→ A mother program to schedule and run other programs is known as OS (Operating System)

→ OS contains a core (Mother program) known as kernel.

SMP → Symmetric multi processing

Ex: Mobile phone

ASMP → Assymmetric multi processing

Ex: Mobile with cortex processor, GPU (Graphical processing unit)

→ GNU → Initiated by Richar M Stallman
↓
Not Unix

→ Linux (man ls) → written by Richard M. Stallman and David Mackenzie.

→ ls cpu → Details about CPU cores and Threads usage per core

→ Multiple utilization of threads (Threads per core: 2) is known as hyper threading

→ Dual mode ~~kernel~~ have 2 modes
- → User Mode
- → Kernel Mode

Alternative names for kernel mode
- (Super Mode)
- (Admin Mode)
- (Root Mode)

→ How to differentiate?
↓
Mode Bit

ls → Shows Sequence of Activities Behind

Strace → These Activities can be traced

→ What happens when function is called?

Ans) Stack space is created

Scanned with CamScanner

→ By default user space - 3GB.
Kernel space - 1GB.

→ User mode - System Utils and libraries
Kernel mode - kernel and drivers

→ ps-aex
→ ps -cat } → Execute and check yourself

→ A highly preserved data ~~with the~~ which ~~gs~~ is executed when flag is set up. The CPU jumps to the address of ISP. This is known as Interrupt

→ man man → Reference manual.

→ OPEN (System call)

Doubt: Refer google for clarity.
→ System calls are also known as software
Note: Not every system call is a software interrupt

→ File Discriptor (Discribes the type of file)

→ 0666 - Enables read/write perms user, group, etc

→ perror vs printf

perror	printf
* Writes to stderror	Writes to std out
* Non buffered (prints immediately)	Buffered (Prints when hits /n)
* This is a library call	* found in Standard C library (<stdlib.h>)

* found in <sys/syscalls.h>

→ fd (file Descriptor): Creates a table to read / write the values. Ex: read (fd, buf, max len)
write (fd, buf, n bytes)

Ex:

Scanned with CamScanner

If $f_d = 0$ (stdin - read)
$f_d = 1$ (stdout - write)
$f_d = 2$ (std err)
$f_d < 0$ (Returns in case of error)

→ Refer chunk by chunk program (If no while loop provided)
nbytes == maxlen (Read all the contents of file at one go)
nbytes < maxlen (Read content until provided maxlen and the consider next content as NULL)

nbytes = 0 (Indicates End of file or file is empty).

Points missed before

→ Memory mapping is stored in .out file
→ .out file contains → (.rodata)
(.data)
(.text)
(.bss) etc ---

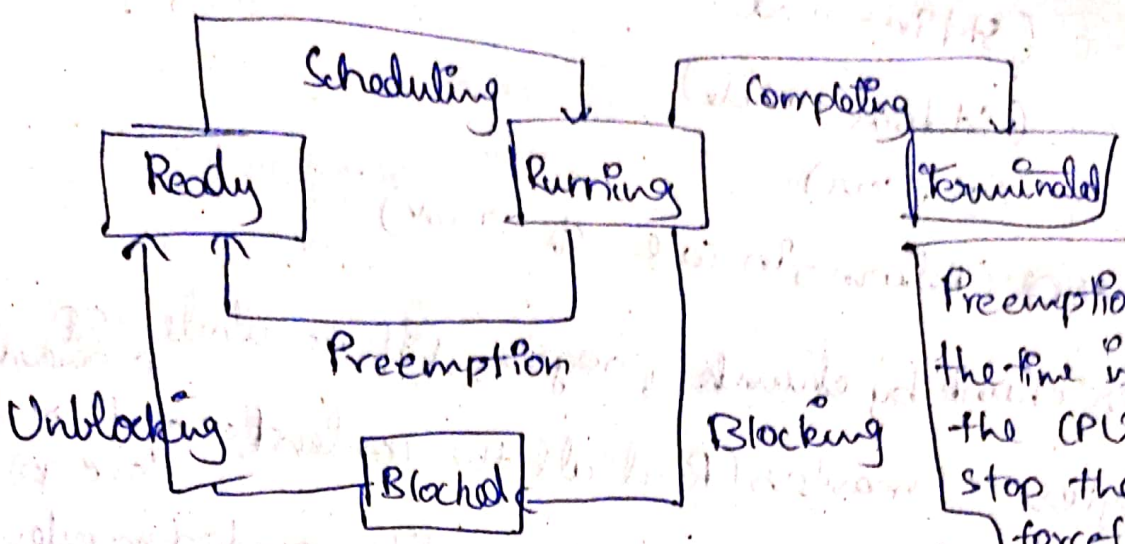
→ Path types
→ Absolute (From Root) Ex: echo $ PATH
→ Relative (From Current directory

