5/9/19

Process Synchronization

Server : Everytime there's a request, a variable 'id' is incremented. → Each thread of this process will be incrementing this variable

Sequential service to process

Server : will process 1 request, handle it & then process 2nd ... :→ Not good

↓ Problem

Other requests are waiting in queue.

B Good way of doing it : create multiple threads, each thread having one request

↓ has

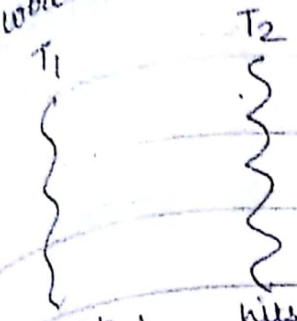
personal space.

Each of these threads can access global data

Teacher's Signature

Original value = 10

Global variable : hits ++



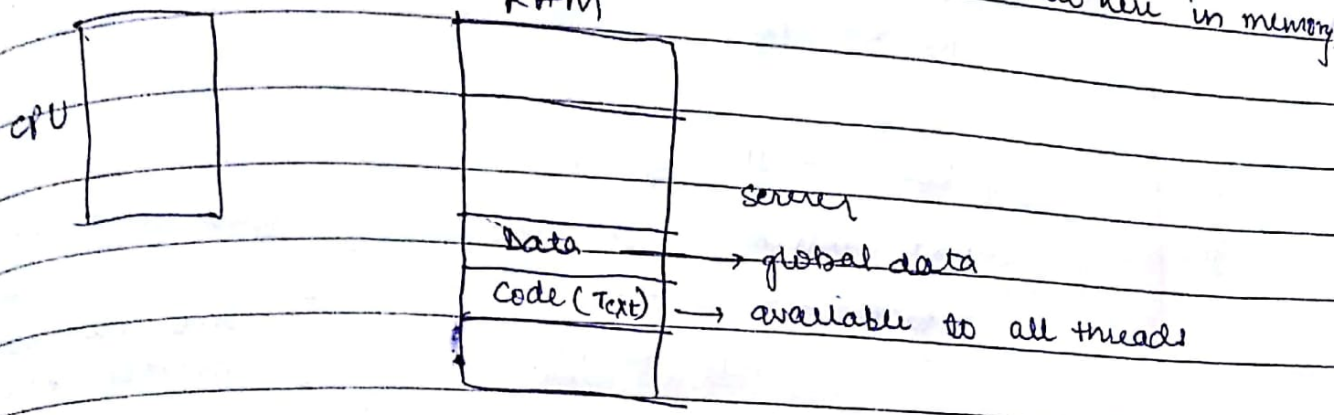
→ Both came simultaneously.

→ know how many requests have come from this variable

Ideally : hits = 12

Actually :

Problem :- hits : global variable stored somewhere in memory



How it'll be incremented ? : has to be brought to CPU, ALU & then stored back.

T1	R1, R2: may be private to this process	T2
R1 ← hits (M) (LOAD)		R2 ← hits
R1 ← R1 + 1		R2 ← R2+1
hits ← R1 (STORE)		hits ← R2

↳ global variable

T1 & T2 are sub processes (If running on separate processors : can have 2 diff. R1's, same : 1st T1 then T2 : both can use R1 in this way)

→ Suppose t_1 executes before t_2 ↳ will get hits = 12.

→ T_2 - - - - T_1 ↳ " "

→ Problem happens because they're not serialized. Context switch happening in

→ Suppose T_1 is running 1st

$R_1 \leftarrow 10$

→ Context switch happens

$R_2 \leftarrow 10$

:

hits $\leftarrow$ 11

Context switch

$R_1 \leftarrow 11$

hits $\leftarrow$ 11

⇒ Race Condition

Both are trying to update a global variable & not getting correct value

Sol": Processes should be serialized

⇓

① Mutual Exclusion should be maintained when shared variable is to be accessed

↓

there's no loss of information

→ Acquire a lock Unless I unlock, no one can enter critical section

```
acquire(lock);
hits ++;
release(lock);
```

```
void acquire (struct lock * l)
{
    while (l=1);        // if lock = 1 ⇒ just have to wait
    l=1;                // if lock = 0 ⇒ make l=1 & enter critical section
}

void release (.....)
{
    l=0;
}
```

→ It's not that T_2 is stopped from handling request. Only in critical section, they're to go sequentially.

