05

OS →

Condition of deadlock

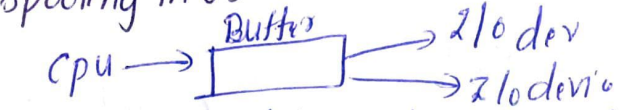
① mutual exclusion
② Hold and wait
③ No premption
④ ciucular wait or resouce wait

Belady anamaloy?

↑page fault & ↑no frames

↑ this abnormal behaviour is called

Spooling in os →

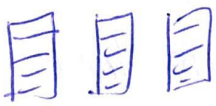


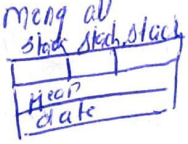
Buffer is used to match speeds of cpu and I/o devices

Sections in process →

Stack ↓	→ local variables, & Returns addres
heap ↑	→ dynamic memory allocatn
Data	→ stores global and static variable
code	→ compiled program code

process	Thread
① program + PCB	① smallest execution unit of process is called thread
② seperate data space	② data of parent is shared among all
③ PCB a heavy weight	TCB a light-weight





kernel → system program responsible for scheduling, managing memory, communicatn. of proces it is core part of OS

OS → is group of programs running so that management of resources is done efficiently

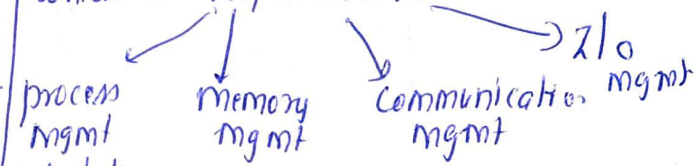
kernel	OS
it is system program of OS.	it is system softwar
it manages memory communication, multiprocess mangemn	it provides filesystem security, permissio maintain system priva
it is interface b/w hardware & appl	it is interface b/w hardware & user

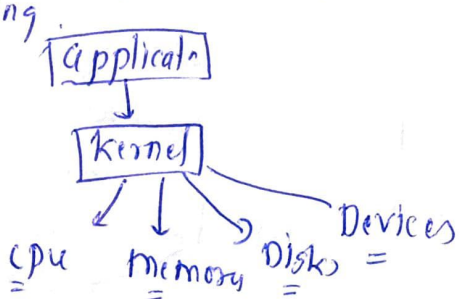
Context-switching → it is proces in which the details of stat of currently executing state is stored on stack and an another process is PCB loaded on cpu

Symmetric multiprocessing process in which processes are scheduled on all cpu's rather than scheduling on master cpu

kernel

① it is system program of OS - core which is responsible for





Race around & data inconsistency.

as multi processing multithreading systems have common resources and while accessing them the result in data inconsistency. which can be avoided by locking the resource

Symaphore	mutex
⊗ it is managed by os signals	it is lock object
⊗ it is slow due to involvement of os	no os is involved
⊗ signals are used	no signals are used.
Binary & counting	Mutex

Starvation :- it is situation in which low priority process dont get a cpu due to present of high priority process for long time.

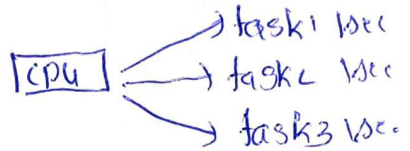
aging :- it is solutn of starvatn in which we increa priority of process as time passes

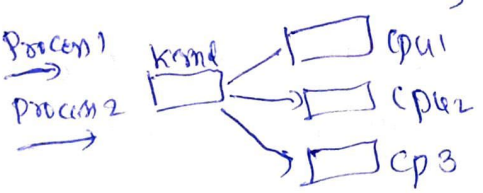
cascading termination :- if parent is dead compulsory child also dies

Zombie :- if child process completes but after completn still we have pcb alive in kernal of dead process, on its Zombie process

socket :- it is combination of port + IP for inter-process communicatn over network.

multitasking :





⊗ tasking → one cpu sharing its time among all process

Asymmetric clustering :-

when there are multiple cpu's one cpu acts as master and according assigns tasks to slave.

multiprogramming :- it is featur of os which enables it to submitt multiple process at at time : so that cpu utilisatn is improved drastically.