~~System calls~~ ~~Special~~ → Special purpose registers
→ Frame/Base ptr
→ Program Counter
→ PSW/flags
→ Stack Pointer

Process:

→ Program is a passive entity and process is active entity
→ Every process has its own independent stack.
→ Kernel maintains process list table in form of double linked list
→ Each process has its own unique id (PID)

Scanned with CamScanner



Preemption : When the time is completed the CPU might stop the process forcefully. This time is known as quantum or time slice. This is when preemption occurs

→ Kernel maintains a process table.

Content Switching
- → Content Saving
  - ↳ Data from CPU to save area
- → Content loading
  - ↳ Data from save area to CPU.

↳ Occurs on system call or Interrupt or preemption (Utilising CPU resource forcefully)

cat → Opens file

cat file.txt & → Starts file.txt in background

fg → Switches process from background to foreground

→ To get process ID (PID) and parent process ID (PPID)

→ Orphan process - Parent is killed/terminate before itself

→ zombie process - Process is completed but still has entry in process table

→ Daemon process - Process which runs at background

Scanned with CamScanner

What is the Origin of linux process hierarchy? → Init

→ pstree → will give process and tree

top → CPU and memory utilization

pgrep → used to search process

kill → to kill the process.

~~Signals~~ fork() → Creates a new process known as child

→ Copies all the parent information

→ For child process return value is 0 and for parent process return value is greater than 0

→ Parents process prints child ~~ID~~ process first and then child process prints copy of parent process.

get pid() → Process id

get ppid() → Parent process id

wait pid → wait until child is terminated

→ Syntax: pid_t waitpid(pid_t pid, int *wstatus, int options)

Ex: waitpid(-1, &status, 0)

WEXITSTATUS → Prints exit status removing extra information at exit process.

WXETTSTATUS (& status) → To get only exit status of current process.

→ -1 for current child process.

Scanned with CamScanner

Important:

(exec)

→ If you want to execute a different executable file then we use exec, #include <unistd.h>

→ all commands (execl, execlp, execle, execv, execvp, execvpe)

→ Syntax: execl(char *pathname, const char *arg);

Eg: execl("usr/bin/ls", "ls", args..., NULL)

Syntax: execlp("ls", "ls", char *file name, const char *arg

Eg: execlp("ls", "ls", args..., NULL)

Syntax: execvp (char *filename, const char *arg

* char* arggs[ ] = "cp file1.txt file2.txt"

Ex: execvp("cp", arggs)

→ Command Line arguments

* int main(int args, char* argv[ ]) { }

Signals:

→ Operates at process level (Similar to phone notifications)

→ Used for abnormal termination, illegal memory and events that goes wrong. Eg: no/0.

→ Signals are similar to interrupt with no interrupt vector table

→ Signals b/w process → SENDER, → TARGET

Scanned with CamScanner

SENDER → Sends / Triggers signal from one proce to other

TARGET → Sets the bit based on senders signal bit. Looks up signal handler table for addresses for each of signal related handle fields

* SIGNALS in common Actions:

SIGINIT → Sends INTERRUPT signal (ctrl+c)
SIG QUIT → User sends Quit signal (ctrl+\)
SIG STP → User sends suspend signal (ctrl+z)
SIG TERM → User sends termination signal (kill <pid>)
SIG CHLD → Stops child process
SIG FPE → Floating point exception
SIG CONT → Resume process
SIG SEGV → null ptr (Segmentation error)

→ Non maskable Interrupts → SIGKILL (Code: 9)
→ SIGSTOP (Code: 19)

→ Custom handler can override the default handler

→ Kill Commands:

→ kill -l → Gives list of all signals
→ kill -SIG xxx <pid>
→ kill -<signo> <pid>
Eg: kill -9 <pid> → terminate process. Sure kill
→ kill (pid, signal number) - System kill
↳ To use kill as a function in code. For syntax refer google or man 2 kill
→ For other types of kill refer man kill

