OS

Process Synchronization

① Mutation :- If one process is using CS than other process can't use that CS.

② Progress: If no any process in CS & at least one process wants to enter in CS. then it should be allowed.

③ Bounded waiting :- waiting of process should be bounded, process can't wait for ∞ time.

Solution for Synchronization

① using lock (S/W solⁿ)

```
Booleam lock = ~~true~~ false;
while(true){
    while(lock);
    lock=true;
      CS
    lock=false;
      RS
}
     P0

while(true){
    while(lock);
    lock=true;
      CS
    lock=false;
      RS
}
     P1
```

mutation Execution	✗
Progress	✓
Bounded wait	✓

② using turn [strict altration] (S/W solⁿ)

```
int ~~turn~~=0;
while(true){
    while(turn!=0);
      CS
    turn=1;
      RS
}
     P0

while(true){
    while(turn!=1);
      CS
    turn=0;
      RS
}
     P1
```

mutation	✓
Progress	✗
Bounded wait	✓

③ Peterson Solution (s/w solⁿ) [two process solⁿ]

```
Boolean Flag[2];
int turn;
```

```
while(true){
    Flag[0] = true;
    turn = 1
    while (Flag[1] && turn == 1);
        CS
    Flag[0] = false
        RS;
}
                            P0
```

```
while(true){
    Flag[1] = true;
    turn = 0
    while(Flag[0] && turn == 0);
        CS
    Flag[1] = False;
        RS;
}
                        P1
```

Mutation ✓

Progress ✓

Bounded ✓

④ TestAndSet() [Hardware Instruction]

```
Boolean Lock = False;
Boolean TestAndSet(Boolean *trg){
    Boolean rv = *trg;
    *trg = True;
    return rv;
}
```

$\boxed{\text{Atomic}}$

```
while(true){

    while (TestAndSet(&Lock));
        CS
    Lock = False;

}

            P0
```

⑤ Swap()

```
Boolean key, lock = False;
void Swap(Boolean *a, Boolean *b){
    Boolean temp = *a
    *a = *b
    *b = temp
}
```

```
while (true){
    Key = True
    while(Key == True)
        Swap(&Lock, &Key);
    ~~Key~~ CS = True
    Lock = False;
    RS
}
```

Bounded Buffer Problem

Mutex = 1 → to lock on buffer (Binary Semaphore) [mutual Exclusio]
Full = 0 → counting semp., to # of occupied slots.
Empty = n → " ", to # of empty slots.

```
Producer(){
    wait(empty) // to check if empty space available
    // Produce item
    wait(mutex) // buffer is shared, only one access at time.
      // add item on
      // buffer
    signal(mutex)
    signal(full)
}
```

Note:
Full ~~कितने~~ कितने भरे है?
Empty ~~कितने~~ कितने खाली है।

```
consumer(){
    wait(full)
    wait(mutex)
      // remove an item from buffer
    signal(mutex)
      // consume the item.
    signal(empty)
}
```

Empty used for not produce if buffer is full
full used for not consume if buffer is empty

mutex used for protect Buffer to avoid ~~an~~ mutual exclusion

Table:- Properties of all the solution

Solution	Mutual Exclusio	Progress	Bounded wait
Lock variable	✗	✓	✗
Stric Alternation (Decker's Algo)	✓	✗	✓
Peterson Algo	✓	✓	✓
TSL Instruction Set	✓	✓	✗

Reader - writer problem -

→ for Protection of readcount variable

mutex = 1 → Bin. Sem. for Mutual Exclusion

wrt = 1 → Bin sem. to restrict readers & writers on writing

read count = 0 → Integer variable, no. of active readers.

```
writer(){
 wait(wrt)
   // perform writing
 signal(wrt)
}
```

```
Reader(){
    wait(mutex)
    readcount ++
    if(readcount == 1)
        wait(wrt)
    signal(mutex)
      // perform reading
    wait(mutex)
    readcount -- ;
     if(read count ==0)
        signal(wrt)
    signal(mutex)
}
```

Dining philosopher problem

chopstic[5]={1,1,1,1,1}
// Bin. Sem.

~~oo~~.

```
 // all will run
wait(chopstick[i])
wait(chopstick[(i+1)}%k])
  // eat
signal(chopstick[i])
signal(chopstick[(i+1)%k])
```

```
// 1 philosopher will run
  wait(chopstick[(i+1)%k])
  wait(chopstick[i])
    // eat
  signal(chopstick[(i+1)%k])
  signal([chopstick[i])
```

Job with same Burst length, ~~and~~ arrive at same time then RR scheduler not able to provide better average turnaround time than FIFO

Process schudling

Note:- In R.R, If Arrival time is zero of all process P, Q and R then they will schudle at this order. | P | Q | R | P | Q | R | - - -

Note:- If Burst time is given + no preemtive than for min. avg waiting time, use $\boxed{\text{SFJ}}$ else use SFRJ

$\boxed{TAT = CT - AT}$ $\boxed{WT = TAT - BT}$

$\boxed{RT = FR - AT}$ $\boxed{\text{Response Ratio} = \frac{W.T + B.T}{B.T}}$

used in Highest Response ratio next

$$\boxed{\text{Throughput} = \frac{\#\text{ of process}}{\text{Max}(CT) - \text{Min}(AT)}}$$

term

TAT - Turn around Time	BT = Burst time
CT - complet time	RT = Reponse Time
AT - Arrival time	FR = First response time
WT - Waiting time	

Note:- If we have to use LRTF, and arrival time is zero and sum of Burst time is B the Turn around of prosses from higher ~~lower~~ ID to lower ~~higher~~ ID will Be B, B-1, B-2 . . .