$i = 0$ (Initially)

T_1	T_2
while ($i < 10$)	while ($i > -10$)
$i++$;	$i--$;

→ T_1 runs before T_2 : final value of $i = -10$

→ T_2 ——— T_1 : " " $i = 10$

→ But we don't know how CPU is handling it. (Order)
Final value = ? can't determine it
Taking lock will be good idea

T_1	T_2
acquire (l) ;	acquire (l) ;
while ($i<10$)	while ($i > -10$)
$i++$;	$i--$;
release (l) ;	release (l) ;

② There should be progress : If a process wants to enter CS & no other process is in CS, it should be allowed to do so.

③ Bounded wait : suppose 2 ~~threads~~ processes want to enter CS. P1 is allowed. As soon as it finished, it again went to CS. P2 will keep on waiting.

↳ ∃ an upper bound how many times a process can enter CS if some other process is waiting to enter CS.

↳ This process of incrementing 'hit' (3 steps) can be made atomic (should not be interrupted in b/w)

→ OS : disables interrupt while Kernel updates data structures. We don't give this power to user.

↳ H/w has given support in form of inst^n :

xchg : exchanges values of memory & Reg. in 1 go (swap)

R ⟷ M

→ Suppose :

Reg = 1

xchg (Reg, M)

↳ whenever lock is free

→ void acquire (struct lock * l)

{

while (l == 1) ;

l = 0 ;

}

Other shouldn't enter this part

This part has to be atomic

Both T1 & T2 are running this code

T1 comes here (let) & stops

T2 has also seen l=0 & put l=0

T1 has put l=1

again a problem

Soln

Teacher's Signature

problem:

1. while (l == 1) ; we keep on checking & wasting CPU time
   ↳ Spin lock

we need use XCHG ; preserves atomicity.

2. acquire (____ , ____)

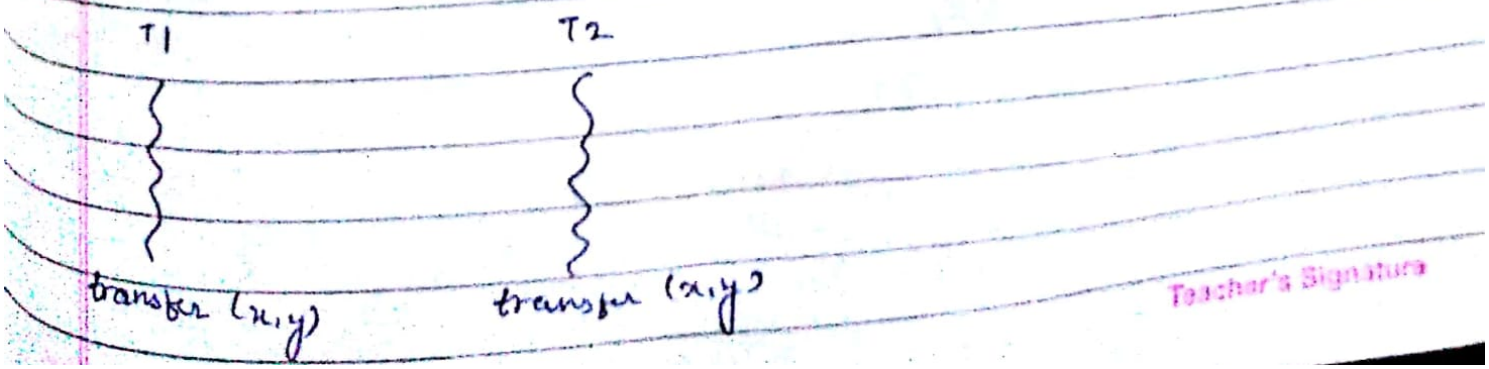
```
{
    Reg = 1
loop: xchg ( Reg , l* )
    if (Reg == 0)
        //enter CS          // This mean lock was 0 & now (unlocked) mem has value 1 ⇒ It has acquired lock ⇒ can enter CS
    else                    // This means lock = 1 ⇒ have to wait
        loop.               go back to loop
}
```

