8th may TUESDAY

FEBRUARY 2022

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

Linux Internals

Wk. 03 • Day (011-354)

January 11

9.00

InterProcess Communication

10.00

→ InterProcess communication allows a process to communicate with another Process.

11.00

12.00 → communication can be one of two types:-

i) Between related Process (Parent & child)

ii) Between unrelated process (one or more different Process)

1.00

2.00 → IPC can use:-

i) Pipes } Sys-III file structure oriented

3.00 ii) FIFO }

iii) Message Queues }

4.00 iv) shared memory } Sys-IV object oriented

v) semaphores }

5.00 vi) signals

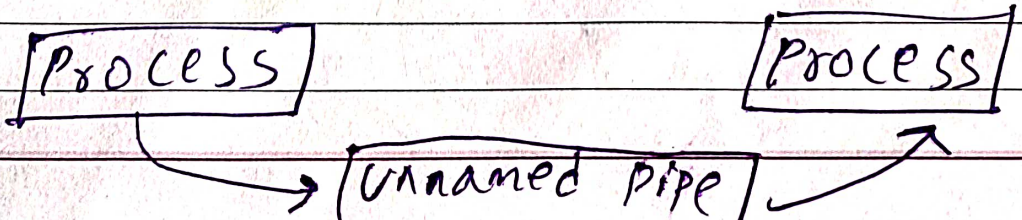
6.00

Pipes

→ unnamed pipes

→ pipes are half-duplex (meaning they can only write or read from process, but only one direction)

→ Ex :- ls -lR / | more



2022

WEDNESDAY

12 January

Wk. 03 • Day (012-353)

JANUARY 2022

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

9.00 → First In, First out (Named Pipes)

10.00 → FIFO are also called named pipes and are half duplex

11.00

→ Named Pipes are also called FIFO

12.00

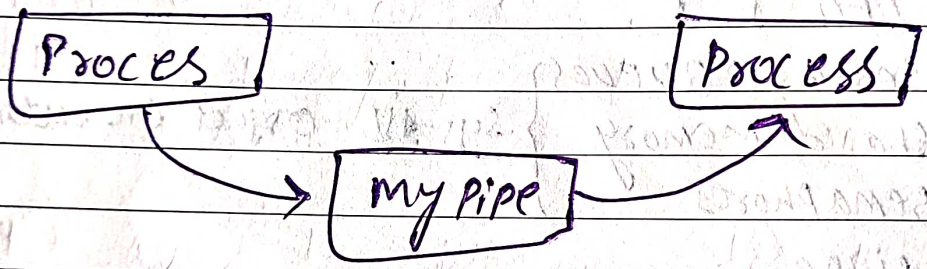
→ Ex

1.00 → mkfifo mypipe

→ echo "Hellow Pipe world" > mypipe } 1st terminal

2.00 → cat mypipe } 2nd terminal

3.00



4.00

5.00

6.00 → until unless the data is out from the pipe the terminal pauses there.

→ once that data is out the terminal comes back to its normal position.

2022

9.00 → # Shared Memory

10.00 → Shared memory is full duplex, either process can read and/or write.

11.00 → Most efficient type of IPC, it does not require any kernel intervention once the shared memory has been allocated/deallocated

12.00 → Requires a program

1.00 → Any number of process can read and/or write to the same shared memory segment

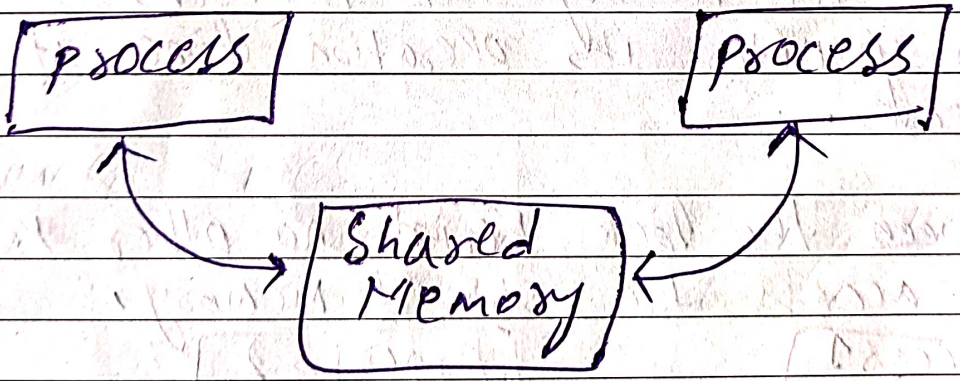
2.00 → you can query shared memory with IPCS command

3.00 → Process must manage shared memory

→ Process must protect (synchronize) shared memory being written or race conditions will occur.

