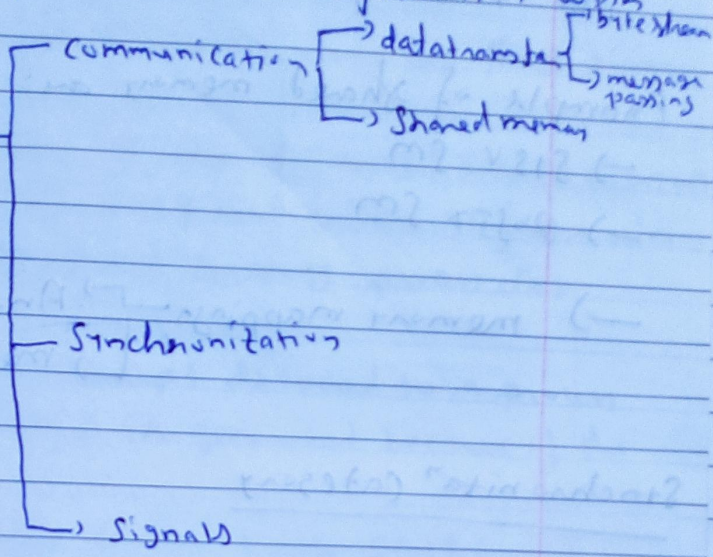
Inter Process Communication: - It deals with the different ways in which processes share information with each other.

Categories of IPC:



In communication category, output output of one process is treated as input of other process. There are 2 sub categories of Communica": (i) data transfer (ii) Shared memory

In data transfer technique, synchronization among the processes handled by the kernel whereas in shared memory, process exchange information using a shared memory. Here, kernel is not involved.

Data transfer tech. is further divided into two categories (i) byte stream (ii) message passing.

In byte stream, each read operation may read any no. of bytes regardless of the size of the block written by the writer. e.g. pipes, FIFOs, stream Sockets.

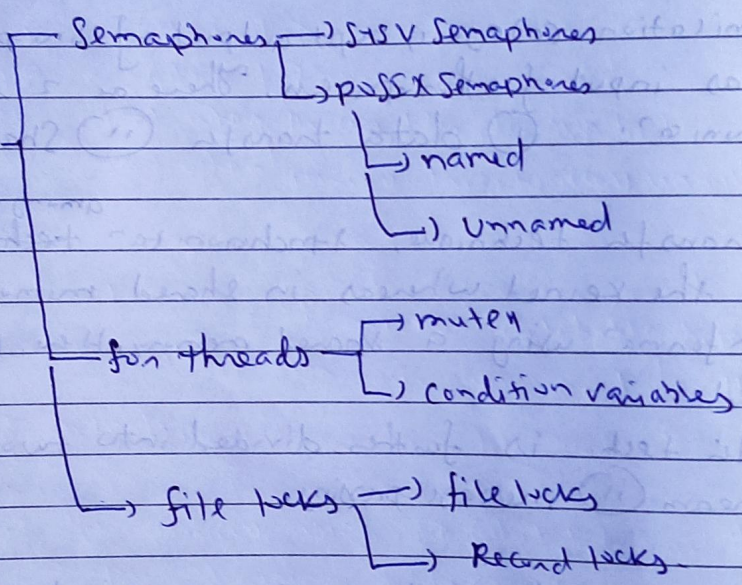
In message passing, the reader operation reads whole msg. from the IPC facility. It can neither read part of the msg nor multiple msg.

eg. - SysV MQ, POSIX MQ, Datagram Sockets
MW stacks for message queue

Example of shared memory are:
→ SysV SM
→ POSIX SM
→ memory mappings → Anonymous mapping
→ memory mapped files

Synchronization category

It allows process to co-ordinate their actions in order to avoid simultaneous updating of shared variables in the shared memory region.



M T W T F S S
Page No.:
Date:
YOUVA

Signal category

Signals are used by kernal to notify processes that some event has occured.

By pressing ctrl + C, we ask the kernel to send SIGINT to the currently running foreground process. After receiving SIGINT signal, the foreground running process dies.

Signal is a software interrupt delivered to a process by OS. because of the signal is generated because of the following reason:

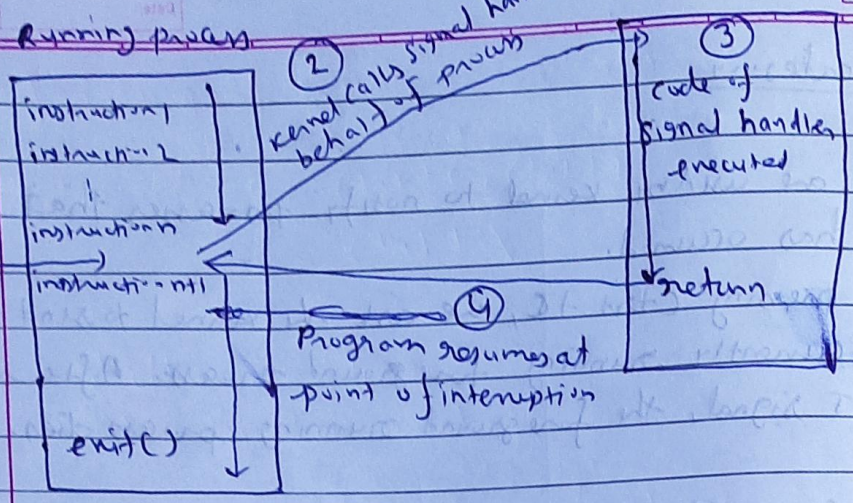
→ the user did something, i.e. e.g. pressed ctrl + C (SIGINT(2), SIGQUIT(3), SIGTSTP(20))

→ the process did something. e.g. division by zero. (SIGFPE(8), SIGSEGV(11), SIGILL(4))

→ one process wants to tell another process something (SIGCHILD(17)) e.g. signal is given to the parent process when child terminates.

→

Whenever a process receives a signal, it is interrupted from what ever it is doing and forced to execute a piece of code called signal handler. When the signal handler function returns, the process continues execution as if this interruption has never occured.



$ kill -l

$ less /usr/include/asm-generic/signal.h

SIGINT :- Ctrl + C

SIGQUIT - Ctrl + \

SIGTSTP - Ctrl + Z

SIGFPE - Sent when process does division by zero

SIGSEGV - Sent when process tries to access unauthorised memory.

SIGKILL and SIGSTOP are non-catchable signals

i.e. we can't write our own signal handler functions for these signals.

→ Stop the process using SIGTSTP Ctrl + Z and then resume the process using SIGCONT

→ check the status of the process using PS -l

T stands for stop

→ Check the status of process after

kill - SIGSTOP pid

SIGCONT

now bring the process in foreground
$ fg %1

⇒ how to make a process ignore a signal or write own own signal handler function

trap is a builtin command, so can't be seen using 'man' command. Its manual can be seen using the command 'type'.

$ trap '' 2 ! 2 is no. of SIGINT in (ctrl+C)
now ctrl+C will not work. or kill -2 pid will not kill the process if run from the terminal.
to see that which signals are trapped

$ trap -p

to trap multiple signals
$ trap ' ' signal no 1 signal 2 signal 3 ...

SIGKILL(9) and SIGSTOP(20) can't be trapped.
i.e. kill -9

$ trap 'hello Jnv' 2
so, whenever ctrl+C is pressed, 'hello Jnv' will be printed in place of terminating the process.

to untrap a signal
$ trap - signal number

$ trap - 2 for SIGINT

ways of issuing of signals

→ by keyboard - e.g. Ctrl+C, Ctrl+Z,
Ctrl+\

→ using shell command e.g.
kill -signal no pid

kill pid (for the signal system(15))

bg command gives SIGTSTP(20)
fg gives SIGCONT(18)

→ using kill() system call

→ implicitly by a program
e.g. division by zero, issuing an invalid address,
termination of a child process.

Behavior of process when OS delivers a signal

① Term:- abnormal termination of process with no core dump file generated (core dump file keeps details how & why process terminated)
e.g. SIGHUP(1), SIGINT(2), SIGKILL(9), SIGPIPE(13), SIGALRM(14), SIGTERM(15)

② Core:- terminate the process with core dump file generated e.g. SIGQUIT(3), SIGILL(4), SIGFPE(8), SIGSEGV(11)

(iii) Stop:- to stop the execution of the process.
e.g. SIGSTOP(19), SIGTSTP(20) Ctrl+Z

(iv) CONT:- Continue the execution of stopped process.
e.g. SIGCHILD(17), & SIGCONT(18)

Kill() System call
int kill (pid_t pid, int sig)

If sig = 0, no signal is sent
sig is signal no. or signal name.

If pid > Pid is the pid of the process to which we want to send the signal.
if pid = 0 then signal is sent to every process in the process group as the calling process, including the calling process itself.

If pid = -1, then the signal is sent to every process for which the calling process has permission to send a signal, except 'init' & the calling process.

raise() can be used by a process to send a signal to itself.
int raise (int sig)
or
kill (getpid(), sig);

→ In signal.c program, OS will send SIGFPE signal to show the ~~error~~ floating point error.

signal()

→ to change the disposition of a particular signal, a programmer can use the signal() system call, which installs a new signal handler for the signal with number SIGNUM.

→ The 2nd parameter can have 3 values:

(i) SIG_IGN: the signal is ignored

(ii) SIG_DFL: - the default action associated with signal occurs.

(iii) A user specified func^n addres, which is passed an integer argument & returns nothing.