* maintained atomicity : checking + acquiring

~~a=0~~ a=1

3. void transfer (struct account *a , struct account * b)

```
{   acquire (l)
    if ( a → money > 0 )        // transferring ₹1 from account a to account b.
        a → money --;
        b → money ++;
    release (l)
}
```



Teacher's Signature

T_1 runs : checks cond") both' ll -- & value
T_2 runs : --) will be -ve

→ a, b : shared variable → use lock → concurrency maintained
→ L : global lock ⇒ has locked all the accounts
suppose c, d : want to use transaction
↓ ∴ l = 1
won't be able to now execute .

coarse control

Fine - grain control :
↳ can have instead → 1 lock per account

can have :

T_1

acquire (x → lock)
—— (y → lock)
transfer (x, y)
release (x → lock)
—— (y → lock)

→ Concurrency is maintained

⟶ suppose T_2 : transfer (y, x)

T_1 | T_2

—— (x → lock) ✓ | acquire (y → lock) ✓
—— (y → lock) | —— (x → lock)

→ may be possibility of deadlock.

TO avoid

do Ordering

Lock :- Locking mechanism

Ordering :

i) on basis of acc. no.

ii) applying condⁿ like 1st x will acquire lock, then y.

→ suppose, there are 5 lockers : 5 have gone. when I come out, I've to come out & telling people that this resource is availabl.

↓

Semaphore : signalling mechanism

N resources processes (let)

when finished using a resource :— gives signal to other

S = N

wait (s)

signal (s)

→ void wait (s)

{

while (s <=0) ;

s--;

}

① again, busy waiting (checking again & again)

wasting CPU time

S = 0 ⇒ all resources are being used.

S < 0 ⇒ S = -3 ⇒ 4 processes are waiting

not getting from this code so we swap s-- & while (s≤0) to show how many are waiting

→ each process :

wait (s)

signal (s)

→ void signal (s)

{

s++; // releasing a resource

}

Teacher's Signature

Better way of doing : sleep till semaphore is unavailable

```
wait (s)
{
  s--;
  if (s<0)
    sleep (S);
}
```

→ as soon as s = 1, it'll get up.

↳ can be T

Suppose s = -3 → 3 processes waiting

1 process finishes → s = -2

Another → s = -1

As S=1 → it will wake up.

```
P : wait(S);
    // CS
    signal (T);
```

Suppose :

```
P :
Pf ("0")

Q:
Pf ("1")
```

Cond^n : 1st 0 should be printed, then 1 should be printed

↓

```
S = 1
P :
wait (S);
Pf ("0");
signal(T);
```

↓

want Q to run now. T=0

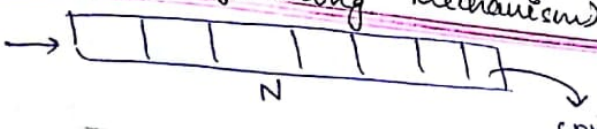
So, signal (T)

```
T = 0
Q :
wait (T);
Pf ("1");
signal (S);
```

→ serialize me use :→ wait & signal

Teacher's Signature

Producer - Consumer (Signalling Mechanism)



Initially, count = 0. As P puts something ↑ count ++;
C consumes ↑ count --;

```
producer () {
    produce (item);
    if (count == N-1) sleep (cv1);    // wait
    count ++;
    else
        produce (item);
        Count ++;                     ;→ if count ↑, ↑ should inform consumer
        wakeup (cv);
}
consumer () {
{
    if (count == 0)
        sleep (cv);
    else
        consume (item);
        count --;
        wakeup (cv1);                 // wakeup the producer
}
```

Protection & Security

Protection

- Protection : control access to system
  - ↳ which user can use which prog
- ↳ mechanism for enforcement of policies governing resource use

CPU, memory, printers, secondary mem → H/w objects + S/w objects ← files, processes, semaphores

If someone uses something not allowed ; policy says & deals with it.