5.00

6.00



→ IPCS

FRIDAY

14 January

Wk. 03 • Day (014-351)

JANUARY 2022

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

9.00

What is Race Condition?

10.00

→ Let's assume Process 1 and Process 2 are sharing a memory location.

11.00

→ Process 1 wants to add 2 to the value that is currently in the shared memory location.

12.00

→ Process 2 wants to add 3

1.00

→ Both process read the initial value of 0.

→ Process 1 adds 2 and changes the shared memory location to 2

2.00

→ Process 2 adds 3 and changed the shared memory location to 3.

3.00

→ This is wrong, final the value should have been 5, this is race condition.

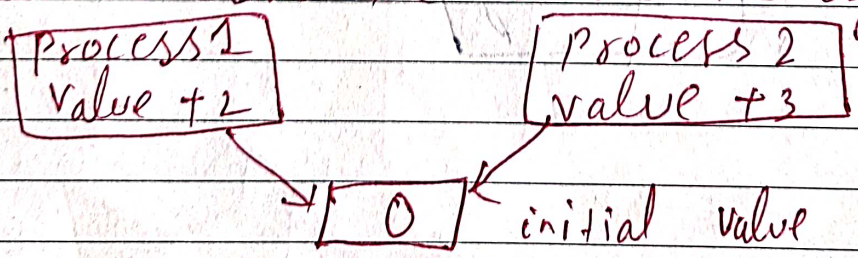
4.00

5.00

→ To avoid the who eves the process access the memory then it locks the memory after its operation done it releases.

6.00

→ Between the lock period no other process can access the shared memory,

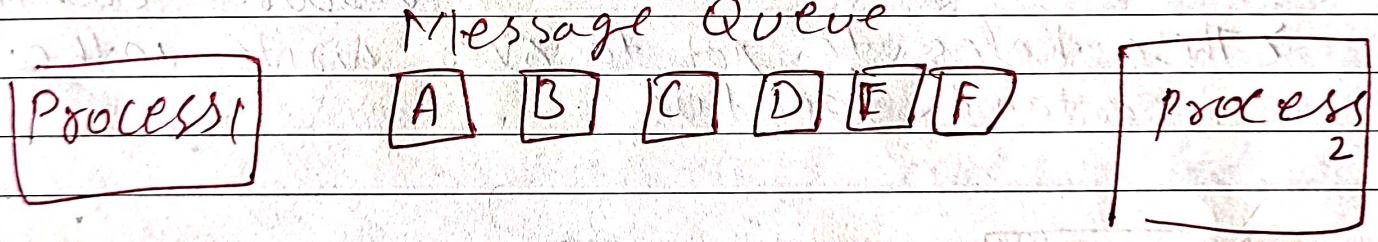


2022

9.00

Message Queues

10.00 → Message Queues are created by a syscall
→ Message Queues are managed by the kernel
11.00 → Once a message is read it is deleted from the queue by the kernel
12.00 → Each Read and write creates a syscall to the Kernel.
1.00 → The message queue helps eliminate the occurences of race conditions but comes at the expense of performance due to the syscall interrupt
3.00



6.00 → Process 1 write the data & process 2 reads the data.

→ Untill Message Queue is fulled the data will not be read by process 2;
→ Multiple process can read / write data in message queue.
→ Once the data is read it's gone.
→ Process 2 & 3 reads the data after process 2 completes it's reading the data of process 2 is deleted.

Sunday 16

202

MONDAY → spin logs

17 January

Wk. 04 • Day (017-348)

JANUARY 2022

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

9.00

Semaphores

10.00

→ Semaphores are used to protect critical common regions of memory shared between multiple process.

11.00

12.00 → Semaphores are the atomic structures of operating systems.

1.00 → Two type of semaphor.

i) Binary :- only two states 0 and 1 locked/unlocked or available or unavailable.

