Process Synchronization

Notes by Rajin

Multiple process এর execution concurrently or parallely যে সমস্যার হয়, during their execution তাদের shared data or variable গুলো corrupted হয়ে যেতে পারে, যাবত Multiple process handle করা হচ্ছে তবে যাবে, তার এই সমস্যাকে বলা হয় Data integrity. এই Data integrity problem prevent করার জন্য Process synchronization concept introduce করা হয়।

Concurrent execution proride করার জন্য CPU scheduler rapidly switch করে process গুলোর মধ্যে।

A process may be interrupted at any point in its instruction stream.

Producer-Consumer Problem

একটি variable use করা হয় যার নাম counter. এবং এই variable টি Producer ও Consumer both side এর জন্য shared variable হিসেবে কাজ করে। Initially Counter = 0

Counter incremented every time when we add a new item to the buffer.

Counter decremented every time when we remove one item from the buffer or consumer consume the item from the buffer.

```
while (true){
    /* Producer produce an item
       in next_produced */
    while (counter == Buffer_size);
        /* do nothing */
    buffer[in] = next_produced;
    in = (in+1) % Buffer_size
    counter++;
}
```

```
while (true){
    while (counter==0);
        /* do nothing */
    next_consumed = buffer[out];
    out = (out+1)% Buffer_size;
    counter--;
    /* consume the item in
       next_consumed */
}
```

Producer and consumer Parallely or concurrently তাদের code execute করতে পারে।

Parallel execution এর time এ যদি Producer এবং Consumer একসাথে Counter এর value increment/decrement করে তখন Counter এর data integrity maintain হবে না, এটাই Data integrity problem.

"Counter++" and "Counter--" in machine language →

register1 = counter	register2 = counter
register1 = register1 + 1	register2 = register2 - 1
counter = register1	counter = register2

Race Condition →

#Several Process access and manipulate the same data concurrently.

#Multiple process যখন same time এ কোন একটা shared variable কে একই সাথে modify করে, এই situation কে Race condition বলা হয়,

* Race condition issue কে solve করতে হবে →
  যখন কোন একটা process shared variable কে modify করবে তখন এই time এ অন্য কোন process এই particular shared variable কে modify করতে পারবে না,
* Processes should be synchronized.

Critical Section

* কোন একটা Process এর code এর যেই section যা execute হওয়ার কারণে shared variables modify হয়ে যায়, সেই section টাই Critical section.
* যখন কোন একটা Process এর Critical section execute করে তখন অন্য কোন Process যেন তাদের Critical section কে execute করতে না পারে System কে সেটা ensure করতে হয়।

Entry Section

কোন একটা process এর Critical section এ যাওয়ার ঠিক আগের few lines of code যেগুলো execute হয়ে হওয়ার সাথে সাথেই process এর critical section execution start হয় অর্থাৎ যে lines of code গুলো দ্বারা System বুঝতে পারে process টা তার Critical section execute করতে চাচ্ছে, সেই few lines of code এর section কে entry section.

Exit section

Critical section থেকে বের হয়ে যাওয়ার code অর্থাৎ Critical section এর immidiate পরের কিছু code

Remainder Section

Critical section, entry section, exit section এর বাকি সব code গুলো হচ্ছে Remainder section

Requirements of solution to Critical Section Problem

1. Mutual exclusion: If a process is executing its critical section, no other process can be executing in their Critical section.
2. Progress: কোন Process ই তার Critical Section execute করে নাই, কিছু Process গুলোর মধ্যে কিছু Process আগে Critical section execute করতে চাচ্ছে, in that case, যে Process গুলো

তাদের Remainder section execute করে নাই কিনা সেটা তারাই decide করবে কোন process টা তার Critical section execution এ যাবে। এই selection process টা কোনভাবে রাখা যাবে না।

③ Bounded waiting: কোন একটা process একবার তার Critical section একবার execute করলে তারপর তার আবার waiting time limitation থাকবে, যাতে করে অন্য process গুলোকে খুব বেশি time wait করতে না হয়।

Critical Section in Operating System

OS এ Critical Section handle করার জন্য দুইটা general approach রয়েছে.

