

WK 14

2 measure non zero error code

EAGAIN → When you don't have sufficient res. to create a thread.

EPERM → sys don't have enough

↓

Have my permissions.

APR 20	Su	Mo	Tu	We	Th	Fr	Sa
				1	2	3	4
	5	6	7	8	9	10	11
	12	13	14	15	16	17	18
	19	20	21	22	23	24	25
	26	27	28	29	30		

Diff b/w compile time & run time Polymorphism

01 WEDNESDAY

Virtual function
Operator Overloading
Function Overloading

Compile time Polymorphism OR Early Binding	Run time Polymorphism OR Late Binding

Run virtual function

02 THURSDAY

093-273 WK 14

11-3-22

Pthread returns the calling id.
Pthread id is same value i.e stored to pointer pthread_t in pthread create function.

Pthread is used to terminate other called threads using thread id.

Using pthread_cancel we can terminate the calling id.

A thread task func. can be shared by 2 diff. threads.

Pthread Attributes (Properties)

Attributes define the behaviour of thread. Once your thread is created how it should behave.

Classification

- Stack management
- Thread Synchronization
- Pthread Scheduling policy & priority
- Thread specific data
- Thread stg management

func

Scenario of Appln s/w with 10 diff threads

Void func. can be exec. using pthread.

2020 APRIL FRIDAY 03 SATURDAY 04 SUNDAY 05
MAY '20 Su Mo Tu We Th Fr Sa

APRIL

06 MONDAY 097-269 Wk 15

* Whenever you want to change the default attributes of your thread:

1) Declare an object of pthread_attr_t type of attribute t.

2) It's not appropriate to have garbage value so we initialise attribute object.

pthread_attrib_init (pthread_attr_t *pa)

07 TUESDAY 098-268

3) Call appropriate thread attribute functions.

4) Create a thread of your stack attributes.

5) Destroy the thread.

No threads can share a stack.

No threads can point to same thread stack.

Su Mo Tu We Th Fr Sa
5 6 7 8 9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30

Thread alloc - set stack :- Reading / Writing

Thread Synchronisation

-) Critical Section :- The inst are a piece of s/w i.e accessing with global shared resource. Its always unsafe to execute critical section by 2 threads simultaneously. Otherwise, leads to race cond^n.

To avoid race cond^n linux kernel has provided various synchronisation techniques called as Synch. locks.

Various Techniques are :-

1) Semaphore locking technique.
2) Spin lock "
3) Mutex "
4) Atomic "

APRIL

10 FRIDAY 101-265 WK 15

optimized for low contention cases.

3) → Semaphores can't be integrated with interrupts {ISR}

→ There's no issue of priorities.

Spinlock & Semaphore

2 imp locking techniques.

11 SATURDAY 102-264 WK 15

Not used with interrupts

2) Spinlock :- Contention is very expensive in terms of kernel resources.

Interrupts can be integrated to spinlocks.

12 SUNDAY

- Should be fast enough to execute.

Can't afford sleep, block.

Su Mo Tu We Th Fr Sa
1 2 3 4
5 6 7 8 9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30
APR

Semaphore lock

APRIL

MONDAY 13

WK 16 104-262

1) Declare the Semaphore object.

Sem_t , my_sem

2) Initialise the Semaphore object.

Sem_Init (S my_sem 0)

non zero — shared b/w process.

0 — sharing b/w thr of same process

TUESDAY 14

WK 16 105-261

Process Operation in Semaphore :-

1) Process - take token from S/P.
2) Use the resource.
3) Give the token to S/P

(HPES)

"Binary Semaphore" : A Semaphore whose Token value changes b/w 1 & 0 & 0 & 1 during the operation called as a binary semaphore.

Mutex enterp. semaphores are employed to the ~~mutx~~ found Mutex

	Su	Mo	Tu	We	Th	Fr	Sa
						1	2
	3	4	5	6	7	8	9
	10	11	12	13	14	15	16
	17	18	19	20	21	22	23
	24	25	26	27	28	29	30

APRIL 2020

15 WEDNESDAY 106-260 WK 16

Counting Semaphore :- Whose token count value 1 is counting semaphore.

Pthreads & Spinlock

1) pthread_spinlock_t my_spin;

2) pthread_spin_~~lock~~ init (pthread_spinlock_t int_shared)

Public — pointer of same arguments

Private

16 THURSDAY 107-259 WK 16

Content Switching :- Switching of CPU from one process code to another task.

Deadlock :- neither of the process moves forward. Each process is waiting for other process to release the lock.

Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

APR '20

are very similar to ~~binary~~ semaphores with a usage point of 1.

- Process are threads unlocked with mutex.
- locking & unlocking done from the same context.
- Process can not exit while holding a lock. It has to be unlocked for execution.
- Can't be used with interrupts.

Pthread Mutex

Declare a variable of type :-

pthread_mutex_t my_mutex

Attributes

Initializing a pthread_mutex object.

Su Mo Tu We Th Fr Sa
31 1 2
3 4 5 6 7 8 9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30

APRIL

20 MONDAY 111-255 WK 17

Two ways of initialising

- Static way
- Dynamic way

Adv:-

- During run-time we can change the properties of a mutex object.

- Static initialisation :- mutex attributes can't be changed throughout the pkg. They are fixed till the program.

21 TUESDAY 112-254 WK 17

There are two Semantics :

pthread mutex lock
pthread mutex try lock

- If process fail to acquire a lock using pthread mutex lock then the particular process thread goes to a block state. If a process thread failed to acquire the mutex

pthread mutex try lock

Su Mo Tu We Th Fr Sa
5 6 7 8 9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30
'20

Unlocking function

pthread_mutex_unlock

pthread_join : make the parent to wait

pthread_detach : put the thread into detachable state
pthread_self : gives thread id to.

Once thread terminate don't need to wait for parent.

pthread_once : is to implement initialisation code.

pthread lib :- provides a data type.

pthread_once_t.

provides a macro to pthread_once_t.

pthread_once_t

once = pthread_once_init.

APRIL

24 FRIDAY 115-251 WK 17

pthread_once :- The movement pthread once executed checks for the 1st argument whether arg is initialised or not. If its initialised it will schedule & initialise the arg (2nd arg)

If not initialised will never schedule the function.

After execution of initialisation code the value is getting changed. then pthread comes to picture & executes pthread_once.

25 SATURDAY 116-250 WK 17

pthread once schedule & initialise a function of type which takes no argument & returns nothing. & also takes nothing

26 SUNDAY

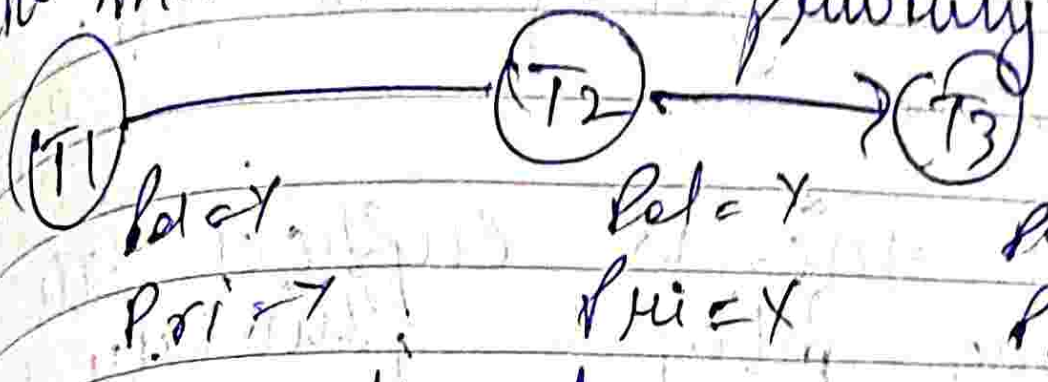
Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

APR '20

inherit scheduler member

Sch. policy & priority

MONDAY 27



When a thread creates a new thread, newly created thread will inherit the scheduling properties of its parent thread. This is acc. to posix thread library. Newly created thread by T2 i.e T3 does not have same pol & priority. So how to do it

↓ There's a one member called as inherit scheduler member that specifies us whether a thread is inheriting scheduling policy & priority from its parent or managing on its own.

TUESDAY 28

WK 13 | 119-247

inherit scheduler can be or from either T1 or T2.

MAY 2020
Su Mo Tu We Th Fr Sa
31 1 2
3 4 5 6 7 8 9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30

APRIL

29 WEDNESDAY

120-246 WK18

i.e → Pthread_inherit_sched
→ Pthread_explicit_sched

Inherits owning from the parent i.e getting the policy & priority.

→ If this is explicit the having policy & priority of its own.

→ Pthread_attr_get_inherit_sched for reading

30 THURSDAY

121-245 WK18

→ Pthread_attr_set_inherit_sched for setting.

LINUX THREAD SCHEDULING PRIORITY & POLICIES :-

Priorities from - 1 to 99
lowest highest

Priorities comes to picture only w.r.t policy.

Policy — Scheduler

Default sched algo

Threads have same priority, CPU time sharing

Round Robin Sched RR

No priority, all threads will get timeslice in circular order

FRIDAY 01

WK 18 | 121-245

Which thread just will get

Real time thread policies.

Which one time just will decide which how thread first entry into run keep.

SATURDAY 02

WK 18 | 123-243

S/m scheduler provides CPU slice time.

struct sched_param
{
int sched_priority;
}

SUNDAY 03

Struct with only element

MAY 20	Su	Mo	Tu	We	Th	Fr	Sa
	31					1	2
	3	4	5	6	7	8	9
	10	11	12	13	14	15	16
	17	18	19	20	21	22	23
	24	25	26	27	28	29	30

MAY Preemptive vs Non Preemptive Scheduling

04 MONDAY 125-241 WK 19

A Process is removed by Kernel forcibly transferring CPU from a process w/o requesting the process.

