OS

1

Process Synchronization

1. **Mutation** :- If one process is using CS then 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)

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

$\boxed{P_0}$

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

$\boxed{P_1}$

mutation Execution	×
Progress	✓
Bounded wait	✓

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

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

$\boxed{P_0}$

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

$\boxed{P_1}$

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

must be same

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

P1

YES Mutual Exclusion
YES Progress
YES Bounding Wait

Mutation ✓
Progress ✓
Bounded ✓

④ Test And Set() [Hardware Instruction]

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

Atomic

```
while(true){
    while (TestAndSet(&Lock));
        CS
    Lock = false;
}
```

P0

YES Mutual Exclusion
YES Progress
NO Bounding Wait

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);
    CS
    Lock = False;
    RS
}
```

YES Mutual Exclusion
YES Progress
NO Bounding Wait

Bounded Buffer Problem

Mutex = 1 → to lock on buffer (Binary semaphore) [mutual Exclusion]
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 फुल है?
Empty 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 mutual exclusion

Table:- Properties of all the solution

Solution	Mutual Exclusion	Progress	Bounded wait
Lock variable	×	✓	×
Strict 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 or 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.

```
// 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. than 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 $\boxed{SFJ}$ else use SFRJ

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

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

used in Highest Response ratio next

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

term

TAT - Turn around Time
CT - complete time
AT - Arrival time
WT - Waiting time

BT = Burst time
RT = Response Time
FR = First response time

Note :- If we have to use LRTF, and arrival time is zero and sum of Burst time is B the ~~Ta~~ Turn around of processes 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   New → Ready
- Medium " "   Suspend Ready ↔ Ready
- (swapping)   Suspend wait ↔ wait or block
- Short " "   Ready ↔ Running

Starvation table

FCFS - NO - every process get chace 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 calculted 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

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

Short I/O > short CPU > long IO > long CPU

$\xleftarrow{\text{Priority}}$

- On Reciveing Interupt, kernal will give sudden interrupt serviced

Memory management

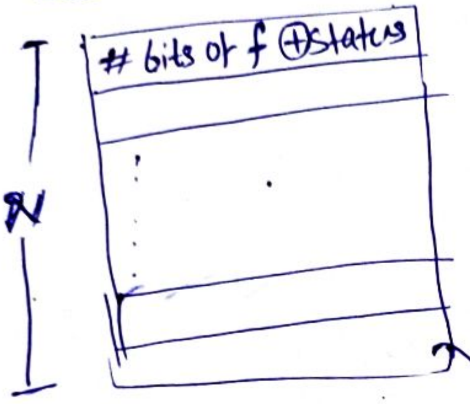
In Virtual address

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

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

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

// TLB - Table look aside buffer

Tag	Index	Offset

$\boxed{\text{Tag} = \text{V.A bit} + \log(\text{Association})}$

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

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

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