Thrashing :- is process in which os is give more time in swapping the pages rather than processing on that pages

paging :- program is divided into fixed size blocks called as pages and memory is divided in fram. size of frame size of page
internal fragmentatn

segmentatn :- in which program is divided into different segments
→ external fragmentatn

scheduling algorithm

① FCFS ② Shortest-job first
③ Shortest remaining time first
④ Round Robin

⊗ all algorithms are used to ensure that no process starvs. & no process holds cpu for more time.

Process :- is a program who is having 'pcb' in kernal space.

28 **States of process.**

① **New state** :- when pcb gets grds created into kernal space is considered new state

② pcb in 'job queue'.

28 **Ready-state** :- state in which process is waiting for the 'cpu' after creation its pcb is in Ready Que

29 **Running** :- process under execution is called Running

21 **Waiting**:- when process asks for any I/O device than that process is kept in waiting state

Terminated:- when pcb gets destroyed from main memory

Thread:- is smallest indivisible unit of process, it is smallest execution unit of process, it is lightweight process,

* thread has thread id, program counter and shared memory of process registers.

They share :- address space, heap, static data, code segment, file discriptor, Global variable, child processes, pending alarm, signals, signal handlers

virtual memory:- if we have ram of 8GB than swap area = 16GB, but OS pretends that we have 24GB of ram by adding swap area which is actually not Ram space

IPC :- it is service provided by OS through which correlated/cooperative process could communicate through OS

① shared memory model:

② message passing (pipes)

③ signals

④ socket.

⑤ semaphor

⑥ pipe

process synchronisation :- many a time in os situation occur where two or more process try to access a common resource, and accessing common resource may lead to data inconsistency where synchronisation is required

* by synchronisation process communicate with each other and data inconsistency is avoided

Demand paging :- when cpu process is executing ~~if it requires~~ process if it requires any page, than that page is fetched into memory / on demand if any page is never used is never fetched.

pure demand paging.

① page is valid ⟶ then process on mm

② invalid → checked in secondary ← fetched.

Bootstrap program :- is a program responsible for loading kernel from HDD to main memory

* If any storage device has BP then that is called Bootable device if first secto.

54

Semaphore operation

wait(), signal().

Pipe:- it is service of kernel through which two interrelated process can communicate each other, it has two ends read & write end, it unidirectional

- named & unamed

---

mcq dear

- OS is interface b/w hardware & user.
- FIFO is non-premptive.
- Time sharing os read and read in terms of actual time
- two types of threads → user & kernel
- example of spooled device is a line printer
- Bankers algorith is used to prevent dead lock or.
- exec system call does not return to calling functn as it stack gets replaced by another process
- main

```
fork
↓ ↓
fork
↓ ↓ ↓ ↓
```

Count after fork ?.

- get pid never returns error
- thread concellatn - terminating before completion
- Command interpreter is also called shell
- the only state transition initialised by user process is dispatch.
- most optimal cpu scheduling algorithm is SJF it gives least avg waiting time.
- Binary Semaphore is initialised with value 1.
- release and request are system calls
- Relocatn register = base register
- MS-DOS → single user OS
- size of virtual memory is dependant on size of address bus

OS

System program:- program which are part of OS
kernel, cpu scheduler, job scheduler, dispatcher, loader, device driver, memory manager.

Application/utility programs which have specific functionality
notepad, google chrome, calculator, Games, ms office.

user program: program written by user programmer in any language
source code

⊗ User can interact with with os using clz
↳ in Linux name of clz → shell
windows → cmd prompt

Unics
linux mac windows

name of kernel of Linux:-
VMlinuz

Swap area → partition in lonux os where running inactive codes are kept.
data is kept in (/)root directory

Swap = 2(mm) GB

pwd → print working directory
whoami → signed in user
clear → clear screen

cd → change directory

ls → list the content of directory.

mkdir → use to create new directory

rmdir → to remove directory (only empty)
rmdir -r path → used to delete non-empty directory recursively

cat → concanetate file content. & display
↳ can be used to write into file.
cat > india.txt → ctrl+D

rwx rwx rwx

chmod → change mod bit or change access permission

inode → unique identifier of file

cp → cp <source> <destination>
mv → mv <source> <destination>
Rename file mv <old> <new>

⊗ When execution of any program starts by default os opens three files for that process Character special device file
① stdin → temp buffer with std i/p device kbd by default program reads i/p from this file

② stdout → standard o/p buffer (monitor) by default program write o/p via
③ stderr → program writes list of errors into stderr

① all linux commands are programs only

① Input redirect <

② Output redirection >

③ error redirection 2>.