All modern OS has this.

Process will never leave CPU on an interruption. It will only when a task is done or timeslice of a process expires.

05 TUESDAY 126-240 WK 19

Linux employs "complete fair scheduler" (CFS Scheduler).

Each & every process will get proportions of your CPU based on factor called as —

NICE VALUES.

↓

NICE gives weightage to the process w.r.t CPU slice time.

Su Mo Tu We Th Fr Sa
1 2
3 4 5 6 7 8 9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
MAY 2024

Nice value used for prioritizing

inc & dec the priority of a process.

MAY WEDNESDAY 06 WK 19 127-239

NICE VALUE is same for all process each & every process will get a timeslice of

$\frac{1}{n} \times$ CPU Time.

n - no. of processes called

NICE VALUE TABLE THURSDAY 07 WK 19 128-238

Nice value	Priority	CPU slice time
-20	Highest	CPU usage. 85 ms
0	Default	10 ms
19	Lowest	5 ms

JUN

Su	Mo	Tu	We	Th	Fr	Sa
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

This called fair scheduling

∴ The lowest priority get timeslice also.

MAY

Command taskset -c

08 FRIDAY 126-237 WK 19

will assign a CPU core to a process. The utility tool comes by default with linux distribution.

-c :- for ~~changing~~ setting CPU core

1

b :- is a special background expression forces the following command execute in the program.

09 ~~SATURDAY~~ 127-238 WK 19

Schedule Algorithm

O(n)

O(1)

SJF :- Shortest job first

10 SUNDAY

FIFO

RR

CFS

Su	Mo	Tu	We	Th	Fr	Sa
31					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

mice1 & touset -c 1 . /mice1

MAY

MONDAY 11

WK 20 132-234

2020 & Process 1 Process 2

top :- to analyse.

RR :-

Each & every process will get equal proportions of CPU slice time in circular order. If a process has finished the given time slice CPU moves to next process.

TUESDAY 12

WK 20 133-233

Each & every process use CPU & resources for same amount of time.

No issues of priority - implements starvation free process. Each & every thread have to be executed in circular order. Each thread will use the resource for same amount of time

JUN 20
Su Mo Tu We Th Fr Sa
1 2 3 4 5 6
7 8 9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30

~~MAY~~ Process content - LINUX.

Kernel content

13 WEDNESDAY
134-232 WK 20

Interrupt content

At any given point of time Kernel will be executing either in any of the 3 contents:

Process content :- (by default)

Kernel executing a piece of s/w on behalf of a process which has initiated the s/m call.

14 THURSDAY
135-231 WK 20

Kernel content :-

Kernel executing piece of s/w (a Kernel service) on behalf of another Kernel service.

Only in high priveleged regions they can access.

Interrupt content :-

Kernel exec. piece of s/w on behalf of interrupt service.

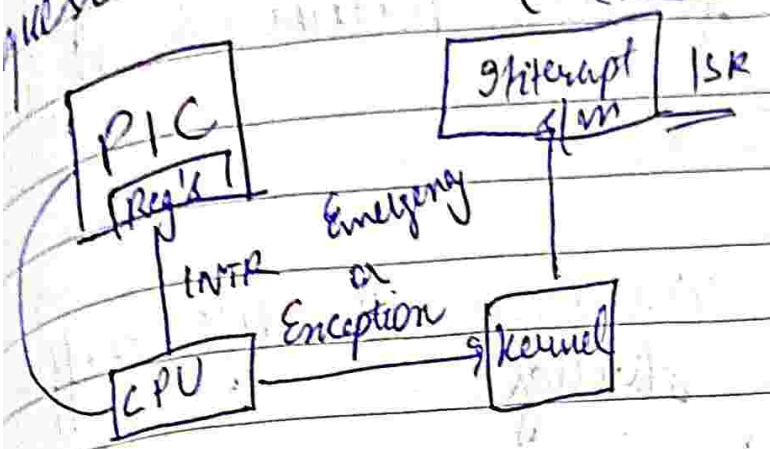
Su Mo Tu We Th Fr Sa
31 1 2
3 4 5 6 7 8 9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
MAY '20

PIC controller.

MAY

FRIDAY 15

Each & every device on a s/m architecture is connected to interrupt controller through a physical wire/bus called as interrupt request lines (IRQ lines)



SATURDAY 16

Kernel exec. a piece of s/w on behalf of an interrupt is called as interrupt content.

Memory manipulation Calls #

memset (void * address, int/char c, size_t n bytes)

SUNDAY 17

When Process executes memset function jumps to the address location provided by * argument & start set the data in with a given constant c for n no. of bytes given by 3rd argument

MAY

2) Memchar :- Same syntax of

18 MONDAY 139-227 WK 21

The mom. Process executes memchar func. Jumps to the address location provided by pointer argument & starts scanning for the given constant c. If the constant found returns # no of bytes.

3) Memcmp :- compare 2 diff buffers/strings.

19 TUESDAY 140-226 WK 21

memcmp (void * addr, void * addr2, size_t, n bytes)

mem return ASCII value & compares data on ASCII value & gets diff.

The mom. Process executes memcmp memory function it jumps to 2 address location & start comparing the address byte by byte until it gets unmatching data. & returns either +1, -1 or 0.

Su	Mo	Tu	We	Th	Fr	Sa
31					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

MAY '20

4) Memmove : source is pointed to buffer.

kind of memcpy.

MAY WEDNESDAY 20 WK 21 141-225

void * memmove (void * dstn, void * src, size_t, nbytes)

The mem. process executes memmove operation copies the data from source buffer to destination buffer for given n no. of bytes.

If destn have n no of data gets overloaded

MAY THURSDAY 21 WK 21 142-224

Memmove finally returns destn.

5) Memcpy :-

void * memcpy (void * destn, void * src, size_t, nbytes).

	Su	Mo	Tu	We	Th	Fr	Sa
		1	2	3	4	5	6
JUN	7	8	9	10	11	12	13
	14	15	16	17	18	19	20
20	21	22	23	24	25	26	27
	28	29	30				

MAY

Memmove
Slow
Reliable (gen. guaranteed data.

Memcpy
Fast
Not reliable

No this sup. in this source fnc. }

22 WEDNESDAY

Use a temp. buffer.
In case of overlapping } its useful.

Alloc a :-

void * alloca(size_t , nbytes).

23 SATURDAY 144-222 WK 21

The mem. process executing alloc a executes memory from stack segment.
{ Heap seg is full in that case }

& returns pointer to allocated region on success.

24 SUNDAY

Slight adv. in comp to malloc.
less memory wastage.

Su	Mo	Tu	We	Th	Fr	Sa
31					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

M A Y 20

malloc uses a pool of block sizes & multiples of 16 & calls free pointer whereas alloc allocates as per need.

Alloca :- 1) Min. memory wastage
2) Fast operation as of malloc
3) Does not maintain pool of fixed block sizes & hence no memory fragmentation.
4) No need to call "free" for memory call. The memory is auto deallocated when func. ends.

5) If repetitive alloca are made stack & heap gets full.

SWAPPING PROCESS

When RAM is running out of memory & OS wants to launch a new appln. to the RAM then OS looks for the inactive processes from the RAM & push them into swap partition of your storage device (Hard disk) & is called as (Swap out) process.

Su	Mo	Tu	We	Th	Fr	Sa
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

When new launched app have done job & kernel out of memory kernel will get back process from swap partition to RAM. process & is called Swap in

MEMORY LOCKS

There could be a login p/g & we don't want kernel to swap out my login process from RAM. Then we can apply memory locks.

Memory locks guard your address is safe from swapping

TYPES :-

~~Munlock~~ int mlock.. (void * address, size_t, nbytes);

int mlockall (int flags);

mlock ( )

0 — ENOMEM

returns on success — EAGAIN

ENOMEM :- The error returned by lock while trying to apply mlock more than permitted limit will return a error called ENOMEM.

EPERM :- No Privilege then it return.

EAGAIN :- If it failed to apply a memory lock for the specified address.

Mlock_all :- uses

MCL - Current flag mark & current process address spaces lock.

Process segment	Program data segment				
text	heap	bss	Program break point	stack	

JUNE 01 MONDAY 153-213 WK 23

P/g data s/g = data s/g + bss s/g + heap

or P/g breakpoint or end address of P/g data segment.

sbrk

void * sbrk (size_t, nbytes);

sbrk (10);

(ie moving ahead with 10 & create a new p/g breakpoint

JUNE 02 TUESDAY 154-212 WK 23

returns a void ptr i.e new address.

sbrk s/m call that manages end of p/g data s/g.

brk

int brk (void * end_addr);

int as return value.

both are brk & sbrk used to adjust p/g breakpoints. memory data

segment value

2020 by which creates a JUNE

new flg breakpoint & brk WEDNESDAY 03

func. takes an desired addr WK 23 155-211

to create a new flg breakpoint.

Memory mapping :- allows direct memory access to a user space device

mapping :- taking a region (file/memory/kernel memory) & append it.

Q Write a small func. of your own memcpy w/o using memove?

Problem with user & kernel space # THURSDAY 04

WK 23 156-210

mmap :- If an appl making repetitive I/O requests a much of CPU time spent in submitting I/O operations.

Solution :- mmap is a form memory function that maps a given kernel file region or a device region on some random kernel memory into the calling process address space.