2.00

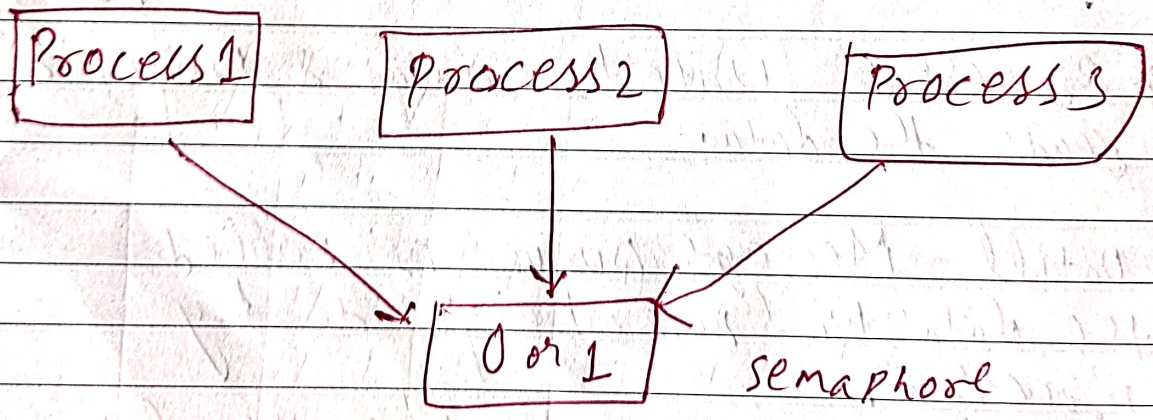
3.00

ii) Counting semaphores - allow arbitrary resource counters.

4.00 → These helps to not to over write in the important location

5.00

6.00



- Named semaphores
- Unnamed semaphores

2022

9.00

Signals

kill -l

10.00

→ A signal is a notification of an event occurance

11.00 → A signal is also know as trap, or software interrupt.

12.00 → If you perform a kill -l at the command line you will receive a full list of signals used by linux.

1.00 → signals can be generated by a user, a process or the kernel.

2.00 → A process is supposed to be written to handle them certain signals.

3.00 Numesic 9-15 can not be handled by the process they will immediately cause termination the the process and can not be blocked, these are called "process crash-outs".

4.00

5.00

6.00

→ Eg:- CTRL + C at the terminal will send a ~~signal~~ SIGINT signal to the process which is currently running in the foreground.

Process Management

→ Any running program is a process
→ Multiple instances of the same program are just process.
→ The shell is a Process
  * When you enter a shell command
  * Control passes from the shell to the Process
  * Control passes back to the shell when the process exits

PID (Process ID)

→ Each linux Process is identified by a unique Process numbe called the PID

→ Every Process has a Parent Process ID (PPID)
  * Except init
  * The PPID is the PID of the Parent Process

→ When a Process is loaded into memory there is structure:
  * stack
  * Heap
  * Data segment
  * Code segment

FRIDAY

28 January

Wk. 05 • Day (028-337)

JANUARY 2022

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

9.00

Creating a Process

10.00

→ Two ways this can be done
- using system() - a simple method, but is very inefficient, has security risks
- Using the fork() and exec()

11.00

12.00

1.00

System()

2.00

→ Creates a sub-process running the standard shell
→ Passes the command to this shell for execution.

3.00

→ Very similar to typing the command name to a shell

4.00

fork()

5.00

→ Creates a child process by making an exact copy of its parent Process

6.00

→ The parent process does not wait, it will immediately continue on

→ The child Process starts executing in exactly the same place as the parameter

2022

9.00

Replacing the "new" process

10.00

→ OK now y have two process running that are identical - now what...