→ ls -l | wc → o/p of ls -l is given to wc → to count lines, words, char

pipe → used to give one out of file to another file

↳ creates internal buffer where o/p of one file stored

files in linux

① regular file (-) → all text files

② directory file (d) → name of files & subdirectory

③ character special (c) → kbd

④ block special device (b) →

⑤ socket files (s)

⑥ named pipe file (p)

⑦ linkable (l)

cd → / → go to directory '/'

ls → ~ → go to home
→ go to root
- → go to previously accessed file
. → remains in current directory
.. → go to parent directory

ls → -l → display in listing format

l RWX RWX RWX →
↓
Type of file

-h → size in human readable
-a → display all (including hidden)
-A → display all (exclud. ..)
-i → display inod

man → system ref manual

man man → display details of man command

tac → prints reverse of cat

Whatis →

whereis → shows path of argument

touch → create file / update time stamp

alias → create alias of large command

unalias →

Integer Return by program:

0 → successfull terminatn

>0 → erroneous terminatn

<0 → abnormal terminatn

pipe | head
tail
sort
uniq
cut
tr
len

Chmod

chmod +x filename → execute permission for all

-x filename → remove exe permision all

chmod u+x fn → user
g+r → group
o+r → other

chmod OABC filename (owner, group, other)

A → 101/110....
B → represent group
C → represent other

chmod 0777 filename all to all

Grep global search

search pattern/string/expression

grep -i ignore case

grep → global regular expression print

grep [options] Pattern file

e.g grep bash /etc/passwd → print lines

-v → invert match
-w → match whole word
-x → match whole line

grep -n "java" subram.txt
print line with serial no

-A2 → after two line
-B2 → Before two line
-C2 → after & before

Regular expression

to suprress effect of key char
// → to sho '/'

"four" | 4 → or

[...] accepts any one character

[abc] a or b or c

[a-d] a b c d

[^a-d] e f g h i j k l m n o p q r s t

Repeatition operator:-

+ → one or more [0-9]+ = 243, 123, 000

* → zero or more

? → zero or one
[+-] → '+' '-' 'empty'

{m,n} 'm-n'
{m} → exactly m times
{m,} → m or more
'm' or 'm+'

Shell script

Comment #

Varname = Value

' ' → char constant
" " → string constant
` `

If Condition
then
statement
Else statemen
fi

operators

- -gt → greather than >
- -ge → greto ≥
- -lt → less than <
- -le → ≤
- -eq → ==
- -ne → !=
- -o → or
- -a → and

Test command

- -e filepath → file path exist
- -f filepath → if regular file
- -d " → if dir file
- -w → if write permission
- -x —— " ——
- -r —— " ——

while	until [ ]
while (condition)	do
do	
	done
done	

for var in collection → 1 2 3 4 5 6 / {1..100..3}
do
statem
done

argv[0] → name of executable file
[1] → first argument
[2] → second Arg

$# → total print positional param

$0 → name of scipt
$1 → first argument
:
$x → all args

Function

```
function function_name ()
{
  echo → returns result to calling fn
{
  $1 $2 $3   → argument parameters if insid functn else they are positional param
```

sum=$(addition $n1 $n2)
echo "sum=$sum".

addition $n1 $n2

OS

Shell → is applicatn program

function of OS:-
- whenever program executn
  - resource allocatn
  - resource manage
  - Control program

.c —#→ .i —asm→ .s —compile→ .o —linker→ .out

[editor] ⇒ Hello.c ⇒ [preprocessor] ⇒ hello.i
⇒ [compiler] ⇒ hello.s/ ⇒ [assembler]
hello.asm
⇒ hello.o ⇒ linker ⇒ hello.out/.exe

Build → compile + link

Loader → (checks file format)
it loads .out/.exe from hdd to main memory

Dispatcher → loads program data from main memory to CPU registers

Linux → ELF } checked by loader
Windows → PE

Sections of ELF file format

① elf header / primary header / exe header
↳ magic no, _start(),

② BSS section → uninitialised global & static variable

③ Data section → initialised global & static variables.