JUL '20

Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

mmap Syntax :-

JUNE

05 FRIDAY 157-209 WK 23

```
#include <sys/mmap.h>
mmap (void *addr, size_t bytes, int protection, int flag, Fd|-1, off_t offset)
```

- void * addr → where in my process address space it will map.

size_t → no. of bytes that we want to map.

int protection → PROT_EXEC, PROT_READ, PROT_WRITE

PROT_EXEC → Pages maybe executed

PROT_READ → Pages maybe read

PROT_WRITE → Pages maybe write.

06 SATURDAY 158-208 WK 23

int flag → MAP_LOCKED
→ MAP_SHARED
→ MAP_PRIVATE
→ MAP_ANONYMOUS

SUNDAY

Su	Mo	Tu	We	Th	Fr	Sa
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

JUN 20

MAP_ANONYMOUS] It does n't belong to any file. & 5th arg supp. to be -1 & 6th arg to be 0.

JUNE MONDAY 08 WK 24 160-205

Mmap shares shared memory reg.

Map_locked :- is equal to applying mlock to shared region.

Map_shared :- shared mem. is now shared b/w n no. of processes.

TUESDAY 09 WK 24 161-205

Map_private :- create a copy private one & changes will not reflected to other processes which are mapping the shared region.

addr arg = 0 : Its recommended to use 0 as address arg. 0 indicates asking kernel map it at free process address space.

Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

JUL 20

fd | -1 | :-

LINUX MMU

High level MMU — independent of process archit.

Low level MMU — dep. on process archit.

dep on

x86 ARM

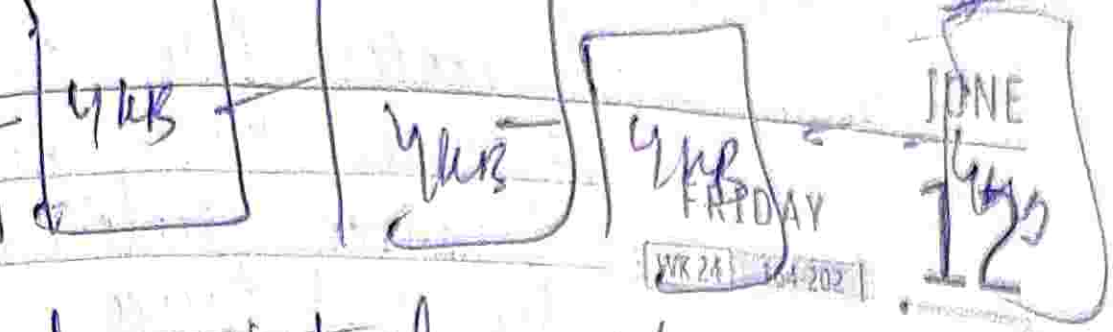
Low MMU

When Kernel booting starts low level MMU also starts & creates initialising lot of kernel memory data structures {struct page}

~~Creating~~ Low MMU conv. entire memory into struct page. Entire memory is a linked list of struct pages.

Each page is of 4 KB.

Struct page size : 32 bits



logical virtual add :- (pg no (100) + offset (500))

Struct Page :-

32 bit

12 [0- 4095] i.e. 4KB

20 {Page no's}

Each & every process maintains one page table entry inside the PCB of the process. If ref. to the "PTE" is lost kernel will never know where the pages of process are in memory of kernel. PTE identifies these particular pages belongs to X particular process.

Inside the kernel there's a dynamic DS table called as page frame relation table.

Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

For every read/write I/O operation processor has to perform logical to physical addr translation.

Processor will take the page no. and look for matching page frame no. in page table. After this, now fetches page frame no for page no. Now takes the frame no & adds to offset of integer variable x.

Now, gets the physical address.

This entire process is called as "Virtual memory address translation."

JUNE

15 MONDAY

167-199 WK 25

16 TUESDAY

168-198 WK 25

Swap Partition has two virtual partition

VIRTUAL MEMORY :- A section of memory created temporarily on swap partition of storage devices.

When user RAM & virtual memory & virtual addresses are assigned to the processes.

Su	Mo	Tu	We	Th	Fr	Sa
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

JUN '20

free : [illegible] of virtual memory

JUNE WEDNESDAY 17

Provide amount of available virtual memory & virtual memory used by the s/m.

vmstat -s :- lists all stats

cat /proc/meminfo current virtual memory report.

JUNE THURSDAY 18

sysm stat :- complete virtual memory statistic reporter.

(b) :- uninterruptable process.

(free) :- free available virtual memory.

(si) :- Swap in (moving back to RAM)

(so) :- Swap out.

(bi) :- blocks rec. from the block device

(bo) :- blocks sent to block device.

(In) : no. of interrupts per sec

JUNE

(CS) : no. of context s per execution.

19 FRIDAY

171-195 WK25

LIBRARY :- group of precompi object files.a

XPES.

Static :- are one which are statically linked to pg executable from compile time.

Dynamic :- lib are dynamically lin to pg executable fil at runtime.

20 SATURDAY

172-194 WK25

Linux OS static library ext is .a

ar :- linux command line tool called as archive

21 SUNDAY

rcs :- replace create symbol.

-L. :- linking the library lib from current directory

Su Mo Tu We Th Fr Sa
1 2 3 4 5 6
7 8 9 10 11 12 13
14 15 16 17 18 19
21 22 23 24 25 26 27
28 29 30
'20

-fPIC :- PIC - Position Independent code.

JUNE

MONDAY 22 WK 26

gcc -c calc_mean.c calc_sub.c -fpic

mean.o sub.o

.so — (Shared objects) ext. for dynamic lib.

lib.so — should be developed independently

[binun internal]

Dynamic lib are also called shared libraries.

TUESDAY 23 WK 26 175-191

Relocatable: should get load to any memory space.

Dynamic lib uses a flag called [PIC] → position independent code.

Dyn. lib should be relocatable. That means the inst. of dynamic lib should ready to get load into any memory address space.

Executable file

A process needs a dynamic lib and address it.

JUL '20
Su Mo Tu We Th Fr Sa
1 2 3 4
5 6 7 8 9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31

dyn. lib. merge with the flg in such a way that dyn. lib. appears to be a part of the executable file.

later, when another exec. file requires the dynamic lib. the inst. of the library should merge with the memory of the exec. flg. So that it should get ready to reload.

ef — shared is a flg used to generate dynamic lib.

ldd :- linux command line tool can provide the dependencies req. for the executable file.

Processor

Real mode

Protected mode

series / array of bytes

Su	Mo	Tu	We	Th	Fr	Sa
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

While your processor is executing
of all dep. on proc. how it is
looking at memory. Based on that
OS will form a policy to manage
memory.

Proc. can look memory into 2
diff. ways :-
1) Real mode
2) Protected mode

Once the s/m is power ON
during exec. of BIOS
loader code exec. processor is in
real mode looking memory as a
series of bytes.

Once the kernel boot up starts
processor shifted to protected mode
& looking memory as a set
blocks { linked list of pages }
& process is called as processor
initialisation.

Requires OS creating illusion to a running some CPU to look memory as set of blocks

GDB :- GNU Debugger tool is a command line tool that works with exec. file that are prod. by the compilation process.

It supports C, Java, C++, Python, Fortran, Pascal.

GDB will look at the symbol table..

Symbol :- Functions & Variable of your source code.

Print :- will print the variable's value

or

info locals

If we observe real-time debug application examples most of the times bugs happens to be from user point of view

e.g :- accidentaley decl. a variable uninitialised.

Su	Mo	Tu	We	Th	Fr	Sa
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

or deref. a ptr. which is already free.

JULY

WEDNESDAY 01

WK 27 183-183

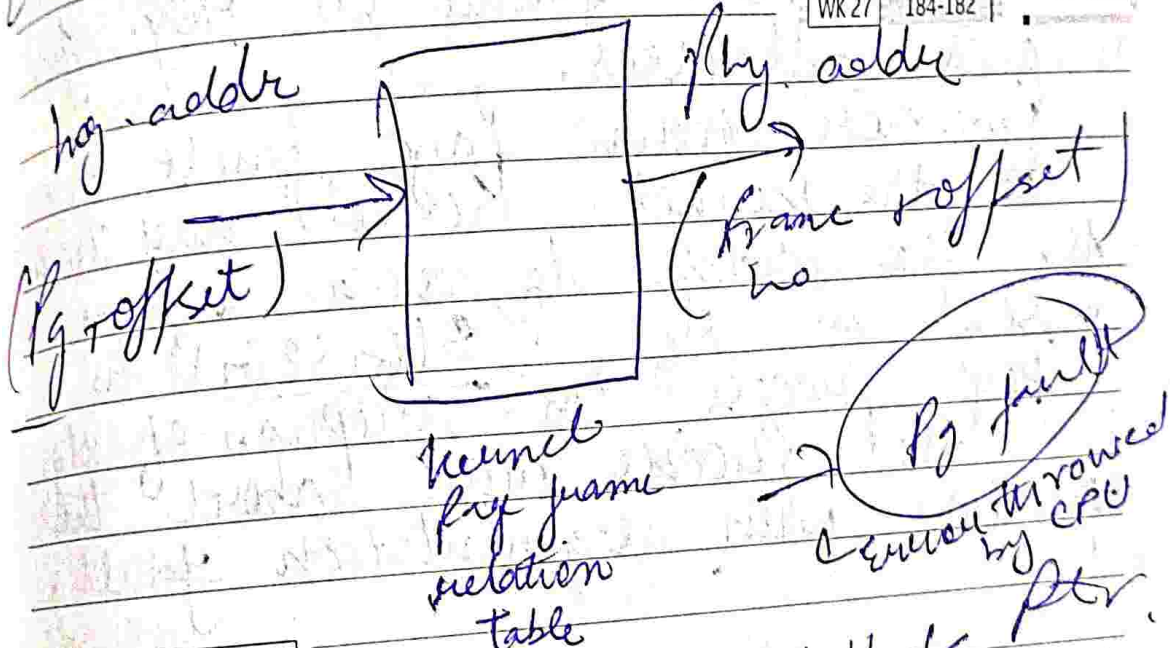
accessing a memory beyond the lower boundary region & beyond upper boundary region.