Note:- Response time:- The time difference b/w first response and arrival time.

note :- SJF < SRT
less min   more min

take minimum average, W.T, TAT, TAT...
out of all algo.

Schudles

- Long term schudler $$\begin{tikzcd} \text{New} \arrow[r] & \text{Ready} \end{tikzcd}$$
- Medium " " $$\begin{tikzcd} \text{Suspend Ready} \arrow[r, leftrightarrow] & \text{Ready} \end{tikzcd}$$
- (swapping) $$\begin{tikzcd} \text{Suspend wait} \arrow[r, leftrightarrow] & \text{wait or block} \end{tikzcd}$$
- Short " " $$\begin{tikzcd} \text{Ready} \arrow[r, leftrightarrow] & \text{Running} \end{tikzcd}$$

Starvation table

FCFS	NO	every process get chance due to arrival time
NP, SJF	yes	if shorter complete and suddenly shorter come in ready.
SRJF	yes	same as SFJ
RR	NO	Every process will run. as per arrival time
NP, LJF	yes	if longer complete and suddenly long come.
LRTF	NO	Memory is limited, It will run all. till they got short time.
NP, Priority	yes	if more priority come again and again
Pr, Priority	yes	same as above.
HRRN	NO	Response ratio will calculated after seeing waiting time
MLQ	yes	
MLFQ	yes	

RR is better than FCFS in terms of Respons time

Reducing the quantum length of a RR will tend to improve the responsiveness of scheduled job.

For a given set of Jobs, all non-pre-emptive scheduling policies will require the same amount of context switch time overhead.

Process

- Program under execution
- contains stack, heap and data section / global section

Program →

① Passive & static
② It resides in secondary memory
③ File containing l

OS - Introduction & Background
[src: made-easy notes]

multiprogramming OS :- If one Job is leaving the CPU to perform I/O operation, then other Jobs which is ready for execution will be scheduled onto the CPU.

Multi-tasking :- multiple Job executed time-sharing mode.

Synchronous I/O :- Process perform I/O operation in block state.

Asynchronous I/O :- process is not placed in the blocked state.

Fork -

```
main(){
  int pid;
  pid= fork()
  ---
  ===
}
```

- Fork return, -ve value to parent if child process creation process is unsuccessful
- " " , 0 value to newly created child process
- " " , +ve value (process ID of child) to parent process.
- Parent and child process have same virtual address, but physical address is different.

Note :- If program has, 'n' fork calls then, there will be $2^n - 1$ child process created

Dead lock

Condition for ~~mutual Exclusion~~ Deadlock

1. Mutual Exclusion
2. Hold & wait
3. No-preemption
4. circular wait.

Recovery from Deadlock

1. Make sure that deadlock never occur.
   - Prevent the system from deadlock or avoid deadlock
2. Allow deadlock, detect and recover
3. Pretent that there is no any deadlock

Some topic notes

→ one switching two kernal theads in context switching register, PC and SP must be changed, but memory context remain same

→ user to kernal mode can occur by interrupt or system call.

→ Multilevel feedback Queue

$$\text{short I/O} > \text{short CPU} > \text{long IO} > \text{long CPU}$$

$$\xleftarrow[\text{Priority}]{}$$

→ On Reciveing Interupt, kernal will give sudden interrup serviced

Memory management

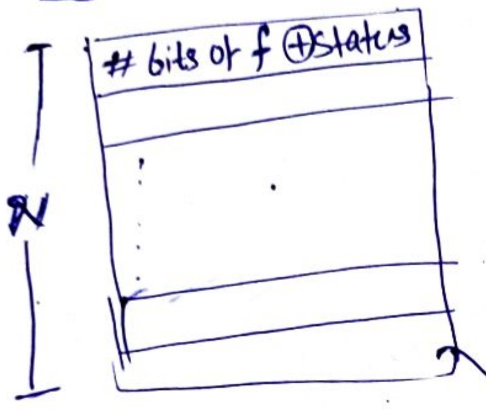
In Virtual address

Page number	Page offset

In Physical address

frame number	frame offset

Page Table format



let P = Page Number
f = frame number
O = offset
$N = 2^P$
$M = 2^f$

→ Page Entry size n

$$\boxed{n = \#\text{ bits for } f + \text{status bits}}$$

$$\text{size of table} = N \times n \Rightarrow \boxed{2^P \times n \text{ bits}}$$

TLB - Table look aside buffer

Tag	Index	Offset

~~Tag = V.A bit + log (Association)~~

$$\left(\text{Index} = \log\left(\frac{\text{line Number}}{\text{Associative}}\right)\right)$$

$$\left(\text{Tag} = V.A + \log(\text{Asso}) - \text{Index}\right).$$

$$\boxed{\text{TLB Entry} = \text{Tag bit} + P.A + \text{status bit}}$$