Policy → **Mechanism**

what needs to be done → how to do it

If process is trying to access another's add. space
policy : not allowed, error should be raised, trap to OS

Access : Each object has an access right associated with it.
(what op's you can perform on it)

Domain :

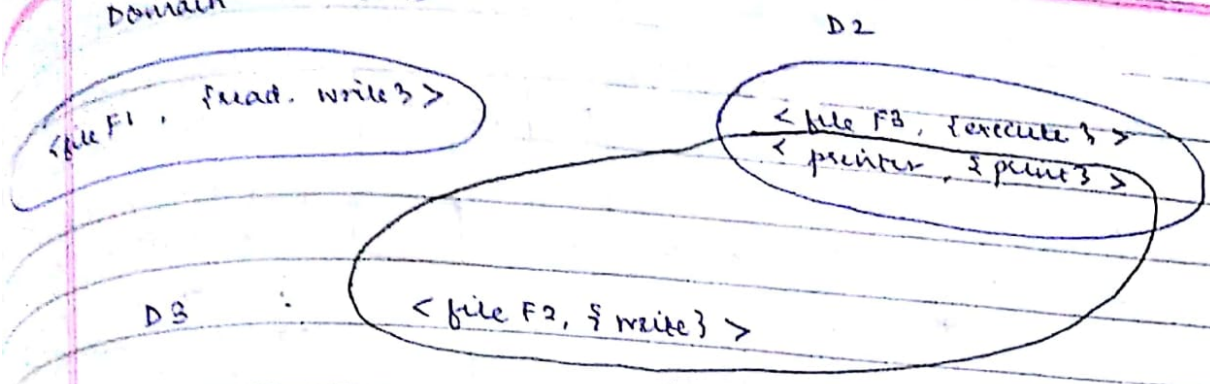
<object , access - rights>

eg. < file F1 , { read , write } >
< Printer , { print } >
} → domain

↳ work on principle of least privilege
A process should be given only those rights which it needs to complete its current task (min. rights)

Teacher's Signature

Domain 1 (D1)



↳ can be overlapping also

↳ let : U1 : work only in D1. → can only access F1 in RW mode.
→ can't use other resources

↳ P2 : works in D3 ⇒ can only access F2 & printer (W mode, print mode)

"Need-to-know" Principle : Only domain should be given to P/User which is min^m required by him at that time

Allocate Resources (Assigning D)

Static
D will remain with you throughout (which given to you in starting)
→ suppose need D1 → 10ms (1st)
D2 → next 10ms
but I'm giving all in starting
↓
violating Need to know Principle

Dynamic
When you need one, I'll give you that Domain

→ All this info. is kept in (Domain) Access Matrix

Proc 1

Objects / Domain	F1	F2	P1	D1	D2	D3
D1	read	write				
D2	read	read execute	owner	control		switch
D3	read*		execute		switch	
D4.			execute			

Can add col. if needed.

→ If I'm in D2, I can switch to D3

→ access right on an object in particular domain.

→ D2 can control D1.

↳ can specify domains

↳ Here, your permission is checked.

↳ Dynamic switching :→ may want to switch to D3 (to execute P1)

⇓

domains are also objects

access (i, j) = switch ⇒ If I am in Domain Di, then I can switch to Domain Dj.

→ Suppose I'm in Domain D3

read (*) → allows me to copy my access right in that column only.

⇓

I can give 'read' access right to any domain for F1

↳ transfer copy rights : give to someone else but I'll loose right

↳ Owner : who has all rights to Proc 1

↳ can give rights to diff. domains of Proc 1 (other or same)

Control : D2 can control D1.

⇒ D2 can change access rights, modify D1.

Security

Protecⁿ :

taking care of system.

Multiple user :

how to control

System Security :

Protect from external environment (hacker, intruder)

Threat : Possibility system may be ~~attacked~~ harmed

Attack : Someone is harming your system.