When a process address space is allocated a p/g is expected to use only the allocated region. whenever p/g makes a violation of memory, segmentation fault occurs.

logical addr — page + offset
physical addr — frame no. + offset

THURSDAY 02

WK 27 184-182



Process abortion

JULY

03 FRIDAY 185-181 WK 27

Pg fault → Kernel ⇒ 20 found Sig

SIASEGV ↑ Aborted

CPU will get the logical address & will map the logical addr. to resp. physical add. with help of Kernel Page frame relation table. In case addr. is invalid memory addresses, CPU giving invalid address to kernel page frame relation table & kernel fails to map & get frame no. reg. for invalid addresses.

04 SATURDAY 186-180 WK 27

Processor throw page fault error to the kernel. Kernel will check for the process sig area & then send a sig (SIGSEGV) to resp. process. On reception of sig the process will abort the operation with segmentation fault.

05 SUNDAY

Su	Mo	Tu	We	Th	Fr	Sa	
			1	2	3	4	JUL
5	6	7	8	9	10	11	
12	13	14	15	16	17	18	
19	20	21	22	23	24	25	20
26	27	28	29	30	31		

infolocals :- will give you all the local variables value.

JULY

MONDAY 06

WK 28 188-178

ptype :- GDB has low used to find any variable type.

Structure padding :- aligning data in memory.

Attribute - Align :- Requesting providing minimum alignment for any structure variable.

~~If its possible to reduce~~

TUESDAY 07

WK 28 189-177

GDB fails to trap down heap memory violations

Electric fence :- is a library not a debugging tool & which has got its own implementation of malloc & calloc memory allocation calls ~~whenever we~~ c using electric fence when req. dynamic memory allocations using malloc & calloc will allocate for specified amount of memory.

AUG '20
Su Mo Tu We Th Fr Sa
30 31 1
2 3 4 5 6 7 8
9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29

JULY

08 WEDNESDAY 190-176 WK 28

Δ calloc allocating more memory of bytes than actually requested for

Electric fence :- is config to report - heap memory violations either upper or lower boundary region but not both at a time.

export CF PROTECT_BELOW = 1
-g flag - config electric fence for lower boundary regions

09 THURSDAY 191-175 WK 28

VALGRIND :- is a runtime tool used to trap down heap memory violations is also called as heap memory profiling tool. "Standalone debugger used at runtime". When process makes use of standard C lib malloc & calloc calls valgrind reports why this process has failed.

-g flag with (w/o debug symbol.

When I am calling FRIDAY 10
buffer print. ASCII values
WK 28 192-174
to the address buffer.

SYS buffer name will print your string until reach new char

SET :- Instead of recompiling the complete code changes can be made from gdb command to modify the value of existing variable with help of gdb command called set.

SATURDAY 11
WK 28 193-173

MM & GDB | Quiz

Makefile :- set of commands similar to.

- Prg building tool in Linux & SUNDAY 12
OS.
Set of commands like to terminal commands.
- Diff. is makefile have organised text instructions

JULY 2020

13 MONDAY
195-171 WK 29

Makefile

Variable Single target Multiple target

Types

Kernel build makefile :- /lib/module/<>/build
Processor architect makefile.
Subsystem

Kernel OS

fs makefile n/w makefile DD makefile

14 TUESDAY
196-170 WK 29

Kernel OS :- OS which you are using.

(1) Kernel build makefile :-

Entire linux kernel OS is build into a makefile called as kernel build makefile located at /lib/module/5.4.0-

kernel version :- uname - r

Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

Version

(2) Architecture makefile.

2020 /lib/module/<kernel version>/build

JULY WEDNESDAY 15 WK 29 197-169

Rules to Write Makefile

- Should start with uppercase letter M w/o any extensions.

- All source files & make files should be in current directory.

- Makefile starts with target name: - Next line should start with a tab space.

THURSDAY 16 WK 29 198-168

Make is a linux utility tool used to gen. appln executables

Make by default comes with linux distribution. Make will reduce workload of recompilations.

make → Makefile (current dirn makefile)

target

gcc — previous — comp — linking — gen. exec. file

Su	Mo	Tu	We	Th	Fr	Sa
30	31					1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29

JULY

17 FRIDAY 199-167 WK 29

Make tool will execute the makefile of current directory & jumps to makefile & starts executing first of makefile.

gcc is indirectly called by makefile in turn develops appln executable

* The primary objective of make tool is to break down large project source code into smaller pieces & assess the smaller piece of code whether they required recompilation or not.

18 SATURDAY 200-166 WK 29

Makefile will contain make variable {CC} & may contain single target or multiple target.

{CC} :- make variable

19 SUNDAY

target :- is the file name i.e generated by makefile. A target can be your appln executable or target can be in object file or maybe an action such as

clean :

NOTE OF Makefile :

JULY

MONDAY 20

WK 30 202-164

applu : main.o mean.o sub.o

o files are "target dependencies" subtargets, In this makefile we are having variable & multiple targets. Here created object files as targets. If I want to recompile only individual file this make will reduce workload of recompiling all files : make is going to recompile only the modified files.

TUESDAY 21

WK 30 203-163

What we've done in fly :

1st) Target clean & Applu.

2nd) Multiple target

3rd) SYSTEM (system call (execl) Wait)

System (fork (parent, child) Execl)

Wait

We a implementation of fork, execl & Wait.

	Su	Mo	Tu	We	Th	Fr	Sa
AUG	30	31					1
	2	3	4	5	6	7	8
	9	10	11	12	13	14	15
20	16	17	18	19	20	21	22
	23	24	25	26	27	28	29

JULY

22nd TUESDAY 2020

203-162 WK 30

#include <stdlib.h>

(const char * command)

shell command

Error child not created

Error too little arg.

10

0 & 1 Value will give the return value of s/m call 0 & 1 will talk about the status of shell.

23 THURSDAY

203-161 WK 30

When you're getting [0] shell is not available. shell is busy executing the command.

When you have NULL given as a command the return value is 1

i.e non zero value indicates shell is available

Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

JUL '20

atomic_t } : Very tiny locks supports for integer data types

FRIDAY 24

Atomics are kernel specific.

tiny lock int variables not used to safeguard resources/pointers.

Whenever declared a var. of type atomic_t kernel provides a set of functions to safeguard the declared integer variables

SATURDAY 25

atomic_t
↓
data type
↓
DS
{ member.
}

lock
X++
unlock

atomic_inc(X);

SUNDAY 26

atomic_t v; } declaring & initializing using macro

atomic_t u } function

atomic variables are u ...

atomic_t

Su	Mo	Tu	We	Th	Fr	Sa
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29

Atomic_int function

JULY

Atomic_t variable ensures safety of int variable on concurrent access.

27 MONDAY

Pthread_exit :- used terminate the current task & allows the pending task to execute

Pthread_cancel :- cancellation req for given thread.

28 TUESDAY

Pthread condition variables :-

Cond :-
Wait
Signal

Declaration :-

Pthread_cond_t my cond;

Pthread_cond_wait

X data

check condition

Pthread_cond_t

* Pthread_mutex_t *

Signal :- should exist the wait

WEDNESDAY 29 JULY
[WK 31] 211-154

pthread_cond_signal (pthread_cond_wait)

pthread has called for mutex lock.

* The mom. thread executes pthread cond^n wait call ***

* Waiting for event to happen

then ** thread is unlocking mutex

*** lock automatically places itself in wait queue of cond^n variable.

THURSDAY 30
[WK 31] 212-154

pthread_cond_sig :- to wakeup only one thread.

pthread_cond_broad (pthd_cond_t.p)

sig to all threads that are waiting

Now: cond^n if any thread waiting on the cond^n will send

	Su	Mo	Tu	We	Th	Fr	Sa
A	30	31					1
U	2	3	4	5	6	7	8
G	9	10	11	12	13	14	15
2020	16	17	18	19	20	21	22
	23	24	25	26	27	28	29

a wakeup thread. sig to

31 FRIDAY 213-153 WK 31 cond_ 2021

pthread_broad argument

-) If a thread executes broadcast function, leads only cond^n arguments & wake up all threads which are sleeping on cond^n variable

Default Policy

kinun threads will have default

01 SATURDAY 214-152 WK 31 sched. policy called as Sched others or Sched normal.

For default sched. policy priority is dynamic and can be changed by a kpm based on

02 SUNDAY

SCHED_FIFO (fixed)

Threads will have a ~~policy~~ priority

from 1 to 99
lowest highest

AUGUST

MONDAY 03

thread with Priority
in Sched FIFO policy runs
execute first when compared to
default FIFO run policy thread.
will not leave until
its terminated. { Real time
priorities }

Pthread_get_Sched_Param() function
going to particular thread
reading policy & 2nd arg
storing into
also reads priority.

TUESDAY 04

WK 32 217-149

	Su	Mo	Tu	We	Th	Fr	Sa
AUG	30	31					1
	2	3	4	5	6	7	8
	9	10	11	12	13	14	15
20	16	17	18	19	20	21	22
	23	24	25	26	27	28	29

AUGUST

05 WEDNESDAY 218-148 WK 32

IPC :- (1) http client process Google chrome

(2) Mail (POP client server) server connecting to process.