④ ro data section ⇒ read only
↳ constants & literals

⑤ Code → executable instructn

⑥ symbol table → contain function and vars in tabular form

Booting → kernel → main m
HDD → bootstrap program

bootable → partition where starting of that hdd contains bootstrap program

System calls are functn of kernel

fopen() → (wrapper) → open() → sy call
fclose() → close() sy call

Bootstrap loader: searches for available bootable devices

Bootloader program: displays list of OS

Bootstrap program ⇒ kernel → mm

---

for execution of any program CPU switches b/w user programs and user programs converted to system calls hence dual mode

User mode = 1
System mode = 0

PCB (>1kB)
↳ pid
↳ ppid
↳ pc
↳ context switch info
↳ resource info.

New state → when submitted by loader
↳ job Queue

Ready state → ready Queue
↳ may be process active or inactive

Running → on CPU
if I/O → waiting Queue

(*) Multiprogramming:- more than one PCB can be submitted at a time

(*) more than one PCB gets created at a time

Job schedular: it is program which decides which program to put in ready queue from job queue

→ long term schedular

cpu schedular: it is system program which decides which program to schedule in ready queue

short term Scheduler (frequent context)

dispatcher → gives control of cpu to process

Context switch: When high priority process arrives in ready que the the state of low priority process gets saved on pcb

state-save → done by interrupt handeler

State-restore → done dispatcher, put the state from pcb to cpu register

cpu schedular call →

non-premtive → natural release of cpu

waiting time: time spent by process in ready queue to get cpu

response time: it is time to get first response after submission

(5) turn around time → total time required for process, from submission to termination

Scheduling algorithms

FCFS:- process gets cpu as per time of arrival

↳ non-premptive

↳ Convoy effect:- shorter process waiting for getting cpu due to larger processes before them,

SJF (shortest job first)

↳ the process having minimum burst time gets cpu first

↳ starvation can occur of larger processes

(4) round robin → premptive technique where every process gets time in equal fashion

fork() → used to create new child process

exit() →

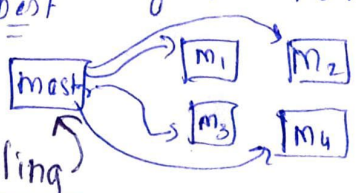
wait() → can be called from parent

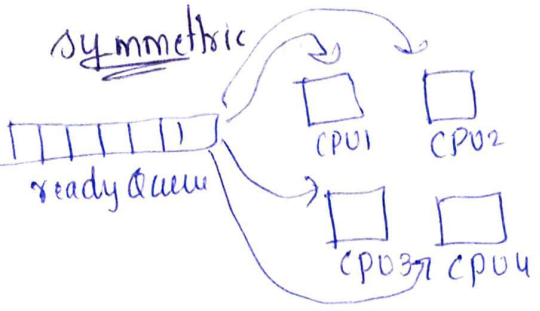
exec() → used to call another process into memory

asymmetric multiprocessing

(*) scheduling is done for master only hence inefficient

(*) utilisation of cpu is not at its best





fork() :- used to create new child process

2, 4, 8, 16

⊛ in parent child fork returns value

Orphan → parents exit before child

Zombie → child exits before parents 'pcb' not destroyed and process becomes zombie

wait() → to avoid zombie state

exec() → loads another process into currently running process

execl → list

execlp → path variable (takes from std path)

v → vector

e → environment variable

⊛ exec → overwrites childs PCB

execv → auto detects from paths

⊛ Thread id & pid of main thread are same

Inter process Communicatn

Shared memory model

Shmget() → used to get shared memory region from kernal returns shmid

Shmctl → to destroy shared resources

Shmat → to attach to shared

pipe

mkFifo → to make pipe

⊛ named pipe :

↳ related process of same parent

↳ only related process can communi-cate

ls | wc → unamed

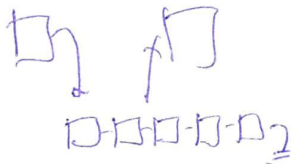
↳ named can be used by non-related as well