① Preemptive Kernel → Allows a process to be preemptive while it is running in kernel mode.

② Non-preemptive kernel → এই mode এ একটা process running অবস্থায় থাকলে, পুরোপুরি complete না হওয়া পর্যন্ত অন্য কোন process কে run করতে দিবে না

* Non-preemptive Kernel → free from Race condition

* Preemptive kernel mode এ Data integrity problem হতে পারে। তাই যদি OS এ Develop করতে হয় তাহলে অবশ্যই careful থাকতে হবে যেন Shared data. free from Race condition

Peterson's Solution for Critical Section Problem

* Software based solution.
* System এ যদি দুইটার বেশি Process থাকে তাহলে এই solution applicable না।
* Peterson's Solution Requires the two processes to share two data items:

```
int turn;
boolean flag[2];
```

* turn → turn এর value দিয়ে বুঝা যাবে দুইটা Process এর মধ্যে কোন Process টা আর critical Section execute করবে।
* flag → flag এর boolean value (true/false) দেখে বুঝা যাবে কোন একটা process আর critical section execute করার জন্য Ready or not.

Let,
দুইটা Process → P_i, P_j

```
do {
    flag[i] = true;
    turn = j;
    while (flag[j] && turn == j);
        critical section

    flag[i] = false;
        Remainder section

} while (true);
```

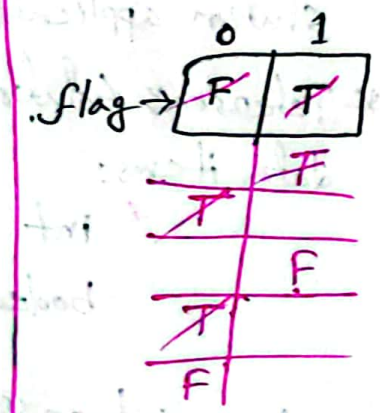
turn = [j]

flag = [T |]
i j
F

Example

→ Each statement takes 2ms to execute
→ Context switch will occur after 6ms
→ Critical section contains 4 statements
→ Remainde section contains 2 statements
→ turn = 0
→ Flag[0] = False, Flag[1] = True

turn → 0 0 1 1




```
do {
    flag[i] = true;
    turn = j;
    while (flag[j] && turn = j);
        Critical section
    flag[i] = false;
        Remainder section
} while (true);
```

Process 0 (i = 0, j = 1)	Process 1 (i = 1, j = 0)
	while loop condition CS1 CS2
stuck in while loop	
	CS3 CS4 Flag[1] = False
while loop condition CS1 CS2	
	RS1 RS2
CS3 CS4 flag[0] = False	
RS1 RS2	

→ Each statements takes 5 ms to execute

→ Context switch will occur after 20 ms

→ Critical section contains 5 statements

→ Remainder section contains 9 statements

→ turn = 0

→ flag[0] = true, flag[1] = false

turn → ~~0~~ 1 0

flag = | 0 : T | 1 : F |

T

T F

F

F

```
do {
    flag[i] = true;
    turn = j;
    while (flag[j] && turn == j)
        critical section
    flag[i] = false;
    Remainder section
} while(true);
```

PTTF TTTT

Process 1 (i=0, j=1)	Process 2 (i=1, j=0)
flag[0] = true turn = 1 while loop condition cs 1	
	flag[1] = true turn = 0 stuck in while loop
cs 2 cs 3 cs 4 cs 5	
	stuck in while loop

Process 1 (i = 0, j = 1)	Process 2 (i = 1, j = 0)
Flag[0] = false RS 1 RS 2 RS 3	
	while loop condition CS 1 CS 2 CS 3
RS 4	
	CS 4 CS 5 Flag[1] = False RS 1
	RS 2 RS 3 RS 4

Hardware based solution for Critical Section Problem

এই solution গুলোর locking concept এর উপর Base করে করা হয়,

Go through on the
slides for better
Understanding