Min^m req. for n/w comm :-
IP address
Port address in n/w

Each machine / node is identified by unique 32 bit IP address & is also used for connect & communicate & 16 bit port address is used to identify process in particular node

06 THURSDAY 219-147 WK 32

Linux has provided "5 Different IPC techniques" :-

1) Pipe
2) FIFO
3) Message Queue
4) Shared memory
5) Semaphore

IPC V Std.

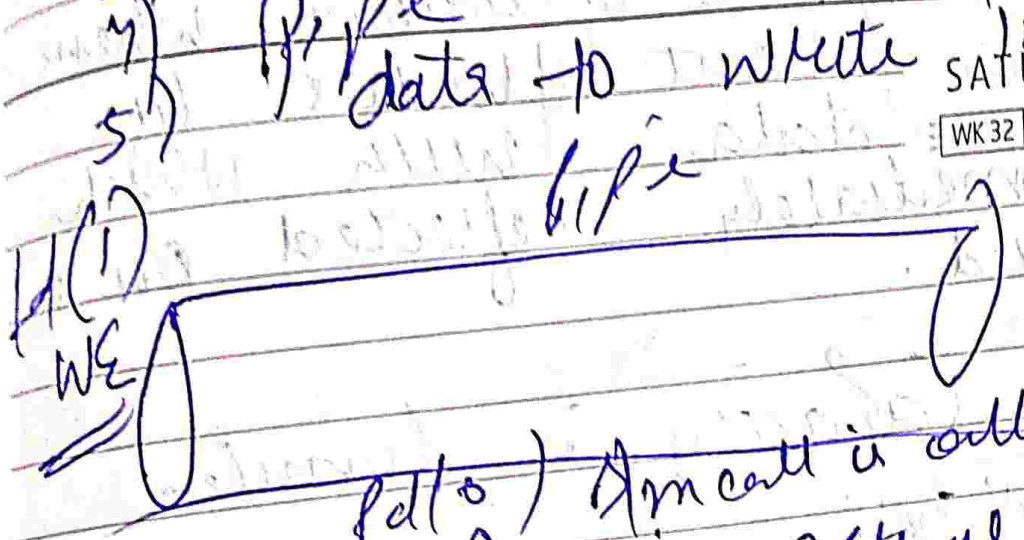
Su	Mo	Tu	We	Th	Fr	Sa	
30	31					1	AUG
2	3	4	5	6	7	8	
9	10	11	12	13	14	15	
16	17	18	19	20	21	22	20
23	24	25	26	27	28	29	

2) Pipe :- Serial comm^n device that permits unidirectional data transfer.

can be used b/w parent & child process (related process).

Creating a file :-

1) int a[2]
2) int read fd
3) int write fd
4) pipe
5) pipe data to write into pipe



fd(0) & fd(1) are called fd process

When a pipe is getting created in kernel space & pipe gets created a pipe also returns 2 descriptors :- fd0 & fd1

fd(0) :- is always assoc with read end of pipe

fd (1) :- is assoc. with write end of pipe

AUGUST

10 MONDAY

223-143 WK 33

When parent process calling fork child is created. When pipe is created parent calling fork. Parent copies fd to the child process. So child inherit a pipe file from parent. Thus a pipe is limited to parent & child.

11 TUESDAY

224-142 WK 33

* After write operation when parent finishes the data with which data immediately reflected on a read end.

PIPE Capacity is limited

Write

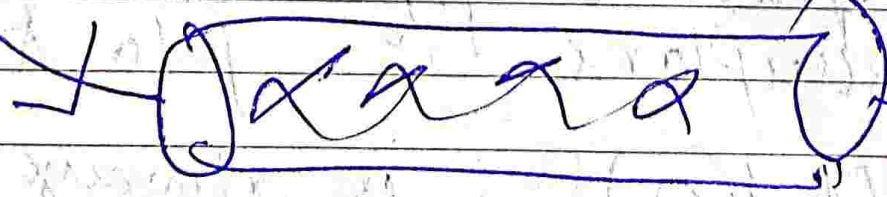
low bps

Blocked

unblock

Read

50 bps



Su	Mo	Tu	We	Th	Fr	Sa	
30	31					1	AUG
2	3	4	5	6	7	8	
9	10	11	12	13	14	15	
16	17	18	19	20	21	22	'20
23	24	25	26	27	28	29	

Blocking & unblocking

WEDNESDAY 12
WK 33 225-141

Writer Process writing at a write end at higher speed than the rd. process which is consuming the data at slower rate. After sometime pipe gets full & if a writer process wants to write data he can't write even though if he has the data to write as pipe is already full as a result writer processor will get blocked until ~~someone~~ space or room created inside the file. {Child reading}

THURSDAY 13
WK 33 226-140

When space created the ~~Pipe~~ writer is getting unblock & starts again writing data at right end of pipe. But there!

~~Reader Process rd. the~~

when a pipe is empty reader process started reading the data from pipe as its (pipe) empty reader process getting blocked.

Su	Mo	Tu	We	Th	Fr	Sa
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30			

until some data remaining to pipe.

AUGUST 2020

14 FRIDAY 227-139 WK 33

The moment data return on a pipe reader is getting unblocked starts reading data from pipe reader.

Pipe automatically synchronizes reader & writer process.

Types of file

1) Regular file
2) Dir file
3) Pipe file (2 fd)

15 SATURDAY 228-138 WK 33

Device File — Special file

4) Char device file
5) Block device file

16 SUNDAY

Keyboard — FD
Mouse — CD
Printer — Hard disk

6) Socket file

Su	Mo	Tu	We	Th	Fr	Sa	
30	31					1	AUG
2	3	4	5	6	7	8	
9	10	11	12	13	14	15	
16	17	18	19	20	21	22	
23	24	25	26	27	28	29	'20

Examine C Source Code

AUGUST

Symbols :- Variables, functions & macros.

MONDAY 17

WK 34 230-136

C Scope :- is a linux utility Tool used for s/w dev. Process to examine the C source code. You can examine the symbols of your source code, e.g. To check to what values they're initialised & where in your pg they're used. Using Cscope tool you can also verify specific function calling what all other functions & also check specific function called by what all other functions, you can change a string, you can find specific files, you can check for ~~#~~ including header files. you can also use argument variable option to change variable value.

TUESDAY 18

WK 34 231-135

Ctags --list -language

Index for name found in same file & header.

name

pathname, in +n

SEP 20	Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4	5
	6	7	8	9	10	11	12
	13	14	15	16	17	18	19
	20	21	22	23	24	25	26
	27	28	29	30			

AUGUST 2020

19 WEDNESDAY 232-134 WK 34

(Ctag) is a linux utility tool generates a tag file i.e a index file for the names found in source code & header files. Index of object refers to name, path name & the line where it is used in the source code. Ctags provides a quick ref. of your source code. e.g:- you need to find definition of func. Ctags uses locatore & locatore will locate the object & pathname of the objects & line in the source code where they're used.

20 THURSDAY 233-133 WK 34

Ctags — location — Index of object — Name, Pathname, line in source code where obj. room is used

Tag file

Su	Mo	Tu	We	Th	Fr	Sa	
30	31					1	AUG
2	3	4	5	6	7	8	
9	10	11	12	13	14	15	
16	17	18	19	20	21	22	'20
23	24	25	26	27	28	29	

Source Control / Version control

AUGUST

20

1) Allows trackdown your file progress of your project.

FRIDAY 21

WK 34 | 234-132

2) Stores all modf^n / changed from one a time line.

3) Protecting against data lost.

4) Complex project dev.

e.g linux is a massive project on github. supported by lakhs of contributors

SATURDAY 22

WK 34 | 235-131

Git :- Source controlled S/W : it has lots of support & documents & industrial standard.

Saves all file progress as series of snapshot.

SUNDAY 23

GitHub :- Application s/w & always save to store git super space & versions by giving

SEP '20	Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4	5
	6	7	8	9	10	11	12
	13	14	15	16	17	18	19
	20	21	22	23	24	25	26
	27	28	29	30			

AUGUST

Github Adv

24 MONDAY 237-129 WK 35

→ Disconnected repo. sys.

Github supp.

GIT PROJECT

Working | Staging | Repositing

3 diff - files

Modified or changed files

25 TUESDAY 238-128 WK 35

Staged file :- Marked as ready to move next level

Safely stored in Stage

Git Config. Variables is a config var. used to set/get git config.

git config --global username "email"

gets inf. & stores in local machine

Su	Mo	Tu	We	Th	Fr	Sa	
30	31					1	AUG
2	3	4	5	6	7	8	
9	10	11	12	13	14	15	
16	17	18	19	20	21	22	20
23	24	25	26	27	28	29	

Clone :- to copy remote repository into your local machine

2020 AUGUST

WEDNESDAY 26

WK 35 239-127

FIFO :- Syntax

<systypes.h> <sys stat.h>

mkfifo (const char *pathname, mode_t modes).

0 -1

makes a entry into linux kernelos file s/m.

FIFO has a special file which has got all the attributes. Any process can open it for reading & writing like any other normal regular file.

THURSDAY 27

WK 35 240-126

Ensure that FIFO is opened from both the ends simultaneously by 2 diff. process (i.e Reader & writer) If your process is opening a FIFO for reading option has to wait for other side process to open FIFO for writing option.