Msg Queue

msgget() → to create entry in table

msgsnd() → to send

msgrcv() → to recieve



Signalling

⊛ Sigterm → to terminate normally

⊛ Sigkill → terminate forcefully

kill → used to send signal

kill pid (kill -sigterm pid)

kill -l

signal() → used to register signal handler

SIGSEGV → os sends when segmentatn fault

signal handler ⇒ then that process nor terminates nor stopped

↳ does not respond to sigkill

- Signal handlers are called internally
  SIGCHLD → register handler with OS
- When child terminated. wait also called
- To avoid zombie zombi

Semaphore	mutex
binary / counting	object not manipulated by OS
OS is involved and signalling is used to ↑ & ↓ to count	locked unlocked
→ Slow due to OS involvement	fast due to OS not involved

~~Condition for deadlock~~

Deadlock :-

1. Mutual Exclusion → at a time only one process can aquire resource
2. No premption :- no forcefully taking away cpu
3. Hold & wait :- every resource is holding one resource & waiting for other resource
4. circular wait = $P_1 \xrightarrow{\text{hold \& wait}} P_2 \rightarrow \cdots \rightarrow P_3 \rightarrow P_1$

1. Dead lock prevention :- by discarding any one condition of above (4)

Detection & avoidance

befor allocating resource check with algorithm if dead lock is possible or not

1. Resource allocatn algo
2. Bankers algo

Recovery

1. process termination
2. Resource preemption

Memory management

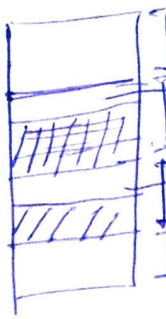
- inactive processes are swapped out of memory when not required

relocation registers

- used by 'os' for relocation of program into main memory
- Values of Relocatn register will be copied in pcb
- cpu call compiler generated address (logical) that are mapped using relocatn registr.

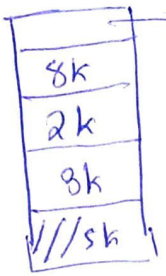
Contiguous memory allocatn

1. fixed size partition scheme
   - linked list of all free partition is maintained & allotted accordingly
   1. first fit
   2. worst fit → largest space allotted
   3. best fit → optimal space alloted



Internal fragmentatn
↑ when space remains after allocatn

Non-contiguous dynamic size



→ free → external fragmentatn.

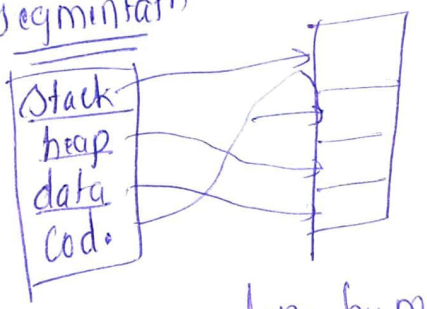
↳ memory is available but it is not contiguous hence cannot be used

⊗ Memory is allotted as per size

Noncontiguous models

① segmentatn ② paging

Segmentatn



⊗ swapping is done by memory manager program

mmu → hardware → Relocatn registers.

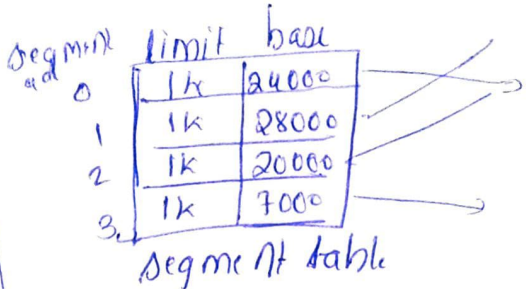
Logical → Physical

memory is available but unable to consume as it is not contigu

is External fragmentatn

Segmentatn table: to keep track of all segment in memory

segmentatn table is used one per process



Segment table

Page table

↳ to store all page details where are pages present in MM

⊗ mapping done using relocatn register

Virtual-memory management

main + swap

8 + 16 = 24GB

12Gb process can run easily

demand paging

pure — ii — → fetched only if required

Belady Anamoly → this abnormal behavio

frames ↑ page fault ↑ → in fifo

PagRA