Breach of Confidentiality : unauthorized reading of data

Breach of integrity : " " modification

" availability : " destruction

Theft of service : " " use of resources

Denial of service (DOS) : preventing legitimate use of service

{ ~~not~~ & not allowing even those to use resources who're allowed }

Stack and Buffer Overflow

Everytime a funcⁿ is called, a frame is created

(COA)

```
main ( )                    → capacity : 9
{ char buf [10];
  char s;
  sf ("%s", &s);
  strcpy (buf, s)
}
```

not even checking size of str

malicious user : game to add.

Teacher's Signature

I've put : malicious code

converted into binary

all stored in stack & gave return add. at bottom of stack.

by debugging so much time so that I know where is return add. & what is bottom of stack add.

→ pretend to be someone else & get info. :→

to avoid

use cryptography (encrypt your msg)

K - set of keys

M - " msgs

C - ciphertext (new msg after encryption)

E : K s.t. (M → C)

encrypt — using key — msg. become cipher text

D : K s.t. (C → M)

Decrypt

Encryption

Symmetric

Both are having same keys

1 K only

used for Encryption & Decryption

Asymmetric

Teacher's Signature

PAGE NO. :

1) Symmetric :

↳ same key will be used to encrypt & decrypt

↳ If this key is compromised (someone else knows this key) → someone else may know msg

↳ msg is gone

DES (Data encryption std.) :

takes 64-bit value

apply 56-bit key

R → also has same key to decrypt.

This is applied on block. : Block cipher (64-bit, ...)

↳ intruder

4 If I keep on checking msgs, I can very soon figure out key being used.

can use

Triple DES : Using 3 keys

E(K1)(m) : Encrypt msg with 1st key K1

D(K2) : decrypt using key K2

E(K3)( ) : send to you

E(K3)(D(K2)(E(K1)(m)))

Better version:

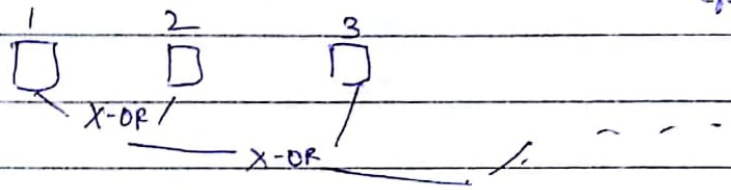
Advanced Encryp. Std. (AES)

Uses 128, 192, 256 -bit keys rather than 64-bit keys

↳ larger keys & blocks

Apply key on 1st & 2nd Block. Then, EXOR the blocks

(Keep on X-OR ing on subsequent blocks) → Block chaining (forms chain)



↳ Becomes difficult for intruder to find out

↳ Apply key of same length

→ Problem : If key is compromised → msg is gone

2.) Asymmetric

RSA (Algo most commonly used):

Telling everyone which key is used for encryption

↓

Public key

Asymmetric ← (K_e → Public Key. (Everyone knows)

K_d → Key for decryption : Only 1 person knows (Reciever)

Private Key

Encrypt :→ $E(K_e, N) = m^{K_e} \bmod N = C$ cipher text

Decrypt :→ $D(K_d, N) = C^{K_d} \bmod N$

Teacher's Signature

$N = p \times q$

↳ very large prime no. (512 bit each)

this cond^n should be satisfied :

$$(K_e K_d) \bmod ((p-1)(q-1)) = 1$$

↳ Suppose $p = 7$, $q = 13$

$N = 91$

$(p-1)(q-1) = 6 \times 12 = 72$

↑ choose $K_e = 5 \rightarrow$ prime — gcd = 1

$5 \times K_d \,\%\, 72 = 1$ $\quad \frac{145}{5} = 29$

$\Rightarrow \quad 5 \times K_d = (72 \times i) + 1$

$\Rightarrow \boxed{K_d = 29}$

$E(5, 91)$

$\Rightarrow (29, 91)$

Encrypt : $E(69) = (69^5) \,\%\, 91 \Rightarrow 62$

decrypt $D(62) = 62^{29} \,\%\, 91 = 69$

Ques Alice generates RSA keys by selecting $p = 11$ & $q = 13$. She chooses 7 for her RSA public key (K_e). Bob wants to send plain text msg no. 9 to Alice.

$N = 11 \times 13 = 143$

$(7 \times K_d) \,\%\, (120) = 1$

120 | 49

240

360 | 121

480