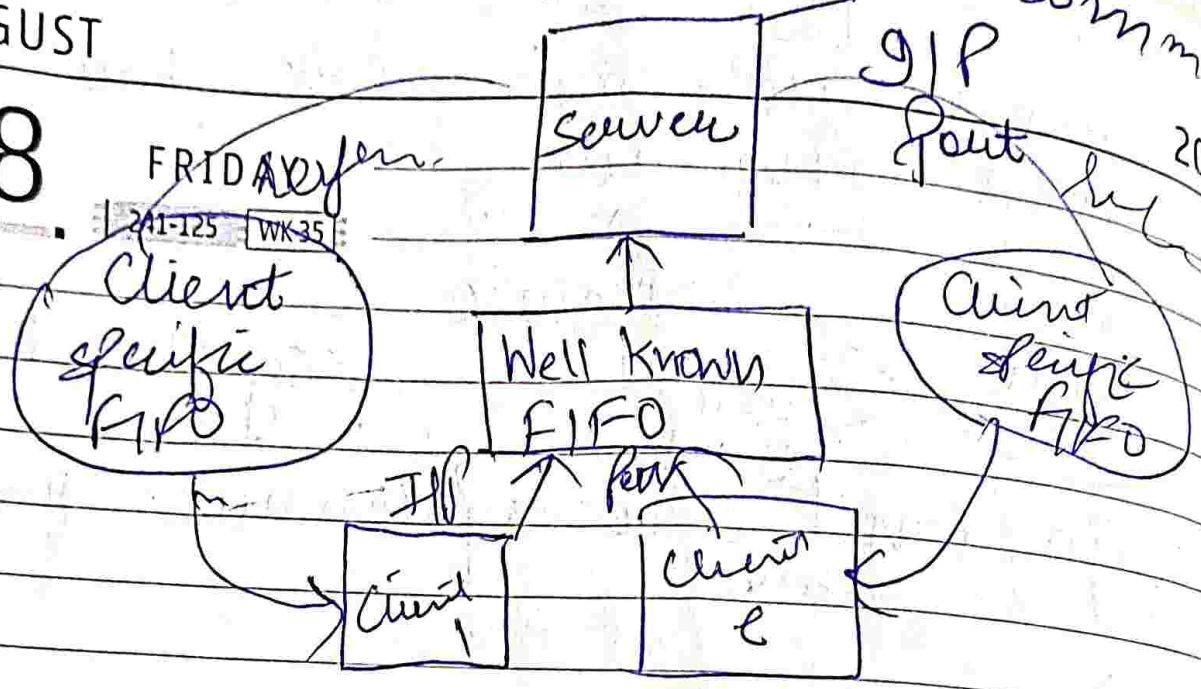
A FIFO can have multiple readers & writers & supports

SEP '20
Su Mo Tu We Th Fr Sa
1 2 3 4 5
6 7 8 9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30

Connection oriented comm.

AUGUST

28 FRIDAY 241-125 WK 35 2020



FIFO is used for simple client server

29 SATURDAY 242-124 WK 35

msg ctl :- (int msg_id, int cmd, struct msg_id_ds *buf)

Supply

IPC_STAT

struct msg_id_ds {

fstat

IPC_STAT

IPC_RMID

30 SUNDAY

IPC_SET

Su	Mo	Tu	We	Th	Fr	Sa	
30	31					1	AUG
2	3	4	5	6	7	8	
9	10	11	12	13	14	15	
16	17	18	19	20	21	22	20
23	24	25	26	27	28	29	

Shared memory :- one of the IPC comm^n technique.

Allows 2 or more process to access a given region of shared memory & its fastest IPC comm^n tech.

∴ no need to jump from one memory location to other memory location for data read & write operation b/w reader & writer process or a client & server

Can be used b/w client & server machines. Only trick is synchronizing access of shared memory with semaphores techniques.

Each of every shared memory and have a structure called as

(struct shmid_ds)

Shmat :- When a process executes shmat s/m call mm take the given shared memory region & map into a free process address space.

SEP 2020
Su Mo Tu We Th Fr Sa
1 2 3 4 5
6 7 8 9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30

In 2nd aug. address proved.

SEPTEMBER is O.

02. WEDNESDAY 246-120 WK 36 2020

Linux Kernel Architecture

System Calls are kernel space services not user space services. They are meant to invoke kernel services. Each & every s/m call has to pass through s/m call interface.

& then, invoke approp. Kernel service

SEPTEMBER

FRIDAY 04

WK 36 248-118

Drivers from Kernel point of view

A device driver is a piece of s/w that comm^n with h/w manages h/w & brings the use of n device to the use. In Linux OS, DD resides in the kernel space ∴ only the kernel space driver has privilege to talk to the h/w.

SATURDAY 05

WK 36 249-117

Driver has 2 interfaces :-

bet^n application & driver i.e O.S. specific. This is completely based on h/w implementation.

SUNDAY 06

Semaphores :- one of the IPC comm^n tech. It deals with array of semaphores but its a bit complex issue, but in large apply s/w

OCT 20

Su	Mo	Tu	We	Th	Fr	Sa
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

SEPTEMBER

Process needs to work on lots of resources

07 MONDAY

251-115 WK 37

need more protection than having array of semaphore is big advantage

semget ( ), semop ( ), semctl ( ) are designed to work on array of semaphores.

semget (key_t key, int no of sem, int semflag)

semid

08 TUESDAY

252-114 WK 37

On succ. exec. of semget func returns sem_id

semop (int semid, struct sembuf *pts, size_t, No of structures)

Function Pointer

+1

sem - no

sem - OP

semctl

sem - UNDO

Su	Mo	Tu	We	Th	Fr	Sa
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30			

2020

semctl (int semid, int sem_no, int cmd, union semunion);

SEPTEMBER

WEDNESDAY 09

WK37 253-113

semop :- s/m call used to change the value of semaphore.

semctl :- used to control the given semaphore.

IPC_RMID :-

Int command as set val :- used to initialize the semaphore to a well known value. The value is req. to pass value member of union semunion. Only then the semaphore will settle. This proc. is performed when we're using semap. waiting semaphore for first time.

THURSDAY 10

WK37 254-112

Su	Mo	Tu	We	Th	Fr	Sa
				1	2	3
4	5	6	7	8	9	10
11	12					
18	19	20	21	22	23	24
25	26	27	28	29	30	31

No semaphore waiting

Sem_undo

11 FRIDAY 255-111 WK 37

Kernel will read sem unread & will undo what changes have been done to the semaphore by the process.

If process terminated w/o releasing semaphore sem_undo will allow to automatically release semaphore.

Signal | LINUX

12 SATURDAY 256-110 WK 37

S/g are s/w interrupts. They notify process about some event occured. S/g are async in nature. They can happen at any time. A H/W kernel or any s/w that runs on CPU can trigger s/g to the process.

13 SUNDAY

A process with enough permission can send a s/g to another process. A process can send s/g to itself.

Types of S/g

1) Terminal gen. (Ctrl S)

SEPTEMBER

S/g gen. by terminal keys

MONDAY 14

WK 38 258-108

Linux s/g have a naming convention they start with 3 char. SIG. Each & every s/g has defined by a no. prov. in header files called s/g.h.

H/W exception s/g.

1) Invalid memory ~~access~~ reference.
2) Divided by 0.

TUESDAY 15

WK 38 259-107

Syntax used to register kernel

void (* Signal (int signal, void (* fun) (int)) (int);

	Su	Mo	Tu	We	Th	Fr	Sa
OCT					1	2	3
	4	5	6	7	8	9	10
	11	12	13	14	15	16	17
	18	19	20	21	22	23	24
20	25	26	27	28	29	30	31

SEPTEMBER P0.7 P0.1 P0.0

16. WEDNESDAY 260-106 WK 38

9/0 DIR0 = _____ 0X

31 - - - . - - 7 6 5 4 3 2 1 0

0.31

1 0 0 0 0 | 0 0 0 0

8 4 2 1 8 4 2 1

1 0 0 0 0 0 0 1

81

17. THURSDAY 261-105 WK 38

To set value 1 we have 2 registers :-

GPIO — SIP
— OIP

IOSET 0 } 1
IOSET 1

IOCLR 0 } 0
IOCLR 1

PINSEL 0 = 0X00

IODIR0 = 0X81

IOSET0 = 1 → 1

IOCLR0 = 1 → 0

Led ON

Led OFF

UC

Every con & L

SEPTEMBER

FRIDAY 18

WK 38 262-104

```
# include <lpc213x.h>
void delay (int ms)
while(ms)
{
while (ms) {
    --ms; } } }
int main (void) {
PINSEL0 = 0X00;
IODIR0 = 0X09;
while (1) {
IOSET = 0X09;
delay (500);
IOCLR0 = 0X09;
delay (500); } }
```

SATURDAY 19

WK 38 263-103

AKM = 3.3V fully
sunny

SUNDAY 20

Vref, V3

supply line

	Su	Mo	Tu	We	Th	Fr	Sa
					1	2	3
37	4	5	6	7	8	9	10
38	11	12	13	14	15	16	17
39	18	19	20	21	22	23	24
40	25	26	27	28	29	30	31

Task :- Binary counter using 8 LED's

SEPTEMBER 2020

21. MONDAY 265-101 WK 39

```
* for (i=0; i<=255; i++)
    IOSETO = i
    delay (500)
    IOCLR = i;
```

common cathode — 0 – 9 display 7 seg

```
for (i=0 i<=9, i++)
  {
    IOSETO = a[i];
    delay [ ];
    IOCLR a[i];
  }
}
```

22. TUESDAY 266-100 WK 39

1 – ON

~ complement sign

Anode me off

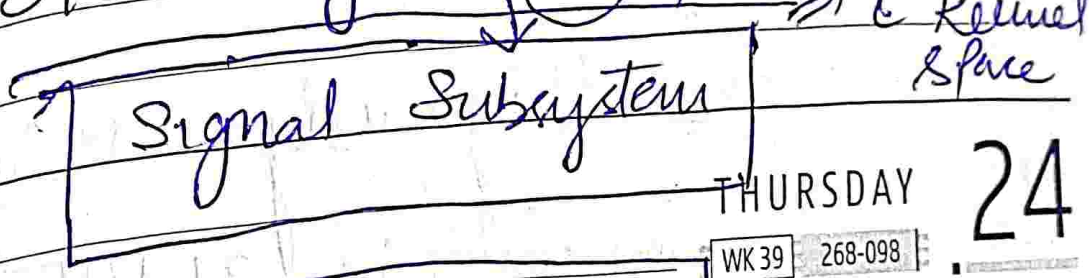
Linux

WEDNESDAY 23

S/g handler takes integer & returns void.