Scanned with CamScanner

Points missed before

→ Types of kernel
- → Monolithic kernel (OS completely works on kernel)
- → Micro kernel (In the form of layers or plates)
  Eg: Embedded system
- → Modular kernel
  Eg: Linux (like Monolithic but modular)

* Dynamic modules - Can be configured any time
* Static modules - Loaded along with system boot

→ API : Application program Interface
ABI : Application binary Interface
↳ Identify system call number
↳ Save user mode content
↳ Store system call number in a register
↳ Store parameter in other General purpose register
↳ Call the trap instruction

ps -el → additional process data

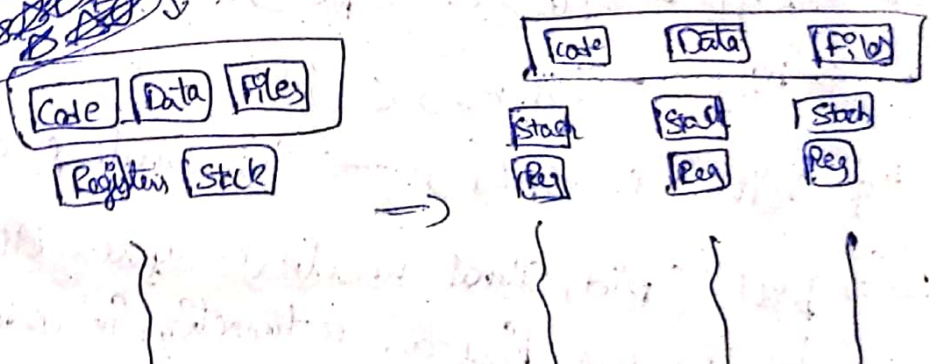
pstree -np → Parent process ID

pstree -rp/less → One page at a time

init → (Super parent) First parent to start

→ Preem

Threads



Single thread

Each thread has different stack and registers

Scanned with CamScanner

They are
→ Various subactivities within application
→ Referred as Light weight process (LWP)
→ Multiple Threads run concurrently
→ Threads have seperate thread control block
→ Threads Private / local data is not shared.
→ To use threads use -lpthread Eg: gcc filename -lpthread

Significance:

★★ If one threads makes a blocking call whole process gets blocked
→ Concurrent execution
→ Resource sharing
→ Child process will have own resources, but threads will have shared resources
★★ → Faster than fork
→ Scheduled threads interchangeably use cpu based on time sharing.
→ Every process is run initially as single thread then multiple threads spawn.

Types of thread:

Thread
- → User thread: Threads are used by applicat programmers above kernel, without kernel support
- → kernel thread: Supported within kernel performs multiple simultaneous tasks to serve multiple kernel system calls.

Scanned with CamScanner

Models:

→ Used to map user threads to kernel thread.

- ↳ Many to one : Threads management is handled by thread library in user space.
- ↳ One to One : Limitations in count of thread that can created
- ↳ Many to Many

POSIX (Portable operating system Information Exchange)

↳ These are application / user level threads

↳ commands:

pthread_create
pthread_join
pthread_self
pthread_equal
pthread_yield
pthread_cancel

Extra points and commands:

pwd → Present working directory

whoami → User (Prints user name)

Paths
- → Absolute (starts with /)
- → Relative (Doesn't start with /)

~ → Gives absolute path

echo $PATH → Show the executable files ~~path~~ directories that are there

Uname -a → To get the name of operating system (GNU/Linux)
(-a is additional information)

Scanned with CamScanner

Uname -O : Same as above but gives GNU/Linux

mv → Used to rename file

char *str = "Hello world"

↳ Stored in stack

↓ String literals (Stored in RO data)

→ Any local data is stored in stack

→ Any data if global than stored in data.

* MemCpy, MemCmp (Working with memory copy)

→ where is pwd.

↳ To know pwd executable file path

char *str → Pointer to String

char *str[] → Pointer to String Array

char **str → Pointer to ''

char str → Character

char str[] → Character Array.

Files syntx: To open file: open(const char* pathname, int flag)

Eg: fd = open(filename, flag)

To read contents in file: size_t read (int fd, void *buf, size_t count)

Eg: read (fd, buf, length)

To write contents in file: size_t write

Eg: write (fd, buf, length)

Scanned with CamScanner

To print on console replace fd with one.

<u>To close the file</u> : close (Prt fd)

Eg : close (fd)

Checkout examples in GEA learn once completed this notes.

Scanned with CamScanner