11.00

→ if you just want concurrency, you are done, but if you want a new process, then...-

12.00

1.00

→ EXEC() function

2.00

→ immediately stops the child process

3.00

→ The new process is loaded in its place

→ Begins executing the new program

4.00

→ Exec will never return unless an error ~~sfiz~~ occurs

5.00

→ This new process takes over the child processes PID and also the Paren PID

6.00

→ EnecvP()

MONDAY

31 January

Wk. 06 • Day (031-334)

JANUARY 2022

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

9.00

Process States

10.00 → The process is built and executing, so now we enter the process state machine

11.00 → States of a process

12.00 → New = When a new process is being created

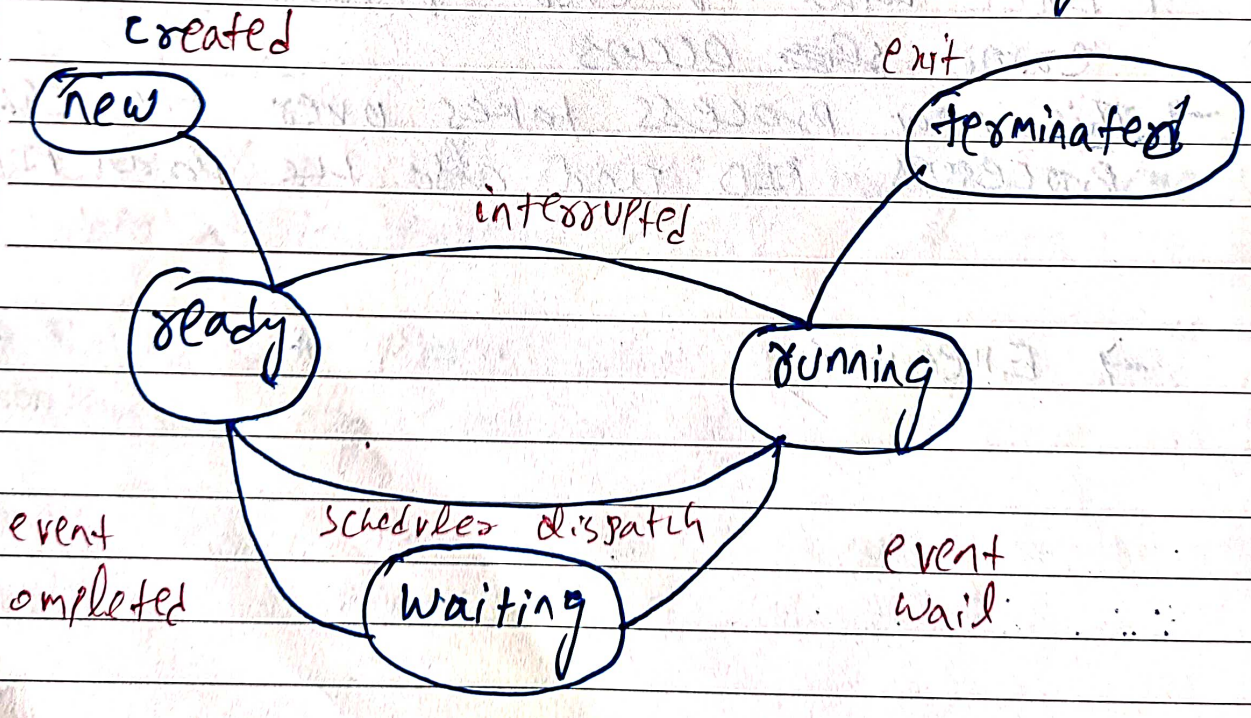
→ Running = Instructions are being executed

1.00 → Waiting = The process is waiting for some event to occur

2.00 → Ready = The process is waiting to be assigned to a processor

3.00 → Terminated = The processes has finished execution and is exiting

4.00



5.00

6.00

2022

9.00 ## Managing Process

10.00 → Linux kernel tracks what each Process is doing
11.00 → Process is assigned a Priority (compiler or user)
→ Address space assigned to the Process
12.00 → Files is the process allowed to access
→ Parent
1.00 → Child
→ Zombie
2.00 → orphan

3.00 ## Context Switch

4.00 → When the scheduler wants to switch another process, it MUST save the current executing process
5.00 → When the original Process is switched back in again, its original state is reloaded
6.00 → This time wasting activity is known as a content switch or scheduling jitter.

Process Termination

→ When a parent forks a child, they can finish is any order

WEDNESDAY

02 February

Wk. 06 • Day (033-332)

FEBRUARY 2022

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

9.00 → Sometimes the parent process could encounter and error and die.

10.00 → Sometimes the parent process will just wait around until the child process all complete before exiting.
- it calls wait() command.

12.00 → There are a variety of wait commands.

1.00

htop, PS, top (commands)

2.00

3.00

4.00

5.00

6.00