Diff Types of Signals

1) Job control s/g.
2) Process comm^n s/g.
3) I/O s/g



THURSDAY 24

TIMER SUBSYSTEM

In the kernel there's a s/g subs/m which is resp. for delivering of a s/g to each process in user space of appln (1) Wants to send a s/g to appln (2) it has to register with the s/g subsystem & convey the appln. id & s/g no. to which the process 1 or appln (1) wants to send a s/g.

Su	Mo	Tu	We	Th	Fr	Sa
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

SEP/OCT

30 WEDNESDAY 274-092 WK 40

Sig subsystem on behalf of appln (1) deliver the sig to appln (2)

CASE 2: Appln 'x' wants to perform periodic task func. then register with the timer subsystem for delivery of a sig at a particular time out.

TIMER subsystem checks the timedlist.

Timer subsystem invoke the sig subs/m after some TIME OUT & sig subs/m will deliver a sig (SIGALRM) to appln space.

01 THURSDAY 275-091 WK 40

Kernel

init Pid=1 | Pid=0

or

~~System Admin~~ | Type of sig.

Job control sig *

Under Job control sig

1) SIGINT 2
2) SIGKILL 9
3) SIGSEGR 11
4) SIGTERM 15
5) SIGSTOP 19

we got

SEP 2020

Preemptive process is one

OCTOBER

kernel forcibly stopping a CPU from process w/o req. for it so everything is under control.

FRIDAY 02

WK 40 276-090

Non Preemptive Process

Here until the timeslice has done the process will not leave the CPU or the process should go to block state.

"SIGKILL & SIGSTOP" sig can't be handled or caught

SATURDAY 03

WK 40 277-089

SIGUSR 1
SIGUSR 2 } process to process

SIGALRM
SIGCHILD

SIGIO → Processes

SUNDAY 04

I/O sig → all drivers.

↳ are delivered by I/O subsys to the processes when a particular file descriptor or sockets are ready to perform I/O operations.

Su	Mo	Tu	We	Th	Fr	Sa
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

Struct sigval *

Execution of Default Handler

05 MONDAY

In sig subsystem there a func called as SEND_SIG & this function is executed before delivery of sig

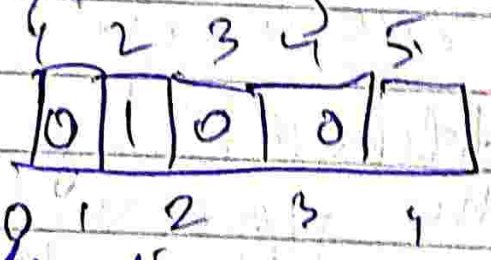
{STEPS FOLLOWED BY SEND SIGN

Send_Sig func will get ref. of PCB to which sig is to be delivered, it will point to the sig structure ref. by PCB, then

06 TUESDAY

a sig pointer to sig structure i.e pointing to a vector element of 64 integer SENDSIG will manipulate the sig structure.

e.g :- For sig int SIG_INT () manipulating 2nd element of vector



After sending send sig raise CPU exception

CPU stops the current task

Su Mo Tu We Th Fr Sa
1 2 3
4 5 6 7 8 9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31

& CPU executes an exception handler function.

Now except. handler function will now access the structure in PCB i.e s/g handler structure.

A ptr of s/g handler func pointing to vector of integers.

Exception handler will look for the s/g structure & for enabled element & maps resp. func. in s/g handler vector & now s/g except. handler is executing default func. in interrupt content & until this handler is executed process will not resume

When def. func. is exec. in the interrupt content, the context switching is not possible.

Thus Process is dealing with critical section & critical section is updating a particular database. During this updation process do not want to preempt from process can

NOV 20
Su Mo Tu We Th Fr Sa
1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30

Block Particular sig. blocked a sig n Appli a 2020

OCTOBER

09 FRIDAY 283-083 WK 41

next appli y want the same sig. Kernel will give the sig n move to the PCB of appli n then places sig in pending sig structure.

This sig n will be released only when appli n unblocks the sig.

Sigempty set :- We have this func (Sigset t * set) & there are no initialization in it by default.

10 SATURDAY 284-082 WK 41

Sigfillset (Sigset t * set)

Initializing with all elements & all sig are set now.

11 SUNDAY

Sigaddset (sigset t * set, signo)

Sigprocmask (flags, sigset t * set, sigset t * oset)

sim call will add a block state or will remove

s/g from block state based on sigblock & sigwakeup flags.

2020 OCTOBER

MONDAY 12

WK 42 286-080

(sigismember) checking whether this is a member of blocked or not.

Appln blocking a s/g called SIGINT for some time out. During the time out the process should be terminated using a kill command.

Error.

TUESDAY 13

WK 42 287-079

A mistake in p/g error. If the error detected by a tester is called as a defect. If the defect is accepted by respected dev. then it is bug. The process of fixing bug is called debug.

When a s/m s/w fails to perform particular func. execution leads to failure.

Fault :- is a cond^n of which a s/m s/w fails to perform an action

NOV '20

Su	Mo	Tu	We	Th	Fr	Sa
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

STATIC CODE ANALYSIS

14 WEDNESDAY 288-078 WK 42

is the process of identifying the prg errors & bugs in the source code before the prg is being run. It is done by performing on set of inet by using some coding standards. This kind of analysis will help to identify the loophole & weaknesses in the source code that might be harmful. Analysis is done on a stationary piece of sw.

15 THURSDAY 289-077 WK 42

Splint :- static code analysis tool is used to identify prog. errors & suspicious code & also stylistic error.

Programming errors are shown & identified by compiler at compilation time e.g. :- Syntax errors, usage of undeclared variables, Type checking, library incompatibility. If there's a prog. lang. that doesn't have any compilers like Splint are used.

STYLISTIC ERROR :-

OCTOBER

If code is deviating from prescribed coding standard.

FRIDAY 16

WK 42 290-076

Resp. of LINKER

Linker job is to provide a.out file. Linker file forms (~~both~~ source code + lib) runtime.

Runtime is not a library. It is a set of routines added by the linker during pg build time.

Runtime code talks about 3 diff macros:-

SATURDAY 17

WK 42 291-075

~~start~~ _init, _start, _fini.

The mom, your appl^n starts exec. it starts with _init macro & is also called as init macro. Initialiser will allocate key resources for your object file & also config. the object file. Prov the addresses to your object file to load & execute.

SUNDAY 18

Once addresses are config. then control goes to _start macro. New start macro is

	Su	Mo	Tu	We	Th	Fr	Sa
	1	2	3	4	5	6	7
	8	9	10	11	12	13	14
	15	16	17	18	19	20	21
	22	23	24	25	26	27	28
	29	30					

Procedure used to execute the main func. & control the jumps to main func. execution. Here the appl^n func. executes when main func. terminates functionality is also terminated & control goes to start macro & start macro calls fini macro. fini deallocates as the task is done from init macro.

- init → initializes (constructor)

- start → Applni
- fini → clean up Destructor

HLVM :- how level virtual machine.

Clang :- It is a compiler used for C/C++ lang & it is build using C++ & release under Apache 2.0 license. Clang is faster & takes less memory when compare to GCC

e.g.:- Google chrome browser is build

using Clang.

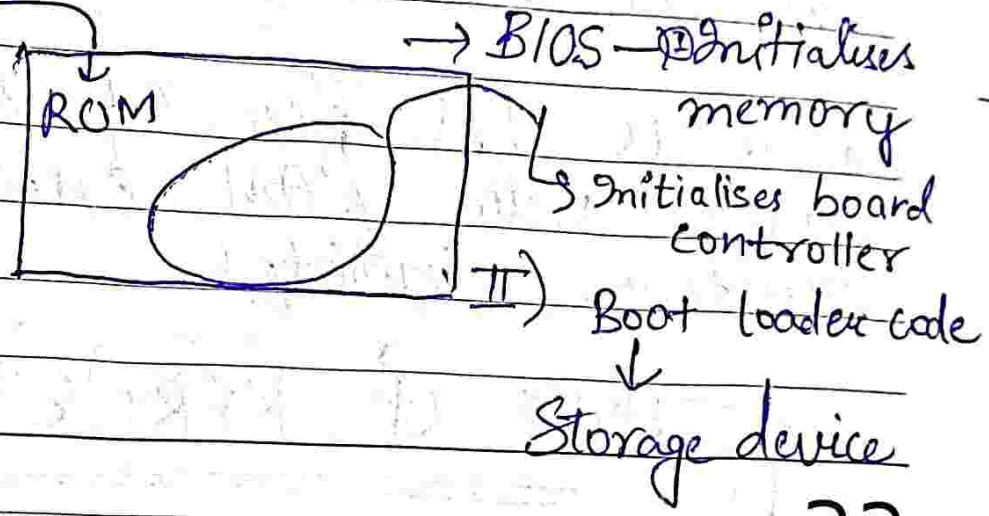
2020 OCTOBER

Clang supports few dev. environments

WEDNESDAY 21
WK 43 295-071

Linux Boot Sequence

POWER ON



THURSDAY 22
WK 43 296-070

GRUB :- Grand unified boot loader. It is used to read root file systems.

While executing BIOS code & boot loader code looking at memory as array of bytes or seq. of bytes. Vendor of the mother-board is providing address of BIOS code.

The kernel IMAGE divided into 2 parts.

NOV '20

Su	Mo	Tu	We	Th	Fr	Sa
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

OCTOBER 23 FRIDAY 297-069 WK 43 2020

(1) Boot strap code
(2) Vmlinuz

The 1st piece of code, bootstrap code executes & shift the CPU from real mode to protected mode & process is called as processor initialisation.

Kernel is 1st piece of s/w that is loaded into RAM executes until shutdown initiated.

ROLE OF KERNEL

24 SATURDAY 298-068 WK 43

will set up memory, DS, interrupt DS device DS, process table, file management, initialise CPU scheduling, initialises kernel threads.

There are few device that have boot options :-

25 SUNDAY

GCOV :- gcc coverage tool. open source tool.

Works only with gcc.

Su Mo Tu We Th Fr Sa
1 2 3
4 5 6 7 8 9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31
O C T 20

Used for analysis of source code & check of untested part of code & identifies unexecuted instn can also be used for browsing & navigating source code allows you to modify or enhance the source code.

Compiling your source code with 2 flags :-

1) -fprofile -arcs
2) -ftest-coverage flag will do analysis on the source code & gen. a file & gen. a hidden code into binary & also records how many time the code is executed.

When we are using -fprofile -arcs with source code & gen. profile O/P file with gcno & gcda.

Su	Mo	Tu	We	Th	Fr	Sa
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

Network Programming

OCTOBER 28 WEDNESDAY 302-064 WK 44 2020

is all about writing appln pkg used in n/w API's & n/w protocol stack. There pkg comm with pkg either on same machine or pkgs on diff. machines at remote location.

Computer N/w

- WAN
  - CSN
  - PSN
- LAN
  - Works on broadcast approach
  - No intermediate switches ∴ of the data rates of LAN > WAN.
  - e.g. Telephone.

29 THURSDAY 303-063 WK 44

- A ded. phy path is established b/w 2 points through several nodes.
- Path is a connection of seq. of phy links b/w the nodes. {Cabled line}
- If a sender wants to tx. packet the packet goes through these phy. lines as rapidly as it can.

(PSN) :- use n/w protocol that divides data into smaller packets called as n/w packets

2020 OCT/NOV FRIDAY 30 WK 44 304-062

n/w packets are tx. over n/w {IN DIGITAL (0,1) format}

e.g. WLP-1

DIFF. N/W DEVICES

Router (switches are n/w devices. They are used for conn. 1 or more devices for other comp or n/w devices or to connect other n/w.

SATURDAY 31 WK 44 305-061

HUB used for connecting devices in a LAN.

BRIDGE :- is a n/w device that connects 2 LANs.

SUNDAY 01

g/f add :- add of a node.

NOV 20

Su	Mo	Tu	We	Th	Fr	Sa
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

HOST A, HOST B — LAN — HUB — bridge — HUB — host 2

There are diff. types of computers which are using diff. CPU & M/C. Few CPU follows big INDIAN format & few little INDIAN format to store integer byte. All the protocol address of a n/w packet should follow n/w byte order. Every machine in n/w should be aware of this & convert host order to n/w byte order. Again at sender side while sending incoming packets should convert n/w byte order to host byte order.

Source code uses API to convert host - n/w order & vice-versa to match, resp. CPU requirements. This API's will work for little INDIAN & BIG INDIAN.

htonl(long host long)

This takes 32 bit host order IP address convert into 32 bit n/w

Su	Mo	Tu	We	Th	Fr	Sa
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

byte Order

2020

htons (u short host short)

NOVEMB

WEDNESDAY 04

WK 45 310-051

Host Port : 16 bit
N/w Byte 16 bit

TCP VS UDP

4 Way Handshaking

Server — Client

FIN →

← WAIT ACK

ACK →

WAIT FIN

FIN →

THURSDAY + FIN 05

WK 45 310-056

← ACK

ACCEPT S/m CALL

is called by TCP server. shall extract the 1st client req. on the queue & creates a new socket of the same socket i.e specified. & Address family of newly created socket is of same type & return

DEC '20

Su	Mo	Tu	We	Th	Fr	Sa
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

a socket fd.

06 FRIDAY | 311-055 | WK 45

Scenario :- 9 implement fork in ser to handle a client request.

DATA TRANSFER API's

S_size_t recvfrom(int sockfd,
char * buf,
int length,
int flag,
struct sockaddr * addr,
int addrlen);

07 SATURDAY | 312-054 | WK 45

Integer Pointer

Taking 6 arg :- empty buffer
fd
socket addr struc

08 SUNDAY

RAW SOCKETS

Raw sockets - sends raw packets to specific appln user. Now user will get untampered final packet. This packet is bypassing all TCP/IP processing

If you want to improve the performance & security on critical traffic workloads on WAN, RAW sockets can be used.

DLH STRUCTURE

```
struct sockaddr_ll {
    sll_family;
    sll_protocol;
    sll_pkt type;
    sll_ha type;
    sll_addr[8] → physical layer
    sll_address }
```


Packet interface at Device level

PF_Packet family used to ~~get~~ create packet socket. Now data tx. is from physical link layer i.e below Internet protocol layer

Socket(PF_Packet, Type (ETH_P_ALL))

Su	Mo	Tu	We	Th	Fr	Sa
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

The packet including when I am using SOCK_RAW the packet including link layer header is passed to the

specific user. When using type as SOCK_DGRAM 2020

11 WEDNESDAY 316-050 WK 46

Packet w/o link layer header is passed to specific user.

(ETH_P_ALL)
↓ value

This protocol defined by IEEE people i.e. Standard Ethernet Protocol

---

int SetSockopt (int sockfd, int level, int option_name, char * option_value, size_t size of option value)

0 / -1 (fail)

12 THURSDAY 317-049 WK 46

Used to change the socket behav
We can set any option of socket.

- int level → Req. from socket level
  Req. from transport
  → Req. from n/w layer.

FRIDAY 13
WK 46 318-048

Writing your own Terminal Emulator
Pg while reading from multiple I/O sources.

e.g:- Serial port & keyboard.
If a process config to read from keyboard when no data from the keyboard the process will be blocked. Even if there's a data at serial port process can't read.
To overcome the scenario we may make use of multiple threads.

SATURDAY 14
WK 46 319-047

File descriptor set :- we will use this using "select" s/m calls.

Select :- s/m call makes programmer to read from multiple file descriptors when fd are ready for I/O operations.

SUNDAY 15

Select (int nfds,
fd_set *readfds,
fd_set *writefds,
fd_set *exceptfds,
struct timeval (timeout));

The 1st parameter "nfds" tells the select s/m call the no. of fd's to be checked.

The 1st ptr. of type fd set is for testing the readyness of file descriptors for read operations.

The 2nd ptr of type fd set is for testing readyness of fd for write operations.

If any of fd is having error pending will be verified by this pointer.

TIMEOUT :- is a var. of type struct timeval structure. It will make select

Su Mo Tu We Th Fr Sa
1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30

s/m call to lock until a fd is ready for r/o operations

2020 NOVEMBER

WEDNESDAY 18

WK 47 323-043

Select → no of fd's set by either read fdset or write fdset (error fdset.

0 → no of fd's found

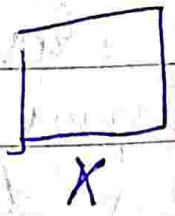
-1 → error code

FD_ISSET

THURSDAY 19

WK 47 324-042

Select s/m call can wait on max. no. of files less than fd set size. i.e 1024



X

Y

SSH username

DEC '20

Su	Mo	Tu	We	Th	Fr	Sa
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

20 FRIDAY 325-041 WK 47

Open /SSH Client/ can connect to remote machine & control them. Open SSH is a open source s/w i.e used to provide a secure & encrypted comm^n in a computer n/w using SSH protocol.

SSH Protocol is establishing a connection b/w SSH client & server

- used by s/m administrator to access a remote machine in unsecured n/w.
- SSH is a TCP/IP protocol that uses port no. 22

21 SATURDAY 326-040 WK 47

SCP :- Secured copy

After setting up comm^n b/w client & server SSH protocol is providing strong encryption & algorithms for ensuring the privacy & data security b/w client & server machines

22 SUNDAY

Read man is a file created under process s/m

2020 NOVEMBER

MONDAY 23

WIRESHARK :- is a open source s/w tool that captures incoming & outgoing n/w data packets in real time. The captured inf. is used for n/w troubleshooting & diagnosing & packet analysis, n/w development & comm protocol dev. With wireshark we can see the things around the n/w & we can get the detailed packet moment inf. It supports 100's of packets of diff. protocols. The only logic update is capturing online & examine them offline. It can read & write diff. captured file formats.

TUESDAY 24

Main tool bar gives 4 functionality filter bar is used to filter the N/W. package.

IP METADATA :- IP packets carry IP headers [20-24] bytes & the actual data.

DEC 20

Su	Mo	Tu	We	Th	Fr	Sa
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		